THE WANDERING LOGIC INTELLIGENCE

A HYPERACTIVE APPROACH TO NETWORK EVOLUTION AND ITS APPLICATION TO ADAPTIVE MOBILE MULTIMEDIA COMMUNICATIONS

A DISSERTATION

SUBMITTED TO THE FACULTY OF INFORMATICS AND AUTOMATION

AT THE TECHNOLOGY UNIVERSITY OF ILMENAU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR-ENGINEER

IN COMPUTER SCIENCE

© Dipl.-Ing. Plamen L. Simeonov

2002

THE WANDERING LOGIC INTELLIGENCE

A HYPERACTIVE APPROACH TO NETWORK EVOLUTION AND ITS APPLICATION TO ADAPTIVE MOBILE MULTIMEDIA COMMUNICATIONS

Dissertation zur Erlangung des akademischen Grades
Doktoringenieur
(– Dr.-Ing. –)

vorgelegt der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau
von
Dipl.-Ing. Plamen L. Simeonov

Die Gutachten erstatteten:

1. Univ.-Prof. Dr.-Ing. habil. Dietrich Reschke, Technische Universität Ilmenau
2. Prof. Dr.-Ing. habil. Jörg Lange, Siemens AG, Berlin
3. Univ.-Prof. Dr. Ken J. Turner, University of Stirling, Scotland, UK

eingereicht am: 19.06.2002
Tag der wissenschaftlichen Aussprache: 18.12.2002

Ilmenau 2002

Abstract

This thesis is organized as follows.

Chapter 1 introduces the overall frame of the thesis: field of research, motivation, author's contributions and structure.

Chapter 2 presents an introduction and classification of the particular areas involved in this dissertation. Section 2.4 reviews the present research areas in wireless mobile communications with the objective to identify the application domain frame for this thesis work.

Chapter 3 is devoted to Active Networks (AN). We build this thesis work on this fundament. The review begins with introducing the reasons for the emergence of active software architectures followed by the challenges in modern communications to which they are exposed. Then, section 3.2 describes the conceptual paradigm of Active Networking, the underlying *Reference Model*[1], and a summary of the research approaches and the current implementation framework. Section 3.3 is dedicated to the active network architectures with the goal to identify the patterns and trends in AN research in order to derive a genealogy of the upcoming idea of the Wandering Network. The expose includes a short review of the enabling technologies for active networking highlighting their key advantages. Section 3.4 reviews the domains of AN research with a focus on a variety of applications. Section 3.5 identifies the mainstream directions in AN research. A special attention is dedicated to the application of active networks in mobile communications. Finally, section 3.6 provides an overall analysis and discussion of the active network approach including a comparison of the network programming approaches. An outlook for further research and a summary with conclusions are given in sections 3.7, 3.8 and 3.9 respectively.

Chapter 4 is devoted to the "hardware" branch of the Wandering Network hypothesis. Here we presents in detail some specific issues in Reconfigurable Computing as related to micro (chip) and macro (computer) component architectures used in present day networks. In section 4.2 special attention is devoted to the implications of active networking and reconfiguration in defining today's network infrastructures. Here we address some open questions from the previous three chapters: mixing active and passive flows, flexibility vs. security and configuring vs. encapsulation.

Chapter 5 presents the kernel of this thesis' research, the *Wandering Logic Intelligence (WLI)*. The exposé describes the general requirements and the future directions in active networking and related disciplines in order to define the architectural base of the WLI approach. The argumentation and the definition of the WLI architecture is given in section 5.2, followed by the four principles of the Wandering Network – *Dualistic Congruence*, *Multidimensional Feedback*, *Self-Reference*, and *Pulsating Metamorphosis* –, in section 5.3.

[1] which is referred in this work's genealogy as the *First Generation* (1994-1999).

Chapter 6 demonstrates the suitability of the WLI approach in a case study with the formal specification and test of an algorithm for active mobile ad-hoc routing in TLA^+, the Temporal Logic of Actions technique.

Finally, **chapter 7** provides an overall evaluation of the thesis objectives, answers some final questions and concludes this work with directions for further research.

Zusammenfassung

Diese Dissertation ist wie folgt organisiert.

Kapitel 1 beschreibt den Rahmen der Dissertation: das Forschungsgebiet, die Motivation für diese Arbeit, die Beiträge des Autors und die Darstellungsstruktur.

Kapitel 2 beinhaltet die Übersicht und die Klassifikation der einzelnen Bereiche, die Gegenstand der Dissertation sind. Sektion 2.4 untersucht die heutigen Forschungsgebiete in der drahtlosen mobilen Telekommunikation mit dem Ziel, die Anwendungsdomäne für diese Dissertationsarbeit zu identifizieren.

Kapitel 3 ist den Aktiven Netzen (AN) gewidmet, dem eigentlichen Fundament der Dissertation. Es beginnt mit einer Einführung in die Entstehung der aktiven Software-Architekturen. Weiter beschreibt es die Herausforderungen, denen sie sich in der modernen Telekommunikationswelt stellen müssen. Abschnitt 3.2 des Kapitels beinhaltet das konzeptionelle Paradigma der Aktiven Netze: das ihnen zugrunde liegende Referenz-Modell. Weiter folgt eine Zusammenfassung der Forschungsvorhaben und der gegenwärtigen Implementationsrahmen. Abschnitt 3.3 ist den AN-Architekturen gewidmet und identifiziert Muster und Tendenzen in der AN-Forschung, um eine Genealogie der Idee des Wandernden Netzes abzuleiten. Das Exposé enthält eine kurze Übersicht der Enabling-Technologies für Aktive Netzwerke, indem es einige Schlüsselvorteile betont. Abschnitt 3.4 umschreibt die AN-Forschungsdomäne mit Fokus auf verschiedenen Anwendungen. Abschnitt 3.5 wichtet die Hauptrichtungen in der AN-Forschung. Insbesondere werden die Anwendungen von Aktiven Netzen in der mobilen Telekommunikation betrachtet. Schließlich liefert Abschnitt 3.6. eine allgemeine Analyse und Diskussion der AN-Methode einschl. eines Vergleichs der Netzprogrammierungsarten. Einen Ausblick für die weitere Forschung und eine Zusammenfassung mit entsprechenden Schlussfolgerungen geben die Abschnitte 3.7, 3.8 und 3.9.

Kapitel 4 ist der Hardware-Seite der Hypothese über das Wandernde Netzwerk gewidmet. Einige Spezialthemen des Rekonfigurierbaren Computings, die bei Micro- (Chip) und Makro- (Computer) Komponenten-Architekturen in den heutigen Netzen genutzt werden, werden detailliert präsentiert. Besondere Aufmerksamkeit wird im Abschnitt 4.2 den Auswirkungen der Aktiven Netzwerke und des Rekonfigurierbaren-Computings auf die Definition der heutigen Netzinfrastrukturen gewidmet. Hier werden einige offene Fragen aus den vorangegangenen drei Kapiteln erörtert, wie das Mischen von aktiven und passiven Flows, Flexibilität vs. Sicherheit und Konfiguration vs. Encapsulation.

Kapitel 5 stellt den Kern der Dissertation, die *Wandernde-Logik-Intelligenz (WLI),* dar. Das Exposé beschreibt die allgemeinen Anforderungen und die zukünftigen Perspektiven in Aktiven Netzen und verwandten Disziplinen, um die Architekturbasis der WLI-Methode zu definieren. Die Argumentation und die Definition der WLI-Architektur ist in Abschnitt 5.2 gegeben. Danach folgen in Abschnitt 5.3 die vier Prinzipien des Wandernden Netzes – *Dualistische Kongruenz, Multidimensionale Rückkopplung*, *Selbst-Referenz (Selbstbezug)* und *Pulsierende Metamorphose*.

Kapitel 6 demonstriert die Angemessenheit der WLI-Methode in einer Fallstudie mit der formalen Spezifikation und dem Test eines Algorithmus' für aktives mobiles ad-hoc Routing in der TLA$^+$-Technik (Temporal Logic of Actions).

Schliesslich liefert **Kapitel 7** eine allgemeine Auswertung der Thesen dieser Dissertation, atwortet einigen Abschlußfragen und beendet diese Arbeit mit Empfehlungen für die zukunftige Forschung.

Acknowledgements

I wish to express my heartfelt thanks to all those who have given me their true appreciation and support over the years to carry out the work that has ultimately become my dissertation.

My first dept is to my advisor, Prof. Dr.-Ing. habil. Dietrich Reschke, who offered me the opportunity to complete my work at the Technology University of Ilmenau and gave me a free hand with my research. He asked all the right questions and always demanded practical examples during the reviews of my work in progress. I am very grateful to his restless efforts in making me work towards bridging the gap between theory and practice. Thank you, Dieter.

To my other mentor and thesis referee, Prof. Dr. Ken J. Turner from the University of Stirling who made time and effort beyond any duty to help turn this work into a dissertation. I have been greatly influenced by his research style that teached me how to put things together in a scientific way. This work has benefited from his detailed comments on every aspect. Thank you, Ken.

I am also indebted to Prof. Dr.-Ing. habil. Jörg Lange from Siemens AG for his support during my stay at the company and for his prompt willingness to act as a second referee. His practical insights helped me answer some of the most critical questions in this work. Thank you, Jörg.

In addition, I would like to express my special thanks to Prof. Dr.-Ing. habil. Ilka Philippow, Prof. Dr.-Ing. habil. Wolfgang Fengler and Prof. Dr.-Ing. habil. Winfried E. Kühnhauser for their valuable advises, comments and for participating in my thesis committee.

Furthermore, I am grateful to my colleagues from the Dept. of Telematics of the Technology University of Ilmenau for their help and for creating a favorable working environment: Mrs. Elfriede Spors, Mrs. Silvia Benz, Mrs. Marion Koch, Dipl.-Ing. Claudia Bergmann, Dipl.-Inf. Martha Barberena Najarro, Dipl.-Ing. Peter Henkel and Dipl.-Ing. Martin Sauebrey. It is also my pleasure to acknowledge Mrs. Christa Kallenbach, Mrs. Katrin Dünkel, as well as Dipl.-Ing. Wolfgang Schulke and Prof. Dr.-Ing. Horst-Michael Groß for their support and for the excellent organization towards my thesis defense.

I would also like to thank Dr.-Ing. Günter Hübel, Dr.-Ing. Peter Jackisch, Dr.-Ing. Werner Horn, Dr.-Ing. habil. Dang Hoang Hai, Dipl.-Inf. Jörg Deutschmann, Dipl.-Inf. Thorsten Strufe, Dipl-Inf. Ralf Döring, and Ralph Mielentz for their help and stimulating discussions.

My deep appreciation to Dipl.-Phil. Helena Piprek for her true friendship over the years from whom I learned that there is no such thing as too much encouragement. Special thanks are due to my good friends: Commodore Alan John M. Donaldson, for his long-term intellectual companionship and unlimited source of humor, and Dr. Jayne Chace for her cheering enthusiasm and valuable advice in tough times. I also wish to thank to my dear friend Ursula Saar for her guidance and motivation along the way.

Finally, I am grateful to my parents for believing in me and to all my family members for their dedication and support during the years of proof.

TABLE OF CONTENTS

List of Tables

List of Figures

Number *Page*

" Since I essentially knew nothing, I had an almost completely free choice ... "

SIR FRANCIS CRICK, "WHAT MAD PURSUIT"
BASIC BOOKS, NEW YORK, 1988, PP. 15-16.

* * *

THESIS OBJECTIVES

In our view, active programmable networks provide a foundation for architecting, composing and deploying virtual network architectures through the availability of open programmable interfaces, resource partitioning and virtualisation of the networking infrastructure. A key challenge in this research is the synchronous development of programmable virtual networking environments based on configurable hardware architecture.

The basic objectives of this thesis work are summarized as follows:

1. This work regards next generation application-aware networks as adaptive systems consolidating both network element and infrastructure flexibility in software and hardware[2]. Thus, if a reconfigurable computing infrastructure is combined with adaptive, i.e. application and user specific (smart), and active (programmable) mobile networking, it will be possible to utilize almost all *degrees of freedom* (down to the gate and bit levels) in managing the network.

2. It is evident that the complexity of such architectures is permanently increasing. In fact, an always growing, evolutionary model of changing software and hardware comes into being. In order to cope with this complexity of the growing network, it is necessary to effectively deploy integrated evolutionary models of the network, capable to describe the desired properties in a dynamic, relational and interdependent way which can be easily supported by formal methods and tools for specification and verification of the underlying architectures and algorithms.

This thesis proposes a new theoretical and practical framework for designing evolutionary communication architectures and their services and applications. The leading ideas and scientific objectives of this work are summarized as follows:

- to propose and demonstrate a flexible mechanism for network evolution based on the emergence and movement of functional units within a given physical infrastructure;
- to discover a set of guiding design principles unifying the numerous ad hoc approaches in network evolution in a thorough logical framework providing *the best available flexibility* in software and hardware technology at a certain level of development;
- to answer possibly directly to the question *how to make a network* like the future Internet and how to let it develop in order to provide the desired performance, quality, security, etc. features of importance to all users.
- to define a highly flexible network model which is adaptable to a wide variety of tasks and applications;
- to provide the formal means for the specification and verification of the generic temporal properties of active mobile nodes and packets;
- to demonstrate the suitability of the theoretical framework on a practical example.

[2] In this context, the notion of *hardware* is also enclosing nanotechnology molecular structures .

The theoretical goal of this dissertation is to provide an *elaborated model* leading to a formal theory (f.f.s.) for the design and verification of evolutionary (step-wise) adaptive systems based on active networks and configurable computing devices by means of temporal logic.

Currently, routing issues in ad-hoc mobile networking are a difficult challenge for protocol designers, since rapid reconstruction of routes is crucial in the presence of topology changes. The primary concerns in ad-hoc mobile networks are bandwidth limitations and unpredictable topology changes. In such an environment, it is important to minimize disruptions caused by the changing topology for critical application such as voice and video.

Therefore, the practical goal towards this thesis work is directed is the proof of the hypothesis that AN technology as an integral part of our theoretical approach delivers an appropriate methodology for automating the process of *route adaptation*, and hence of propagating topology changes within a dynamically changeable network infrastructure

* * *

CHAPTER 1: THESIS FRAME

1.1 INTRODUCTION

Active Networks (AN[3]) have been a subject of intensive empirical research for more than a decade, [Tenn99]. Figure 1 illustrates the main milestones in the evolution of active networking.

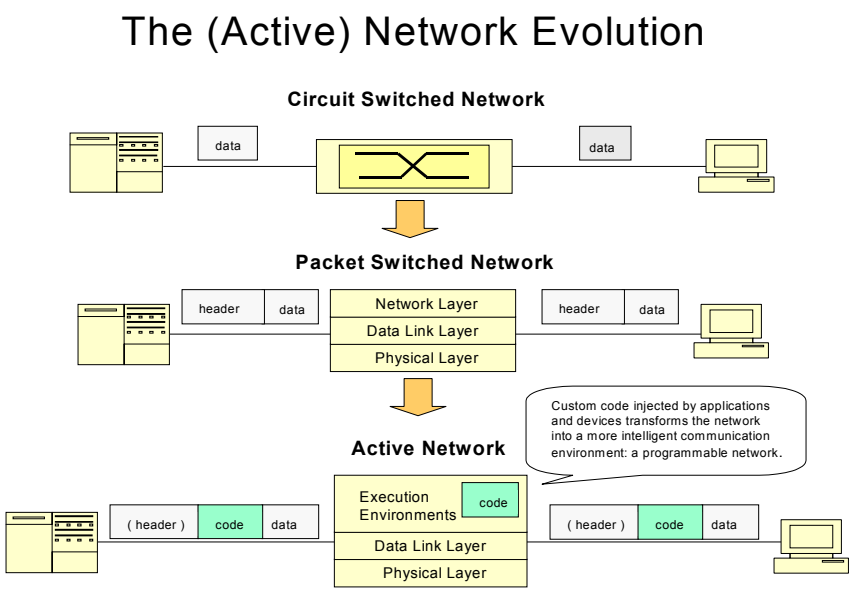


Figure 1: The Genesis of Active Networking

In this work, we propose a possible next step in this process which can be simply denoted as a shift from *user activity* to *network autonomy* in terms of self-organization, self-maintenance and even self-creation and self-assembly as a typical characteristic not only of biological and artificial life systems ([MaVa80], [White00],[Sim01]).

For the time being, a few general implementation strategies of Active Networking have been identified. Practical automation methods include such methods as multiple platform and system code support, node interoperability based on intermediate instruction encodings, on-the-fly compilation for optimization of common processing routines and operating system support for more specific strategies, such as path–based scheduling, protocol code reorganization, and low-level extensibility.

[3] The explanation of unfamiliar abbreviations can be found in the glossary at the end of this thesis.

In addition, research issues such as feasibility of capsules and code distribution schemes, security and performance optimization, resource management and fast introduction of new services have been on schedule.

Most AN approaches investigate and implement to some detail the above technical issues within a specific solution. A few survey papers were published trying to provide directions and goals for engineering within the field ([Tenn97], [Weth99a], [BWG99]). These efforts were mainly focused on defining a framework for *a common programming model* of active networks.

Recently, an integration and consolidation of the several different AN engineering approaches can be observed. This trend is particularly evident at technology frontiers with some innovative research fields such as deeply embedded networked systems, autonomous software, configurable computing, adaptive systems, etc.

However, implementations have shown that every single network issue such as caching, routing, management, etc. can have a specific *active* network or open signaling solution. A number of basic requirements and concepts for enhanced *virtualization* has been collected to (self-) activate networking. Yet, there is still no general recipe to address all the problems with only one end-to-end active or programmable network. Despite the broad interest in the subject, [ChJa98], the "killer" network of the future has not been found yet.

According to a DARPA review [Press99], XXI century networks will be:

- *active* (i.e. programmable w. r. t. network management with dynamic distributed entities),
- self-*reconfigurable,*
- ad-hoc mobile,
- *self-organizing*, and
- *truly* open, i.e. security with no firewalls but traffic monitoring, (e.g. DREN[4])

This work is an attempt to address these future aspects of networking in a structured fashion with the intention to improve and accelerate evolutionary network design. The goal of this thesis is to unify the numerous ad-hoc approaches for network design by defining an overall conceptual framework for the specification and verification of (step-wise) evolutionary, *autopoietic*[5] (i.e. self-creating, [MaVa80]) communication architectures based on: i) active networks ([Tenn97],[Calv98],[Camp99a]), ii) reconfigurable computing ([CoHa99], [BoPr00]), and iii) adaptive systems, [ACS].

The three research fields were brought together in the working hypothesis for the following reasons. Firstly, active networking defines the *principle*, the strategy and the goal of our research. Secondly, configurable computing brings up the required *detail* and understanding within a context, the tactics and the approach. Thirdly, adaptive systems contribute to understanding the large field of heuristic approaches and techniques in AI for the purpose of organizing and optimizing *wandering* communications.

[4] Defense Research and Engineering Network, http://www.hpcm.dren.net/Htdocs/DREN/DREN-HPCMO/.

[5] Appendix D provides some basic definitions in Autopoietic Theory.

Finally, mobile wireless multimedia communications provide a challenging perspective for applying our model for the design and verification of autonomous adaptive architectures ([Sim94a], [SpMe99], [Sim00]).

The following section presents the motivation for this work along with the scope, the depth and the objectives of the dissertation, followed by a review of the author's contribution.

1.2 MOTIVATION

Active Networks (AN) pursue a similar role in the data network domain like the Intelligent Network concept in the telecommunications domain today: they were expected to provide the means for the rapid creation and introduction of new Internet services. For this reason, the pioneers of the AN idea suggested to shift more computation inside the network, thus compromising the "keep-it-simple" principle of present day Internet. Two key benefits were envisioned with the introduction of Active Networks, [TeWe96]:

- rapid introduction of *new applications* for multimedia and e-business communications;
- an acceleration of the pace of innovation by *decoupling services* from the underlying infrastructure.

After almost 10 years research substantial progress in this area was achieved. However, the practical realization of Active Networks generated much controversy because of the serious performance and security concerns raised by the presence of untrusted code within the network.

In order to build a real, working system, it was necessary to revise some of the original concept premises. In 1999, Wetherall reconsidered the vision for active networks in light of the experience in designing, implementing and using the ANTS toolkit [ANTS] which was based on the "capsule" design principle that adds extensibility for processing data by embedding code at the IP packet level, [Weth99b]. The evaluation was made in three areas that characterize a "pure" active network: i) *approach*: the capsule model of programmability; ii) *safety*: the accessibility of that model to all users; and iii) *applications*: prospective areas for practice.
The following three paragraphs summarize the results of this analysis:

- ***Approach***: Capsules have proved a worthwhile model as a "*clean means" for upgrading processing* along an entire network path. This deployment model is considerably more powerful than the administrative upgrades practiced today. The efficient implementation of capsules depends on the upgrade demand frequency, on the amount of code and time window for loading and on traffic patterns for which code caching is effective.

- ***Safety****:* The free network customisation by untrusted users has been partly successful. It is now possible to isolate different services from each other without trust or centralized control, but not to protect the network as a whole from untrusted services. To overcome this drawback in the general case, the capsule approach has fallen back on certification by a trusted authority until better solutions are found. However, this still allows easy software change and maintenance compared to standardization efforts today.

- ***Applications**:* It was found that the most compelling application of capsules is the *network layer service evolution*, rather than the migration of application code to locations within the network. Capsule code is well-suited to the task of introducing *many variations of a service*, which is valuable for experimentation. It is expected that capsule code will act in synergy with network embedded devices (caches and transcoders) to provide more effective network operation by careful *feedback*.

The first argument affirms that the capsule approach provides a useful practical solution for code mobility in networks. If we discard the second argument of the evaluation, the safety issue, we find out that the most promising application is the *network layer service evolution*, which can be based on a large scale of means and mechanisms for *traffic regulation feedback*.

Hence, we decided to continue our research from this point with the intentions: a) to enhance the capsule model and generalize these results for the AN paradigm as a whole, b) to expand the scope of network programmability and adaptability to the hardware layer, and c) to define a set of generic network design principles for the target application domain of *service-usability-based network evolution*.

1.2.1 SCOPE

The intention of this thesis is threefold:

1. to organize and classify the broad range of existing and emerging approaches in active networking from the viewpoint of a *reasonably* evolving network infrastructure w. r. t. market forces, effectiveness and usability by including the *new dimensions of open network programmability*, -- reconfigurable computing and adaptive systems, -- while focusing on the application domain of mobile communications;

2. to define a model framework for a new, more dynamic and *vivid* generation of active networks based on a set of generic evolution principles;

3. to demonstrate the application of the new model in mobile networking.

The major goal of this work is to propose and to demonstrate a simple and flexible mechanism for network evolution based on the emergence, change and movement of functional units within a given physical infrastructure, which is aware of its own boundaries.

`Why do we need a systematics in programmable networking ?`

Recently, an integration and consolidation of the several different AN engineering approaches can be observed. This trend is particularly evident at technology frontiers such as deeply embedded networked systems, autonomous software, network-aware middleware, reconfigurable computing, adaptive systems, etc. However, implementations have shown that every single network issue such as caching, routing, management, etc. can have a specific *active* network solution. A number of requirements have been collected to activate the network.

Yet, there is still no general recipe to address all problems with only one end-to-end active network. The “killer” network of the future has not been found yet.

Along with the growing scope and number of ad-hoc solutions to active networking, the demand for their systematic categorization, evaluation and integration within a common research framework becomes increasingly evident.

In particular, an *evolutionary* approach to active networking requires the development of common models for: a) the encoding of network programs in terms of mobility, safety and efficiency; b) the description and allocation of node resources; c) the built-in primitives and *behavioral* patterns available at each node.

`What do we expect from the classification of technologies ?`

By analyzing the diverse approaches to active and programmable networks, we are striving *to derive some common solution patterns*, which can be organized as a set of principles for active network design in order to develop a methodology for a particular problem domain.

`What is the purpose of the new network model ?`

The new network model has to be open, hierarchical and dynamically structured. It should be able to address specific problems in wireline and wireless communications architectures, services and multimedia applications in a unified fashion and with a great degree of differentiation, flexibility, and granularity both in hardware and software. The new model should be capable to describe new types of active mobile architectures, which are particularly characterized by:

- flexible, multi-modal[6] specialization of network nodes as virtual subnetworks;
- mobility and virtualization of the net functions as hardware und software;
- self-organization as multi-feedback-based[7] topology-on-demand and a tool of the user-oriented network evolution.

In addition, the network elements in such an autonomous network architecture should be able to host different functional modules, and perform diverse network roles simultaneously, e.g. depending on the actual user location and/or environment characteristics.

[6] The single network nodes can execute multiple functions, and thus perform multiple roles in the network, in parallel, such as e.g. a protokol booster, fusion server, etc., as sub-classes of the generic roles: server, client and agent [Sim99c]. These functions can be realized as programmable software of configurable, i.e. resident, or plug-and-play hardware.

[7] i.e. user, application/protocol and buddies as network elements and functions delivering information for the self-regulation.

Why did we select mobile networks as application domain ?

In their prospective review *"Next Century Challenges: RadioActive Networks*", [BWG99], Bose, Wetherall and Guttag stated that the key challenge in wireless networking were "*to utilize the spectrum as efficient as possible given the current channel conditions and in the most effective way for each application*". The authors argued that it is difficult to achieve this target by conventional radio technology because "the physical layer functionality is fixed, while channel conditions and applications can change rapidly". Accordingly, the *RadioActive Networks*, as an active offspring of the Software Radio approach ([Blu95]], [BoSh98]), were satisfying the requirement for an adaptable network architecture by drawing the strengths of software radio and active networks.

We join this opinion with the claim that RadioActive Networks and other technology innovations the area of wireless communications such as active ad-hoc mobile networks [Tschu99a] belong to the first efforts to define the experimental field of evolutionary network design. These research areas project the *living network* of the future by investigating the feasibility of *diverse short-term models for a long-term infrastructure*.

By addressing functional transformations in the communicating nodes on the border between hardware and software, we address the same goal in this thesis with the Wandering Network approach. For this reason, we selected mobile wireless communications as a target application domain. A case study, which demonstrates the capabilities of our approach, is given in chapter 6. Section 2.4 provides an overall review of the research in mobile networks.

1.2.2 DEPTH

The ultimate goal towards we align this research is the creation of a model-based formal theory for the design and verification of (systematic) evolutionary, *autopoietic* (i.e. self-creating, [MaVa80]) architectures based on active networks, reconfigurable computing ([CoHa99], [BoPr00]), and adaptive systems [ACS]. We brought these three research fields together in our working hypothesis for the following reasons. Firstly, active networking defines the *principle*, the strategy and the goal of our research. Secondly, configurable computing brings up the required *detail* and understanding within a context, the tactics and the approach. Thirdly, adaptive systems contribute to understanding the large field of heuristic approaches and techniques in AI for the purpose of organizing and optimizing *wandering* media communications. Finally, the application field of multimedia communications provides a challenging perspective on applying the WLI approach to the design and verification of autonomous adaptive architectures [SpMe99], [Sim94b], [Sim00].

Our medium and long-term goals, which are not part of this work, are summarized as follows:

- to provide a formal means for the specification and verification of the generic temporal properties of active mobile nodes and packets;
- to support the reflexive dynamic adaptation of both mobile code (software) and node architecture (software and hardware);

- to provide the formal means for specification and verification of dynamic QoS and routing properties in active ad-hoc mobile networks at both application (service) and packet level;
- to assist the formal transformation of systems' properties into mobile code.

1.3 CONTRIBUTIONS

The essential novelties, ideas and contributions of this thesis are listed as follows:

1.3.1 THE WANDERING LOGIC INTELLIGENCE (WLI)

The Wandering Logic Intelligence (WLI) represents a hyperactive approach for the high-level specification of adaptive and evolutionary communications systems. It provides an open, hierarchical and dynamically structured model which allows to address specific problems in communications architectures, services and applications with a great degree of differentiation, formalization and flexibility that can be tracked down to the gate level.

I. The WLI Framework:

- **An evolutionary adaptive system model**

- **Four basic design principles and their characteristics**:

    1. Dualistic Congruence
    2. Multidimensional Feedback
    3. Self-Reference
    4. Pulsating Metamorphosis
        - *Knowledge quantum*
        - *Genetic transcoding*
        - *Network resonance*

- **Generic architectures for *netbots* (active mobile nodes) and *shuttles* (active packets) which can transport executable *genetic code* about a node/network state/process.**

- **Extending adaptability to four dimensions of network reconfiguration and programming**:
    1. Applications
    2. Operating system resources
    3. Node hardware components
    4. Clusters of nodes

- **A two-level profiling scheme for functional components in a reconfigurable network architecture** characterized by:

- *horizontal inter-node functional wandering (self-organization)* of the active nodes (netbots), called 1st Level Profiling, and
- *virtual vertical intra-node overlay functional wandering (self-organization),* called 2d Level Profiling

as well as:

- two special functional roles of reconfigurable netbot in the 1st Level Profiling: `Replication` and `Next-Step` for packet / function replication and netbot state temporal description respectively;
- two special functional roles for `Routing` and `Propagation` of functionality in the 2d Level Profiling.

II. The Netbot Abstraction

- Active nodes can be *mobile* - hence the name ***netbots*** - and *reconfigurable* (in terms of software and hardware) during runtime. In WLI, reconfiguration is just another type of network service.

- Depending on the ***class*** of service to be installed via shuttles onto an active node, a special manipulation of the shuttle in the execution environment may be applied including cashing of its contents and/or state to ensure *awareness* about the flow. The *class* of service is a new concept in WLI used to describe *multiple code systems*, either at the "byte level", or at higher-layers associated with different service functions.

- A*ctive nodes (netbots) can adapt (themselves) to communications in such a way to best-match the structure of the active packets (shuttles) at the time of delivery: the Dualistic Congruence Principle*.

III. The Shuttle Abstraction:

- In WLI, ***shuttles*** can carry code and data (like capsules) not only for the execution within**,** but also for the upgrade/degrade and re-configuration of active nodes. Thus, shuttles can modify netbots. For this reason, the capsule APIs and the execution environments can be extended by special routines allowing the accommodation and execution of code that changes a netbot's configuration and resources. In this way, new functionality can not only be delivered to and injected into the active node, but also distributed and optimized throughout the node itself.

- In addition to the data related code contained in traditional capsules, shuttles provide a secondary layer of virtualization carrying ***genetic code*** which can be selectively invoked by special routines at the netbots to perform structural changes in the node/network architecture, e.g. to spawn/collapse a virtual (sub-)network, to expand/derive a new reachability (sub-)tree for routing, chapter 6.

- The WLI formalism allows the creation of new capsules/shuttles (or the replication of "old" ones) in the intermediate active nodes under the supervision of the node operating system (NodeOS). Furthermore, a special class of shuttles, called *pilots* are allowed to replicate themselves and to *create/remove/modify other capsules and resources* in the network. *Pilots* have network administrator authorities. The term "resource" can be extended to an entire *virtual* active node.

- In WLI, the network protocol itself can be particularly embedded within the shuttle. Furthermore, code distribution throughout the network and inside the nodes/netbots *can be maintained by the shuttles themselves.* A shuttle approaching a netbot can *re-configure itself* becoming a *morphing* packet to match the netbot's processing requirements: *the Dualistic Congruence Principle*.

The ***essential characteristics*** of the WLI approach in this thesis work are listed as follows:

1. *Role Change*: The *role* of the network node within a particular virtual architecture can change during its operation. The new functionality is either resident on the node and waiting to be activated, i.e. it is not yet involved in the next step virtual scheme, or *transferred* via Active Networking to the destination node.

2. *Parallel Roles*: The execution of the parts of a distributed algorithm can be performed within the different roles of an active node's/ netbot's, configuration.

- *Node Genesis ("N"-geneering):* encoding, embedding and transporting the structural information about a mobile node, the netbot, and its environment, such as e.g. rooting tree information, into a secondary virtual layer of the active packets / shuttles composed of node genes, "*n"*-genes. N-genes provide context-related information on *HOW* the basic code and data in the primary level of the shuttles should be processed.

WLI defines a novel generic model for specification and verification of autonomous, evolutionary, and in particular, *self-aware* active ad-hoc mobile wireless networks.

1.3.2 THE WANDERING NETWORK (WN)

I. We define **a new type of communication network model**, the *Wandering Network* (WN), based on a *WLI* framework which extends previous models of active programmable networks, by three essential characteristics:

   a. it is a *real* active network which means that it is truly programmable and reconfigurable, including the network hardware up to the gate level;
   b. it is a runtime extensible and exchangeable network in terms of both software and hardware components, i.e. a *wandering network*;
   c. it is an evolutionary network which realizes *adaptive* self-distribution and replication of sub-networks:

- by guided or autonomous node and component mobility in terms of hardware;
- by including essential engineering information in the mobile code of the active packets and applying *genetic transcoding* mechanisms in the active nodes (*netbots*).

II. We define **a genealogy classification system** for Wandering Networks to accommodate the diversity of architectures and models of active, programmable and reconfigurable networks developed until now and to devise directions for future research.

III. In a WN, network functions can change their hosts (netbots), wander and settle down in other hosts, thus creating a valuable statistics about the frequency of usage of wandering functions in the network. The results obtained after a careful evaluation of this data can be used for the (self-)design of new network architectures and topologies.

The Wandering Network approach differs from other research frameworks by two characteristics: a) *adaptable function migration,* and b) *pulsating metamorphosis.*

1.3.3 THE TWO-LEVEL RECONFIGURABLE INTRA-NODE PROFILING SCHEME

A netbot's internal functional organization according the WLI model (section 5.4, Figure 55).

1.3.4 THE SECONDARY SHUTTLE VIRTUALIZATION LEVEL

A shuttle's internal functional organization according the WLI model (sections 7.2.3 and 7.2.4).

1.3.5 THE WLI ROUTING ALGORITHM

The major contribution of this thesis is a theoretical framework for constructing formal models of mobile communication architectures and algorithms illustrated with the **WLI Adaptive Routing Algorithm for (Active) Ad-hoc Networks (WARAAN)**. It describes an autonomous adaptive routing architecture and protocol for mobile ad-hoc networks based on an instant auto-feedback for dynamic update of the reachability trees to compensate spontaneous network topology changes. The following contributions are unique for this thesis work:

- Generic architecture of a WLI router
- WLI routing model
- Routing methodology
  - General conditions
  - Target oriented strategy decision
  - Maintenance
- Reachability tree generation and maintenance
- Propagating of the reachability information in the ad-hoc network
- Routing algorithm: formal specification and test in TLA.

The new model provides a unified and structural approach for a flexible intelligent network of the new generation. This solution is not only applicable for out-band signaling „intelligent networks“, but also for the new generation of the so-called „programmable“, active networks where extensions, new services and new versions can be easily installed and configured in a *usability* driven manner.

The main advantage of the Wandering Network approach is that it unifies the macro-world of active networking and the micro-world of configurable computing in a generic model, which reflects the intrinsic nature of intelligent communications. It facilitates the application design and allows sophisticated network growth, adaptation and rapid introduction of new services while supporting both engineering approaches by making only minor changes on the available infrastructure. In this way, the WN architecture represents a *vivid, scaleable object-oriented model* for intelligent service provisioning and control when compared to the traditional OSI-like layered network architectures.

1.4 THESIS STRUCTURE

This thesis is organized as follows, Figure 2. In chapter 2 we present an introduction and classification of the particular areas involved in this dissertation. Section 2.4 reviews the present research areas in wireless mobile communications with the objective to identify the application domain frame for this thesis work.

Chapter 3 is devoted to the Active Networks (AN). This is the fundament on which we build this thesis work.

The review begins with introducing the reasons for the emergence of Active Networks and with the challenges in modern communications they are exposed to. Then, section 3.2 describes the conceptual paradigm of Active Networking, the underlying *Reference Model*[8], and a summary of the research approaches and the current implementation framework. Section 3.3 is dedicated to the active network architectures with the goal to identify the patterns and trends in AN research in order to derive a genealogy of the upcoming idea of the Wandering Network. The expose includes a short review of the enabling technologies for active networking highlighting their key advantages. Section 3.4 reviews the domains of AN research with a focus on a variety of applications. Section 3.5 identifies the mainstream directions in AN research. A special attention is dedicated to the application of active networks in mobile communications. Finally, section 3.6 provides an overall analysis and discussion of the active network approach including a comparison of the network programming approaches. An outlook for further research and a summary with conclusions are given in sections 3.7, 3.8 and 3.9 respectively.

[8] which is referred in this work’s genealogy as the *First Generation* (1994-1999).

The Thesis Structure

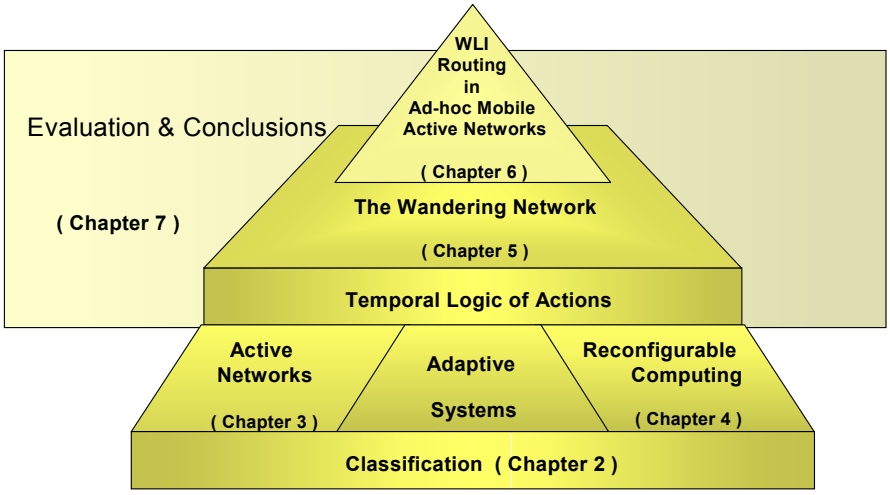


Figure 2: Exposé

Chapter 4 is devoted to the "hardware" branch of the Wandering Network hypothesis. Here we presents in detail some specific issues in Reconfigurable Computing as related to micro (chip) and macro (computer) component architectures used in present day networks. In section 4.2 special attention is devoted to the implications of active networking and reconfiguration in defining today's network infrastructures. Here we address some open questions from the previous three chapters: mixing active and passive flows, flexibility vs. security and configuring vs. encapsulation.

Chapter 5 presents the kernel of this thesis' research, the *Wandering Logic Intelligence (WLI)*.

The exposé describes the general requirements and the future directions in active networking and related disciplines in order to define the architectural base of the WLI approach. The argumentation and the definition of the WLI architecture is given in section 5.2, followed by the four principles of the Wandering Network - *Dualistic Congruence*, *Multidimensional Feedback*, *Self-Reference*, and *Pulsating Metamorphosis* -, in section 5.3.

Chapter 6 demonstrates the suitability of the WLI approach in a case study with the formal specification and test of an algorithm for active mobile ad-hoc routing in TLA$^+$, the Temporal Logic of Actions technique. Finally, chapter 7 provides an overall evaluation of the thesis objectives and concludes this work with directions for further research.

CHAPTER 2: BASIC DEFINITIONS AND CLASSIFICATION

"But things are *defined by* their working and power;
and we ought not to say that they are the same
when they no longer have their proper quality …"

ARISTOTLE (384-322 BCE.), *POLITICS*

Along with the growing scope and number of ad-hoc solutions to active and programmable networking, the demand for their systematic categorization, evaluation and integration within a common research framework becomes increasingly evident. This chapter provides some basic descriptions and a classification structure of the target domain to be used in the sequel.

There are two modern schools of thought addressing the next generation communication networks today[9]: *Active Networks* (AN), [DoDAN], and *Open Signalling* (OPENSIG), [OPENS], both referred to as *Programmable Networking*[10]. The OPENSIG approach has a telecommunications background and thus clearly separates *network control* from *information transport.* It has been primarily focused on programmable ATM switches, and recently --- on IP routers and mobile networks that provide some level of QoS support. The AN approach, in contrast, has been historically focused on IP networks, where the *control and data paths are combined*. Both communities share the common goal to go beyond existing approaches and technologies for design, deployment and management of new network services including a broad spectrum of projects with diverse architectural proposals. However, the OPENSIG advocates argue that open access to switches and routers can be provided by modelling communication hardware as distributed computing objects[11] using a set of well-defined and standardized *open programmable network interfaces* [P1520] which allow service provides to manipulate the states of the network and create and manage new services using middleware architectures such as CORBA [Vin97]. On the other hand, the active networks community promotes the dynamic deployment of new services at runtime mainly within the existing IP networks and by means of mobile code (not signalling) as the main vehicle for program delivery, control and service creation.

The OPENSIG approach, which is crucial for making the network more programmable, pursues the traditional view of separation between the different network layers. However, in Active Networks the granularity of control can range from the packet and flow levels through the installation of completely new *switchware* [Alex98a]. The term '*granularity of control*' was defined by Calvert et al. [Calv98] and refers to the scope of switch/router behaviour that can be modified by a received packet. For instance, a single packet could boot a complete software environment seen by all packets arriving at the node, or the switch/router behaviour seen only by that packet.

---

[9] [Camp99a] and still by mid July 2001.

[10] in terms of *software (!);* later in this work we will see that hardware can be also regarded as programmable.

[11] e.g. virtual switches[Chan96], switchlets [Merw97a] and virtual base stations [ACKL98]

Thus, Active Networks allow the customisation of network services at packet transport granularity, rather than through a programmable control plane and thus offer the maximum flexibility in support of service creation, yet at the cost of additional complexity to the programming model.

Therefore, the AN approach is an order of magnitude more dynamic than the OPENSIG's quasi-static network programming interfaces which require a new round of standardization every time a new software or hardware technology is going to be used. Nevertheless, the OPENSIG approach provides the basic idea of network interfaces that we use in combination with the AN framework to define the scope of an evolving new generation of networks, the *Wandering Networks*. Figure 3 illustrates the general unifying model of a programmable network[12] for both the telecommunications and the Internet worlds as it was given by Campbell *et al*., [Camp99a]. Since the rest of this work is going to be mainly focused on active networking technologies, we are adopting the terminology in the above reference which we call the *Columbia Model* with the consensus to use the terms "active" and "programmable" interchangeably.

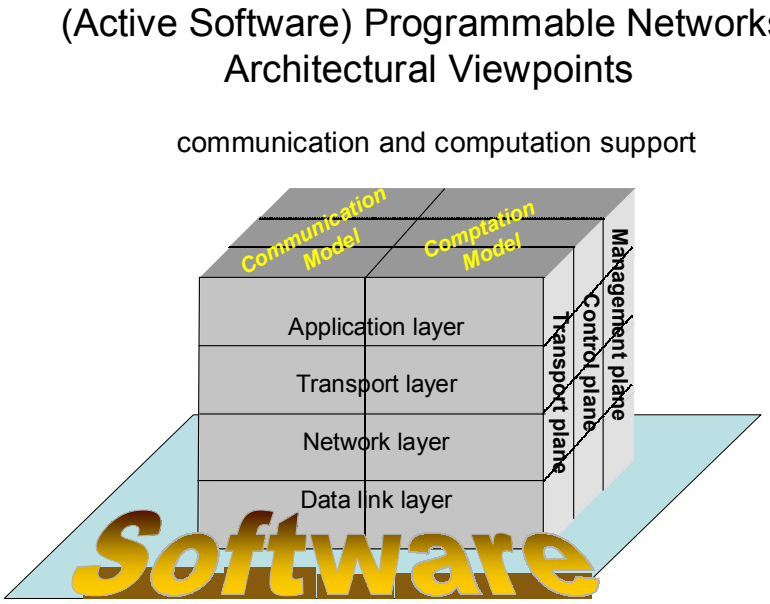


Figure 3: The basic views at (active software) programmable networks

The following sections introduce the three main directives in modern computer science and engineering which are going to play a fundamental role in forming the future concept and reality of *living* networking: programmable networks, reconfigurable computing and adaptive systems. They stem from different research areas – computer science, electrical engineering, and artificial intelligence, and deal with quite different domains of knowledge: communication protocols and services, VLSI logic processor design and information system modeling.

[12] The IP world is addressed by the front plane with the OSI layers, whereas the telecommunication world is given by the side plane. To emphasize the *programmability* of the network, the communication top plane has been extended with a computation model which means that the network nodes are *actively* performing operations on the packets.

Yet, recently they are increasingly sharing the same key problem: the *network evolution* while trying to answer the question: “How communication networks should be designed to allow easy and optimal upgrades in functionality and performance to provide reasonable and effective services in a rapidly changing environment at least adaptation and maintenance costs.” A more detailed presentation and discussion is following in chapters 3 and 4 respectively.

2.1 WHAT ARE ACTIVE NETWORKS ?

Traditionally, the function of a packet switched network has been to deliver packets from one endpoint to another by performing only the processing necessary to forward packets towards their destination and enabling the *sharing of transmission facilities* so that packets may be efficiently moved between the interconnected systems. The processing within the network has been limited to routing, congestion control and quality of service (QoS) management. This kind of a networking is known as “passive”.

Over time, as computing power becomes cheaper, more and more functionality is being placed inside the network, in nodes such as firewalls, Web proxies, multicast routers, and mobile proxies. Examples of such functionality include admission control (to guarantee delay and other performance characteristics for certain classes of users), explicit congestion notification (to enhance the congestion-adaptation of certain applications), packet filtering (to protect end systems from attempts to exploit security holes), TCP "ACK-spoofing" (to improve reliability over lossy links) and even content transcoding (to support the efficiency of multimedia applications).

Along this path, several problems with “passive” networks have been identified: slow pace of network innovation (approx. 10 years from a prototype solution through standardization to a large-scale deployment), difficulty to integrate new technologies and standards into the shared network infrastructure, poor performance due to redundant operations at several protocol layers, and difficulty to accommodate new services in the existing infrastructure.

In the absence of architectural support for providing better services to the users, network applications have adopted a variety of ad hoc services for performing *user-driven computations at the network nodes*. A need was felt to replace the numerous ad hoc approaches to network-based computation, with a generic capability that allows the users to configure and program their networks.

This innovative idea of enabling the user to configure and program the network architecture is called *Active Networking*. These *programmable* networks are “active” in two ways:

a) routers and switches within the network can perform computations on user packets flowing through them; and
b) users can “program” the network, by supplying their own programs to perform these computations.

The first approach is known as *Active Nodes* and the second one -- as *Active Packets*. In this work we are extending the general view of network programmability defined in [Camp99a] to the node hardware and down to the switching circuit layer, and hence --- to the concept of a *Real* Programmable or a Hyperactive Network (Figure 4), which we are going to call simply Active Network in the sequel. The details of this model will be discussed later in this work.

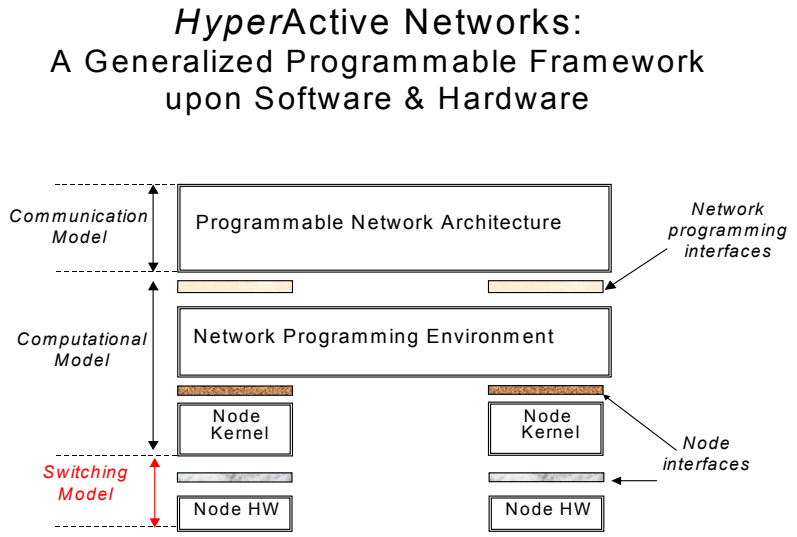


Figure 4: The *hyperactive* network concept

The present backlog of Internet services includes multicast, authentication and mobility extensions, Resource Reservation Protocol (RSVP), and Internet Protocol, Version 6 (IPv6). IP enables interoperability by defining a standard packet format and addressing scheme; although router implementations may differ, they implement roughly equivalent programs. Thus, the mechanisms for IP innovation are changing the IP service, which means changing everything (since IP is the basis for interoperability), or establishing overlays (e.g., the Mbone).

In contrast, active nodes can support many different protocols and execute programs on data flowing through them, both provided by users. Instead of insisting that all the routers perform equivalent computations on every packet, active networks specify that all nodes support equivalent computational models (i.e. virtual instruction sets).

Thus, active networks raise the level of abstraction to a *virtual-virtual*[13] *networking* in which even such different protocol worlds as IP and ATM are regarded as a sort of *transport* [Zegura96]. In this way, interoperability is realized in a straightforward manner allowing applications to customize message processing to suit their purposes. In fact, due to the transparent behavior towards "pure" IP traffic, active networks are acting as "shadow" overlays inside present day IP networks by enabling effective, *on-demand* service deployment and management mechanisms.

The dynamic control provided by active networks is potentially useful for a number of reasons. A few of them are described below:

[13] because *virtuality* is realized not only at the application layer, but also at the lower OSI layers.

- **rapid development and deployment of new services:** This is the biggest benefit of Active Networks from the viewpoint of the network service provider. The shared infrastructure of the network currently evolves at a much slower rate than other computing services. Therefore, the deployment of any new service requires a lengthy standardization process, whereby everyone involved should agree on a generic procedure for a solution framework (the standard). This has led to a huge backlog in network services waiting to be universally deployed. RSVP and IPv6 are two well-known examples. The ability to change the behavior of network nodes on the fly is expected to simplify greatly the process of deploying new network services.
- **dynamic customization of network services and resource allocation**: At a finer level of granularity, Active Networks might enable users or third parties to create and tailor services to their particular topology, applications and even to prevailing network conditions. This should make it possible to develop a much richer class of applications than the ones that are currently deployed.
- **open network management and administration**: Active Networks are open to deploy and administer. For researchers, a dynamically programmable network offers a platform for experimenting with new network services and features on a realistic scale without disrupting the regular network infrastructure.

The overall scheme of active networking provides the base for an elaborate evolutionary model of future integrated application networks.

In [Sim01] we defined a new network generalization for programmable active networks, which we call the *Wandering Network (WN)*. The new concept is based on previous research in intelligent and smart networking [Sim96], [SiHo97], [Sim98], [Sim99c] and a formalism called *WLI* (the Wandering Logic Intelligence) [Sim99e], [Sim99f] which extends the Columbia University Model of a programmable network, [Camp99a], by three essential characteristics:

1. it is a *hyper*active network which means that it is truly programmable and reconfigurable, including the network hardware up to the gate level;
2. it is a runtime extensible and exchangeable network in terms of both software and hardware components, i.e. a *wandering network*;
3. it is an evolutionary network which realizes *adaptive* self-distribution and replication of sub-networks:
    a. by guided or autonomous node and component mobility in terms of hardware;
    b. by including network engineering information in the mobile code of the active packets and applying *genetic transcoding* mechanisms in the active nodes (*netbots*).

Essentially, we distinguish between four generations (Figure 5) of Wandering Networks (WN). The First Generation WN includes most of the traditional active network approaches as known to be programmable at the highest executing environment (EE) layer. The Second Generation WN addresses programmability at both EE and node operating system (NodeOS) layer. Some of the present AN systems can be classified to the 2G WN. The Third Generation WN addresses programmability at the last layer of networking, an active node's hardware and switching circuitry, in addition to the 2G WN capabilities. Some of the present AN systems such as *ANON* [Tschu97], *Tempest* [Merw97a], and *Genesis* [Camp99b] can be classified to the 2G WN.

We are not aware of any end-to-end network architecture that could be classified to the 3G WN[14]. Finally, the 4th Generation Wandering Networks is defined by adaptive self-distribution, replication and genetic transcoding techniques. This is the research domain of this work.

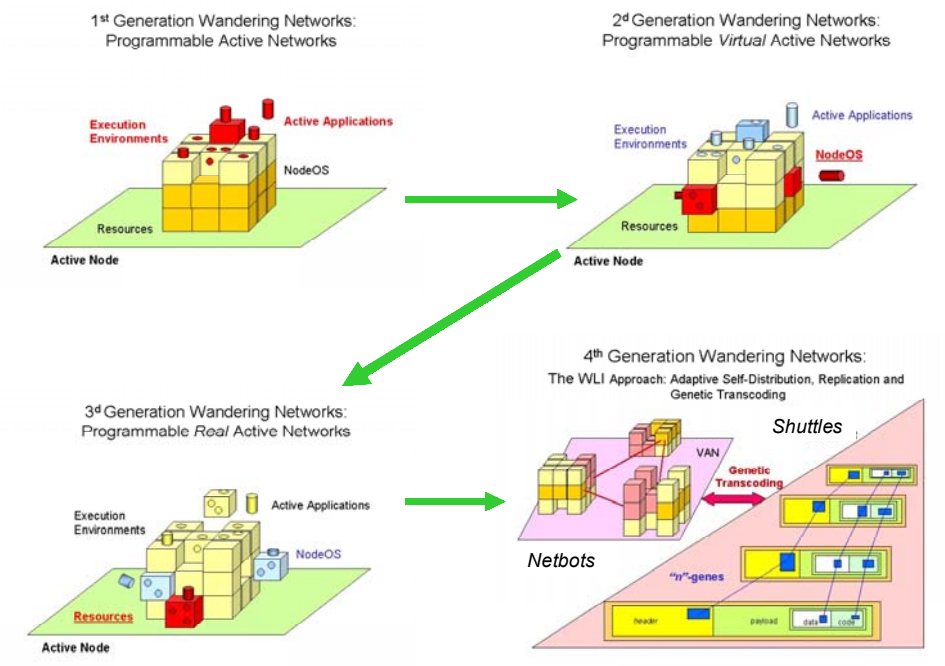


Figure 5: The evolution of Wandering Networks

2.1.1 PRESENT ACTIVE NETWORK MODELS

Active Networks (AN) has been a subject of intensive empirical investigation for more than a decade[15]. A number of different models have been proposed to implement active network architectures. Three major schools have been established until now:

1. *The "programmable switch" model:* User code is downloaded out-of band into the network nodes and executed on the normal flow packets treated as the code's data.
2. *The "capsule" model:* Each packet is treated as a complete program to be executed at each node of the network being traversed by the packet.
3. *The "option"[16] model:* Some standard services or modules are residing in the network nodes. These are selected and invoked through *options* carried in the user's packet. The rest of the user's packet will be treated as data to be processed by the invoked routine.

[14] Of course, hardware re-configuration and programming is possible in terminal devices and (to some extent) at the FPGA-level of some specialized cards in present day switches and routers. However, there is still no commercial product or research prototype that allows the runtime exchange of switching circuitry (plug-and-play modules) synchronized by driver updates in the operation system.

[15] According to a recent DARPA/ITO report [Tenn99], initial research began in the early 90ies.

All these models belong to the 1st Generation Wandering Networks. Figure 6 illustrates the individual streams and topics of research in active and programmable networks following a personal view on a classification provided in [Camp99a]. It is evident that the classical AN approaches still dominate the field. Some of them will be discussed in more detail in chapter 3.

2.1.2 BASIC DEFINITIONS

What makes a net *active* ? Perhaps the most distinct feature of Active Networks (ANs) is that they do not clearly distinguish between network, transport, and application layers. An active network represents a *programmable* environment with accessible storage for running distributed programs and designing customizable network services to ensure application-specific processing.

For the reason of understanding, we provide three basic definitions, which are going to be used in the sequel of this work. These definitions explain the differences between passive and active networks, and the related field of mobile agents.

> **Basic Definition 1:** A *passive* or traditional (packet switched) network is composed of: a) *"smart/intelligent"* hosts allocated at the edges of the network which perform computations on data up to the application layer, and b) *"simple*/*stupid"* routers inside the network interconnecting the hosts and capable to perform computations only up to the network layer.
>
> **Basic Definition 2:** An *active* (packet switched) network allows intermediate routers to perform computations in the application layer. Besides, users can configure and program the network by injecting pieces of executable code into the network elements ("active nodes", both hosts and routers), and thus by modifying their state and behavior. The executable code is embedded within the transmitted packets ("active packets") along with the data exchanged between nodes of the network.
>
> **Basic Definition 3:** A *mobile agent* is an autonomous program or processor unit that acts on behalf of a user or another program or processor, and is capable of moving within the network under its own control. The mobile agent chooses why, when, and to where it will migrate upon evaluation of the executing environment it is currently operating in, as well as how to interrupt its own execution and continue it elsewhere on the network. The agent can communicate and return results and messages in synchronous and asynchronous fashion.

In the extreme case, there may be no difference between internal *active nodes* and the end user ones, since both will be capable to perform the same computations on data. In addition, *active packets* can be ultimately complete programs with the data they require.

---

[16] Initial research in AN was primarily focused on the first two network models. Yet, researchers are arguing that it is unlikely that a significant number of users will wish to perform network programming .

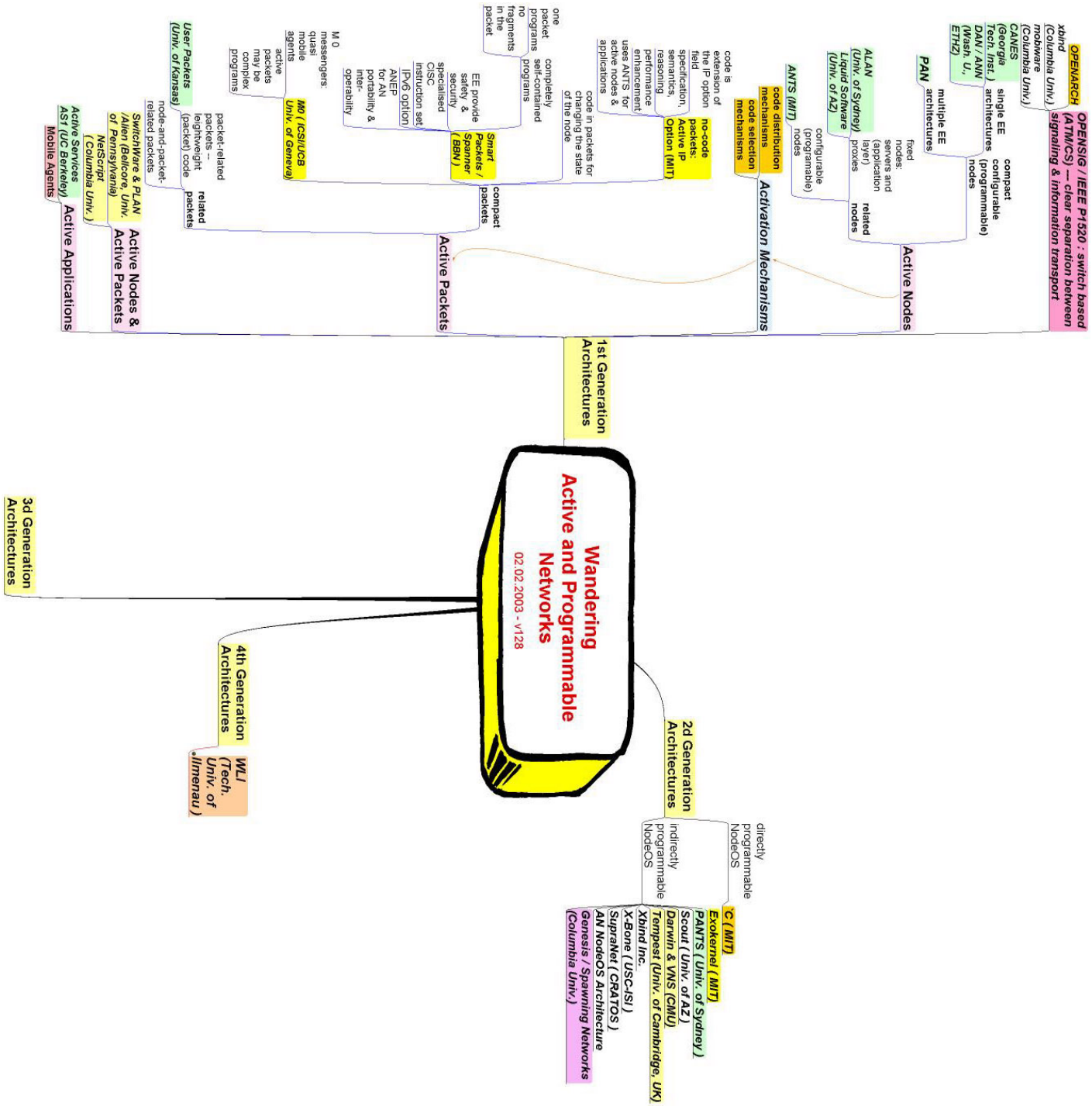


Figure 6: The AN origin of the Wandering Network

The concept of a packet as a wandering program inevitably leads to the notion of *intelligent agents* and, in particular, of *mobile software agents, which can be,* regarded a specific class of the former. The similarity between the two approaches, active packets and mobile software agents, is obvious. Indeed, many of the active network architectures use mobile code techniques that are very close to mobile software agent technology.

However, the idea of active networks is much more general. Active networks visualize the network as a collection of active nodes that can perform any computations, and a collection of active packets that carry code and are indeed programs. Under that viewpoint, a mobile agent may be regarded as a specific type of an active packet, and a mobile-agent-compatible node of traditional networks (which is required to execute the mobile agents) could be regarded as a specific type of an active node, since the latter is safe and secure.

The fundamental difference between the two ideas is that active networks use the concept of *network layer processing* whereas mobile agent systems run as *application programs*. Therefore, an active network, which is programmable by definition, offers the deployment of mobile software agent technology as a kind of “primitive” option.

The notion of *processor* should be regarded at the broad sense, e.g. from a VLSI device, through a plug-and-play terminal component, to a *mobile* base station controller or an ad-hoc networking vehicle and a space shuttle. The mobile agent paradigm proposes to treat the network as multiple agent-friendly environments and the agents as program entities or plug components that move from location to location-performing tasks for users.

2.1.3 ACTIVE VS. PASSIVE NETWORKS

Passive Networks (PN) such as the present day Internet are transporting user data between end terminals where intermediate nodes of the network are simply forwarding these data to other nodes based on look-up-tables. Active Networks (AN), in turn, represent a *quantum leap* in the network evolution. By providing a minimal set of programmable interfaces in network nodes, they open the resources, mechanisms and policies underlying their enhanced core functionality, and provide mechanisms for constructing or refining an infinite spectrum of new services from those elements. Thus, routers can support applications in performing *computations* on selected user data (Figure 7), while packets can carry programs for execution on routers and possibly change their state, configuration or execution environment. In other words, active networks support dynamic modification of the network behavior as seen by the user, which can be developed to a *network-on-demand* paradigm.

The active network differs from traditional architectures primarily in what it does *not* specify. Instead of defining how the nodes work together to provide the network service (for instance, through best effort datagram delivery), the active network describes functional slots that must be instantiated to provide a particular network service. These slots create a new degree of freedom, a sort of *slackware,* i.e. idle functionality that can be invoked on demand in the network architectures, which in turn opens up the opportunity to speed-up network evolution and thus accommodate new network types, algorithms and applications.

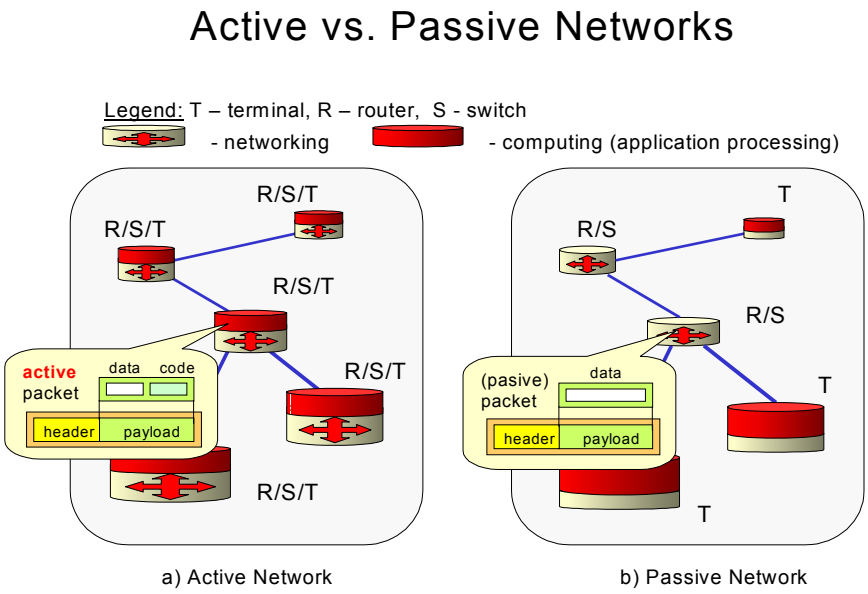


Figure 7: Active networks allow the asymmetric and asynchronous allocation of some application- and user-specific parts of the service inside the network

It is to be noted that programmability in network nodes is not a novelty. In fact, telephony networks including switches and their intelligent service control points, as well as packet routers are nothing else, but high-performance programmable machines, programmed to perform a specific task, i.e. call-switching, service control or packet-forwarding. What is new about the Active Networks is that they expose this programmable interface to the user.

The frame presented in this work addresses a generation of active networks, where some converging technologies are going to be utilized to transform networks in a qualitatively new kind of *expanding soft networking*.

2.1.4 Major Challenges for Implementing an Active Network

In order to be successful, any implementation of a programmable active network should satisfy the following requirements:

- **The network services should be usable**: any active programming model will impose new and unfamiliar programming models and constraints on the user. For ease of use, it is important that these constraints should be as limited as possible.
- **The network should be highly flexible**: Flexibility is the primary reason behind the Active Networks research. The network subsystem should be adaptable to a wide variety of tasks and applications.
- **The implementation should be secure**: Security is expected to be the major obstacle to large-scale deployment of active networks in the future.

. The network should guarantee high performance. Performance is usually the price that we have to pay for flexibility. Therefore, the deployment of Active Networks should not create new bottlenecks in the network infrastructure. In particular, the implementation should provide for fast and transparent path processing of non-active packets.

2.2 WHAT IS RECONFIGURABLE COMPUTING ?

Reconfigurable computing is a relatively new, but important field of research in computer architectures, which began in the late 1980's. It is an attempt to bridge the traditional gap between hardware and software within the computing field. By placing the computationally intense portions of an application onto the *reconfigurable hardware*, the overall application can be greatly accelerated. This is because reconfigurable computing combines the benefits of both software and application specific integrated circuit (ASIC) implementations. This extra hardware is called the reconfigurable device, also known as *Reconfigurable Computing Machine (RCM)*. The reconfigurable device allows designers to build part, or all of their design in hardware rather than software. Like software, the mapped circuit is flexible, and can be changed over the lifetime of the system or even during the execution time of an application.

Reconfigurable hardware systems come in many forms, from a configurable functional unit integrated directly into a CPU, to a reconfigurable co-processor coupled with a host microprocessor, to a multi-FPGA (flexible programmable gate array) stand-alone unit. The level of coupling, granularity of computation structures and resources are key points in the design of reconfigurable systems. In addition, *run-time reconfiguration* provides a method to accelerate a greater portion of a given application by allowing the configuration of the hardware to change over time. Apart from the benefits of added capacity using virtual hardware, run-time reconfiguration also allows circuits to be optimized based on run-time conditions. In this way, the performance of a reconfigurable system can approach or even surpass that of an ASIC.

Reconfigurable computing machines have shown the ability to accelerate greatly program execution, providing a high-performance alternative to software-only implementations for computation intensive applications such as DSP transcoding. Since Active Networks allocate more computation inside the network, RCM technology is of special interest for building programmable network-wide hardware overlays for multimedia and security based applications, which integrate (mobile) components from multiple network elements.

2.3 WHAT ARE ADAPTIVE SYSTEMS ?

The adaptive systems research in computer science involves the application level “adaptability” (e.g., multimedia presentation, user level security and communication), the processor and memory level (course grain architectures: cache hierarchies, multiprocessing), all the way down to the gate and devices level (e.g., programmable logic devices.). In this way, *reconfigurable computing architectures represent adaptive systems if they are organized as reactive systems*, i.e. if their programmable state is changed in response of external actions.

2.4 RESEARCH TOPICS IN WIRELESS MOBILE COMMUNICATIONS

Kleinrock's breakthrough paper on Nomadic Computing in 1995, [Klein95], defined the margins of a new discipline that combines the characteristics of computing, communications and mobility. He was the one who predicted a paradigm shift in the way telecommunications will evolve in future.

The characteristics of nomadicity include *independence* of location, motion and communication platform while assuming the presence of access to remote data, systems and services. The notion of independence here does not refer to the quality of service one obtains, but rather to the *user's perception of a computing environment that automatically adjusts to the processing, communications and access available at a given moment*. The ability to adjust all aspects of the user's interaction (computing, communication and storage) in a transparent and integrated fashion is the essence of a nomadic environment. The key systems parameters include bandwidth, latency, reliability, delay, storage, processing power, interference, error rate, interoperability, user interface, etc. The values of all these parameters may change dramatically in a nomadic environment.

According to Kleinrock, one of the key problems in this new world of mobile communications is to develop a *complete system architecture and set of protocols* for nomadicity.

The nomadic protocols are required to meet the following basic requirements:

- interoperation among different infrastructures
- integrated, ad-hoc access to services
- provision of graceful degradation of network services
- scalability of address space, QoS, bandwidth, number of users, etc.
- cooperation among system elements such as sensors, actuators, devices, network, operating system, middleware, services, applications, etc.

In addition, the new platform has to satisfy the following criteria of wireless communications:

- ability to deal with unpredictability of user behavior and network platform capability
- ability to match the nature of the transmitted content to the bandwidth availability (compression, approximation, partial information, etc.)
- maximum independence between the network and the applications from both, the user's viewpoint and the development viewpoint

The research fields, which are expected to help in providing the above requirements, are:

- Integrated software framework representing a common virtual network layer
- Location services (for people and devices)
- Resource discovery
- File discovery and predictive caching
- Adaptive data base management and synchronization
- Appropriate replication of services at various levels

Another research opportunity in developing the future "nomadic" networks paradigm is the provision of a reference model of nomadicity, which will allow a discussion to its attributes, features and structure in a straightforward fashion.
The model should be able to explain:

- System state consistency up to the level of files, database, applications, etc.
- Functionality (bandwidth, infrastructure, nature and QoS of communication)
- Locality, or awareness, i.e. how aware is the user of the local environment and the environment of the of users and their profiles

The development of mathematical models of the nomadic environment should allow the study the performance of the network under various workloads and system configurations.

A serious constraint in wireless communications, which creates a whole range of research problems, is the limited frequency spectrum set up by the laws of Maxwell and Shannon. Therefore, another major research theme in wireless networking is the profound exploration of all available degrees of freedom in utilizing the available resources within a given range. The spectral limit of GSM will be probably reached quite soon. Therefore, the driving force behind the 3^d Generation wireless systems were the ITU submissions IMT-2000 and the perceived needs for high-speed data transmission up to three mbps compared to 9.6 kbps in GSM. Among the leading UMTS/IMT-2000 proposals, CDMA/TDMA and W-CDMAOne are the ones providing a graceful upgrade from GSM. With optimized packet switching, data throughput is considerably increased (more than doubled) compared to data over voice circuits. However, the available bandwidth is going to oscillate during a transmission session because of the characteristics of the wireless link such as packet losses and delay jitter resulting from field interferences, handover and disturbing Internet traffic. These effects inevitably lead to unacceptable Quality of Service degradations of multimedia applications.

Furthermore, although the bandwidth offered by wireless communication media varies over an broad range, the nature of the error rate, fading behavior, interference level, mobility issues, etc. for wireless are considerably different so that the algorithms and protocols require some new and different forms from the ones in wireline network [Katz95].

The main reason for QoS degradations is that the existing Internet traffic control mechanisms such as TCP and UDP were primarily developed for the use in wired networks. Therefore, the wireless part of the packet network has to be treated separately[17]. The primary characteristics of wireless mobile networks are relatively lower bandwidth, intermittent connectivity, and higher error rates. Wired networks offer high bandwidth, steady connectivity, and very low error rates. Bandwidth-intensive or near real-time applications require special handling to surmount the limitations of wireless mobile networks. The main reasons for this dichotomy are:

- Deficient resources,
- Limited bandwidths in wireless links,

[17] For instance, the Wireless Access Protocol (WAP) is such an approach. However, it is completely orthogonal to the straightforward end-to-end IP QoS concept. WAP breaks the IP net into two parts and requires a complete exchange of the Internet protocol suite and applications.

- Failing transfer channels,
- Limited energy supply.

Therefore, bandwidth and resources (whatever they are) have to be shared or at least managed in an intelligent way. Some of the main research topics in this area are outlined as follows:

- Supporting the development of new multimedia applications is of crucial importance for the market success of mobile wireless multimedia technologies. New applications have to be *capable to serve in an adequate*[18]*, reasonable and reliable fashion*.
- Multimedia applications (in particular novel applications) for wireless communication require a *higher performance, adaptability* and *flexibility* of the networks in order to able to effectively use the limited RF spectrum and resources of the network.
- Traditional protocol structures (IP) are based on the layer concept. There is no support for guaranteeing QoS between the layers. Because of the permanent fluctuations in the wireless channels, the QoS concept should be able catch with the prevailing dynamic changes and demands of the communication environment.
- The new protocol versions should be developed with respect to the characteristics of the wireless networks[19]. In particular, the limited performance of the current mobile terminal equipment has to be taken into consideration.
- Because of the spontaneous, dynamic nature of wireless mobile communications, multimedia routing should be increasingly considered in an application, QoS, user behaviour and infrastructure aware fashion. Ad-hoc, multi-hop approaches have to be systematically investigated to derive patterns and policies for adaptive system state changes towards an autonomous, self-organizing communication infrastructure.
- Mobile wireless multimedia requires a flexible network topology on demand.

Among these research issues, the evolution and the expansion of the mobile network as a whole, has not been investigated sufficiently until now. This subject defines the focus of the Wandering Network approach, which we advocate in this thesis.

* * *

[18] "All Bits are Not Created Equal" ([Negr97]). All networks also. Therefore, wireless communications should be designed with the nature of the bits in mind.

[19] In most cases, new and custom protocols (such as WAP) are developed and deployed in the wireless network to meet user needs. Unfortunately, the current network infrastructure is rigid, and the protocol stack in the network is usually fixed. To avoid the time-consuming standardization process, network protocols must possess the following properties to enable users to build their own modular, extensible and verifiable protocol frameworks [KuMi00]:

- Modularity — decomposing complex protocols into smaller components, each implementing a piece of communication functionality;
- Introspection — being able to access an existing protocol component and work with abstractions to it;
- Intercession — being able to modify behavior of an existing protocol component.

CHAPTER 3: ACTIVE NETWORKS

"Iron rusts from disuse, stagnant water loses its purity,
and in cold weather becomes frozen, even so
does *inaction* sap the vigors of the mind."

LEONARDO DA VINCI (1452 - 1519)

3.1 INTRODUCTION

In the 80[ies] Zander and Forchheimer worked on an experimental system (`Softnet`) in which packets of FORTH code were interpreted by network elements [ZaFo83]. Their objective was to improve the performance of processing software in a communicating environment. Later, in the mid 90[ies], the concept of *active networking* (AN) emerged in the DARPA research community while discussing the most familiar network problems:

- the poor performance due to redundant operations at several protocol layers,
- the difficulty of integrating new technologies and standards into the shared network infrastructure, and
- the difficulty to accommodate new network services in the existing architectural model.

Within traditional packet networks, such as the Internet, computation is extremely limited. Although routers may modify a packet's header, they pass the user data without examination or modification. Furthermore, the header computation and the associated router actions are specified *independently* of the user process or application that generates the packet. For this reason, there is no way to influence the behavior of the network through the application itself.

Active networking ([TeWe96], [Pso99]) represents a new paradigm in communications in which *the nodes of the network are user-controllable* by providing a programmable meta-level interface to the network (i.e., they offer an open execution platform on which user code can be executed). These networks are active in the sense that nodes (network elements) can perform computations on, and modify the packet contents. We can identify three typical application scenarios for active networking: firewalls, proxies and mobile terminals. The advantages of the new approach are summarized as follows:

- Firewalls act as filters on packets. They implement application and user specific functions. Updating such functions or adding new features to them is complex. Active Networks allow the injection of software updates to firewalls by approved security vendors.
- Web proxies are used for caching web pages. Today they are usually part of the end user organizations. Active networks allow caching at other specific points in the network (e.g. to support ad hoc multicast streams).
- Mobile terminals in nomadic computing are characterized by different bandwidth requirements on the access network. Active networks offer the opportunity for effective service management on a per-user and per-connection basis.

The idea of *messages carrying procedures* and data (e.g. the message passing interface, MPI, [MPI]) is a natural step beyond traditional circuit and packet switching, which allows the rapid adaptation of the network to changing requirements of the execution environment.

This program-based approach provides a foundation for expressing networking systems as the composition of many smaller processing elements with specific properties where services can be distributed and configured to meet the needs of particular applications. In addition, the overall network behavior can be observed and modified in terms of the properties of individual components. As mentioned in Chapter 1, we regard programming as extended to the hardware.

This chapter is organized as follows. We continue with the basics of active networking in section 3.2, and the underlying conceptual paradigms, followed by the Reference Model[20] active networks and a summary of the mainstream directions and the current implementation framework. Section 3.3 is dedicated to the active network architectures with the goal to provide a classification based on the results of the surveys given in [TeWe96], [YeSi96], [Tenn97], [Calv98] and [Smith98]. The expose includes a short review of the enabling technologies for active networking highlighting their key advantages. Section 3.4 reviews the domains of AN research with a focus on a variety of applications. A special attention is dedicated to the application of active networks in mobile communications. Finally, section 3.5 provides an overall analysis and discussion of the active network approach. An outlook for further research and a summary with conclusions are given in sections 3.6, 3.7 and 3.8 respectively.

3.2 FUNDAMENTALS

Active networks apply active technologies, that are used today mainly at the network terminals and in the end-to-end network layer, to every network layer (Figure 8). They combine the properties of mobile agents, which carry mobile code from clients to servers with the properties provided by Web applets that transport mobile code from servers to clients. In short, active networks allow applications to dispatch computation to where it is required [TeWe96]. Thus, a single internal network node can be configured in several different ways to perform some dedicated tasks with the transferred data in the network.

The idea behind active networking is to enable nodes to performing computation on packets in order to make the network *adaptable* to changing requirements of the environment. They are expected to facilitate a) the creation and introduction of new network services, and b) the deployment of new protocols for emerging applications.

There are two basic approaches in active networking, *discrete* and *integrated*, depending on whether programs and data are carried discretely (i.e. within separate messages) or in an integrated manner. There are also two basic architectures in active networking according to the entities enabled with the property of “being active”, Figure 10: *Active Packets* (AP) and *Active (Network) Nodes* (ANN).

[20] which is referred in this work’s genealogy as the *First Generation* (1994-1999).

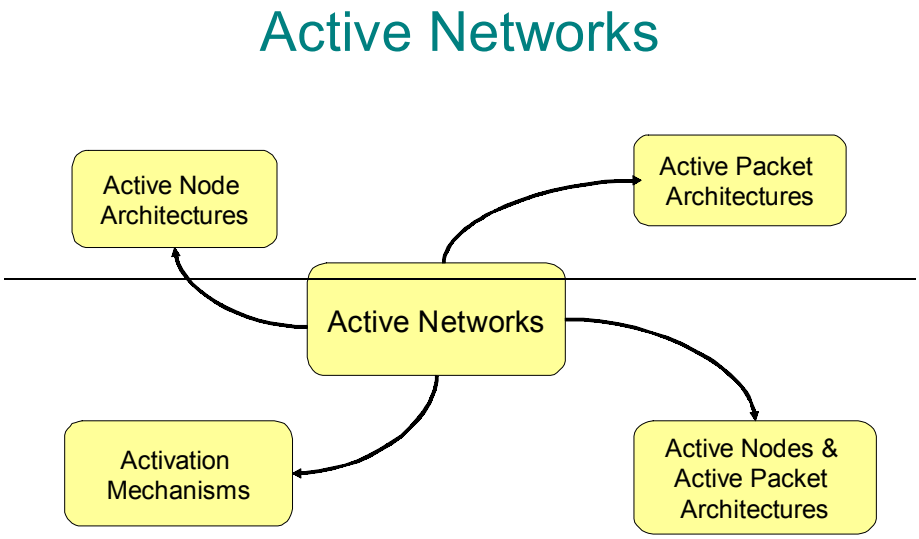


Figure 8: Active network design approaches

In addition, there is also a hybrid approach to active networking, which is based on both active packets and active nodes. We adopt the nomenclature used in [Tenn97] and [Camp99a] with the consensus to apply the term “active node” also to programmable (configurable) switches.

3.2.1 The Reference Model

In the following, we review the *Wandering Network Reference Model* which emerged out of the overall AN model developed at the DARPA active networks program, [DoDAN], and the generalized architecture of a programmable network defined by Campbell et al. [Camp99a] along with our own enhancements related to the network hardware components. This further generalization of the concept of network programmability in this work was necessary in order to distinguish between the basic AN attributes related to run-time extensibility and programmability of active applications and executing environments, and the enhanced ones emerging in some recent developments and addressing the lower layers of the AN architecture.

Despite adopting some of the terminology proposed in [Calv98], [Camp99a], it should be noted that there is still no consensus on the provided definitions. As already mentioned in Chapter 2, the terms “active” and “programmable” will be used interchangeably.

An active network (Figure 7) is a kind of store-and-forward network. It consists of a set of (active) *nodes* interconnected by transmission *links*. The purpose of the network is to share the transmission facilities.

The basic unit for multiplexing the transmission facilities is the (active) *packet*. The (end) *user*[21] is either the originator or the recipient of the packets carried by the network. Nodes receive packets from users and other nodes, perform computations based on their internal state and the control information (header) carried in the packet, and as a result of that computation may *change its internal (soft) state* and forward one or more packets towards other nodes or to users.

The nature of the *network service* is defined by the behavior of the particular nodes in the network, and how users can control that behavior through coded information embedded in their packets. In present (passive) IP networks, where routers examine the destination address field of the IP header with internal routing tables to determine to which neighbor they should forward the IP packet, the role of user control over the network's behavior is fairly limited to the range of values that can be placed in that field (and a few others) of the IP header. Active networks can offer new network services capable to evaluate the information contained in *all* of the packet headers.

The *network application-programming interface (network API)* comprises those aspects of the network behavior that are visible to the end user. Open signaling networks [OPENS] have a clearly defined set of network APIs [P1520]. In active networks, the network API is a kind of *virtual machine (VM)* capable of interpreting different protocol languages representing the different views of the network. As the network evolves, the network API develops as well.

The general approach to network programming [Camp99a] specifies an environment, which offers a *set of network programming interfaces and tools for designing network architectures* from building blocks in a similar way as constructing applications by using software development tools. These interfaces specify how programmable architectures are constructed in terms of:

- *network service*s, which the network architecture realizes as a set of distributed network algorithms and offers to the end systems;
- *network algorithm*s, which includes transport, signalling/control and management mechanisms;
- *multiple time scale*s, which impact and influence the design of the network algorithms; and finally
- *network state managemen*t, which includes the state that the network algorithms operate on (e.g., switching, routing, QoS, etc.) to support consistent services.

Related to the network node level, such architecture environments address *node kernels*[22] and define the common base functionality[23] for: a) *how* data is processed; b) *what* resources are available at the node; and c) *how* they are accessed.

[21] Users are, in general, different from *node/network administrations*, which control the configuration and interconnection of network nodes.

[22] This is used to address the node operating system by both the Active Network and the OPENSIG communities.

[23] In the *WLI* network model presented in Chapter 5 we introduce two other definitions: a) *how* resources are configured, and b) *how* they are enhanced and exchanged.

Programmable network architectures may range from simple best-effort forwarding architectures to complex mobile agent protocols that respond dynamically to changes in wireless QoS and connectivity. Given this diversity it is necessary that both[24] network programming environments and node kernels are extensible and programmable to support the large variety of programmable network architectures.

According to Campbell *et al.* [Camp99a], the node kernel[25] represents the lowest level of programmability of the network. Yet, we argue that programmability does not stop at the operating system level. The network hardware[26] can be also (re-)configured and programmed down to the gate level if required (Figure 4, the Hyperactive Network). This last level of programmability is reviewed in Chapter 4 (Reconfigurable Computing).

The following section discusses in some detail the main concepts of active networking.

3.2.2 CONCEPTUAL PARADIGMS

3.2.2.1 Active Nodes

Active nodes are programmable elements in an active network that enable the *deployment of custom services and protocols*. Initially, they emerged as programmable switches, [Tenn97], maintaining the existing packet/cell format, and providing a discrete mechanism that supports the downloading of programs. Later on, the term active node was introduced to include additional functionality which was able to change the data format (e.g. for the purpose of transcoding in video transmission).

An active node plays a double role in the network. On the one hand, it has to be transparent for the present day IP traffic. It operates as a router. On the other hand, it has to perform a specific task on selected packets in accordance to its active function.

The reference model of an active node is illustrated in Figure 9. It consists of a passive and an active part separated by a packet filter. The passive part contains the IP routing stack as we know it from present day Internet. The active part has been historically associated to local[27] protocol enhancements in the layers above the network known as the *protocol booster* approach, [Marc98]. However, recent research has shown that it can be also related to layers below the network (e.g. DLL, [Sim00], [SRBW01], Figure 26) to improve performance and error-resilience, e.g. in video transmission.

---

[24] These two levels of network programming were progressively associated with the *First* and the *Second Generation Wandering Networks,* respectively (Chapter 1).

[25] Henceforth, the terms “node kernel” and “NodeOS” will be used interchangeably.

[26] This level of network programmability was associated with the *Third Generation Wandering Networks* (Chapter 1). A detailed discussion of Configurable Computing follows in Chapter 4.

[27] for the particular active node only; this is an adaptation technique for dynamic protocol customisation to heterogeneous environments on an as-needed basis proposed by Bellcore and the University of Pennsylvania. The boosting mechanism is under control of a policy determining when augmentation of the protocol stack is required.

Active Node Architecture

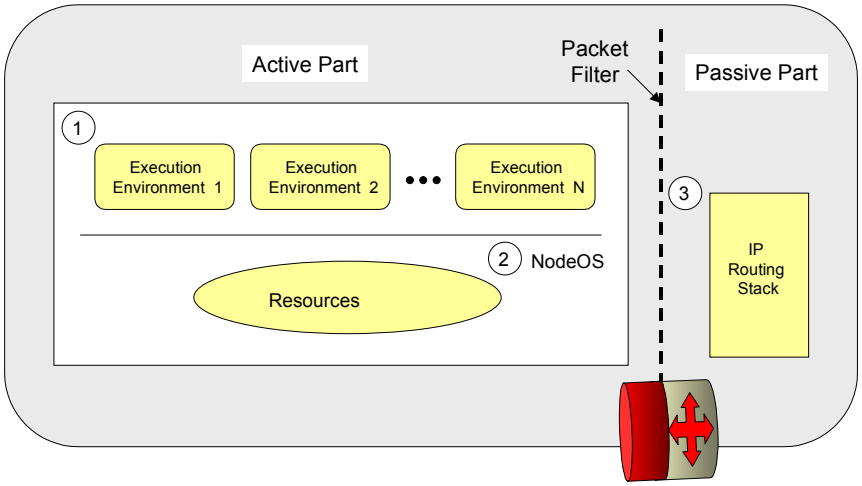


Figure 9: The Active Node reference model

The basic Active Node components include:

1. one or more execution environments (EEs), virtual machines (VMs) where active applications can be executed[28];
2. a node operating system, NodeOS, which manages the node's resources and the communication inside the node on behalf[29] of the EE, and
3. the IP routing stack of the passive part.

There are three layers of code running on each active node: active applications[30] (AA), execution environment and NodeOS. The EEs role is to offer AAs a sufficiently high-level programming environment, whereas the NodeOS is responsible for multiplexing the node's resources among the various packet flows. The AN reference model, i.e. the First Generation (1G) Wandering Networks, allows network programmability at the AA[31] and EE layers. It defines a NodeOS interface to the EEs [NOSIS].

---

[28] The Execution Environment corresponds to a Unix Shell for processing active packets.

[29] The NodeOS provides a programmable interface for the EE to the node's resources.

[30] Not shown on Figure 9. For an illustration, please refer to Figure 12. An Active Application is a program loaded from active packets that implements a particular application (e.g. an application-level routing protocol in an overlay active ad-hoc network as shown in [Gold01]).

[31] Application Level Active Networking (ALAN)

The Second Generation (2G) WN considers also[32] programmability at the NodeOS layer, whereas the Third One (3G) expands programmability to the resources layer. It requires a second NodeOS interface to the hardware components.

3.2.2.2 Active Packets

Active packets[33] (or capsules) are executable programs (possibly containing data) which are the means of communication in an active network. They are characterized by a unique type and, optionally, a destination address, data, and program code in the form of methods that can be executed locally at any node in the active network. With the capsule approach, the passive packets of legacy architectures are replaced by active miniature programs that are encapsulated in transmission frames and executed at each node along their path. By injecting active (smart) packets into the active nodes to modify their behavior, applications are enabled to customize network resources for dynamic adaptation. Active (or smart) packets can also carry code in the form of application-specific protocol frameworks composed from custom protocols.

Active (or smart) packets differ from the "passive" or (traditional) ones by the fact that besides structured data they also carry executable[34] code. Active nodes, in turn, are executing on the arriving packets. They differ from the passive ones by the property that besides the simple processing at the transport layer (switching, routing), code can be executed at the higher layers up to the application. In this way, some tasks that have been initially allocated at the *smart* edges of the packet network can be performed now inside the network for some reason[35].

An active packet consists of two parts: the *header* and the *payload* (Figure 10). The header of the packet has a specific format and contains information about the protocol being used, about the *target handlers*, the entities that would process the packet on the intermediate nodes, the sender, the receiver and the device drivers. Generally, a target handler can be an execution environment, a classic IP router stack or a dynamic handler, e.g. a flow. The NodeOS uses the Active Networks Encapsulation Protocol (ANEP, [Alex97]) in case that the target handler is an execution environment. It is also in charge of directing incoming packets to the corresponding target handler by analyzing the packet header.

Currently, each implemented execution environment such as ANTS [WGT98], CANES [CANES] or PLAN [Hicks98] has a specific ANEP identifier assigned to it. The payload can contain data or pieces of code or both. Its length can vary from zero to the maximum length that will keep the whole packet within the SDU limits allowed by the underlying transport system.

---

[32] 2G active networks can be programmed at both EE and NodeOS layers where the NodeOS can be addressed in a twofold manner: a) indirectly, via the EE, and b): a) directly through a special interface.

[33] **Note:** Sometimes, active packets are refered to as *mobile agents*. Generally, this is not true, since mobile agents are application-layer software entities, which can be deployed in traditional networks without affecting the network architecture or functionality. In this work, we introduce the term *pro-active agents* when we want to address the specific network adaptation role of active packets or their aggregations as application layer entities.

[34] An active packet is self-contained and usually embedded within a single IP packet along with the data it is defined for. An applet is a complete executable program (mobile code) exchanged between networked computers.

[35] For instance, error detection and correction should not necessarily occur at the destination or source side respectively, if these procedures can be performed more effectively at some active hop (proxy) inside the network.

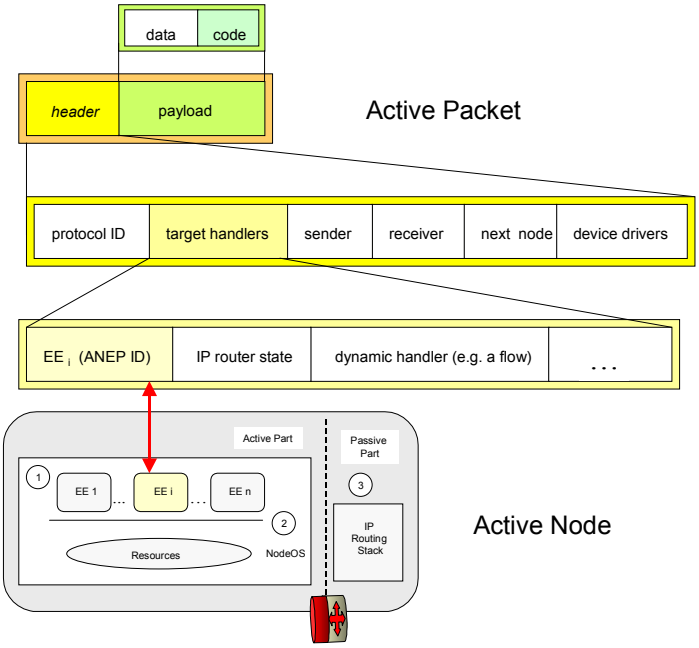


Figure 10: The structure of an active packet and its mapping into an EE

In some architectures, the executable code refers to the data of the same packet that carries the code. We call them *compact*[36]. Other architectures let packets transfer some identifiers or pointers to indicate which code should be executed on behalf of them. We call them *related*, because the executable code can be allocated:

a) in some of the network nodes: *node-related* architectures;
b) in some of the following packets of the same stream (or even a different stream[37]): *packet-related* architectures; or
c) in both some active node(s) and some active packet(s), i.e. *node-and-packet-related* architectures.

In addition, the active network defines mechanisms for *code distribution* and downloading since not all nodes need to have all the code that they can execute at a time. Finally, some architectures provide the user with some mechanisms to choose between *lightweight* code that is carried by active packets and *heavyweight* code that resides in the active nodes.

[36] Traditional nodes can be also regarded as compact architectures from the viewpoint of code allocation since they execute code, which is local to them. Also, compact architectures can be configured to perform different tasks on packets. Chapter 7 provides a generalized definition of the active network architecture.

[37] consider, e.g., an audio / video synchronization.

The following two sections discuss the major characteristics of the two basic schools in network programming: Active Networks and Open Signaling.

3.2.3 THE ACTIVE NODE APPROACH

The software architecture of an active network node (ANN) consists of two parts (Figure 11), *Execution Environments (EEs)* and a *Node Operating System (NodeOS).* The EE is responsible for the implementation of the network API, while the NodeOS manages access to local[38] resources by EEs. The architecture supports multiple network APIs simultaneously.

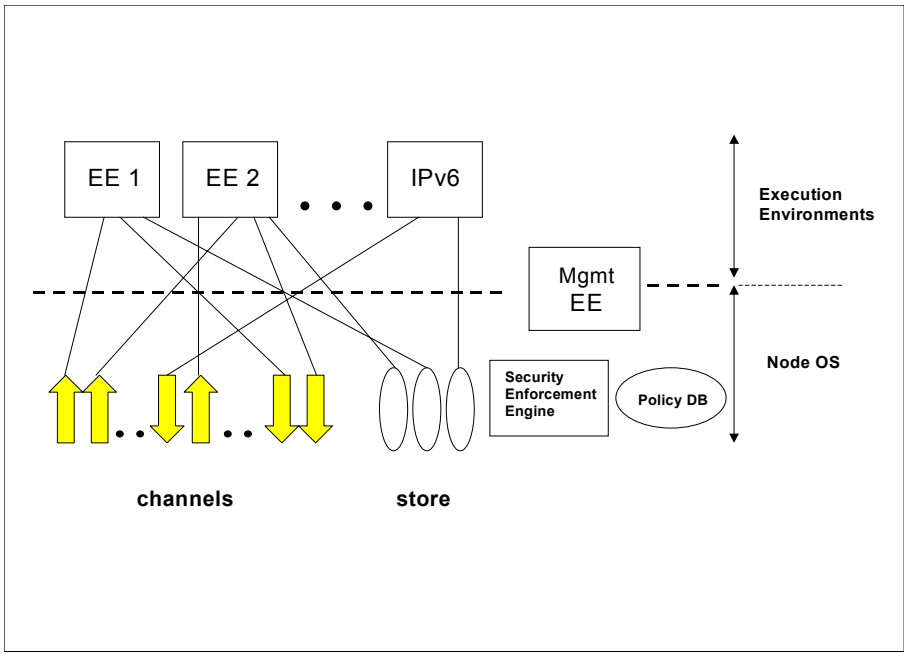


Figure 11: The software architecture of an active node

Each execution environment represents a virtual machine defining a programming interface through which end-to-end network services are provided to users. The NodeOS hides the details of the internal AN architecture (resource management, configuration, etc.) to the EE and the user. The EEs, in turn, abstract most of the details of the external network environment (e.g. interaction with the user and other nodes and entities comprising the so-called *principal*) from the NodeOS. In addition, the NodeOS is responsible for the enforcement of security policies defined in terms of principals.

[38] The WLI approach requires that the NodeOSs manage the configuration, the enhancement and the exchange of resources between the nodes through the corresponding EEs by *building virtual resource sets (VRS)*. To create new behaviors of the node, we allow the injection and creation of *own EEs* on user request. Yet, deploying such "customer" VMs requires special resource management procedures which are not discussed further in this work.

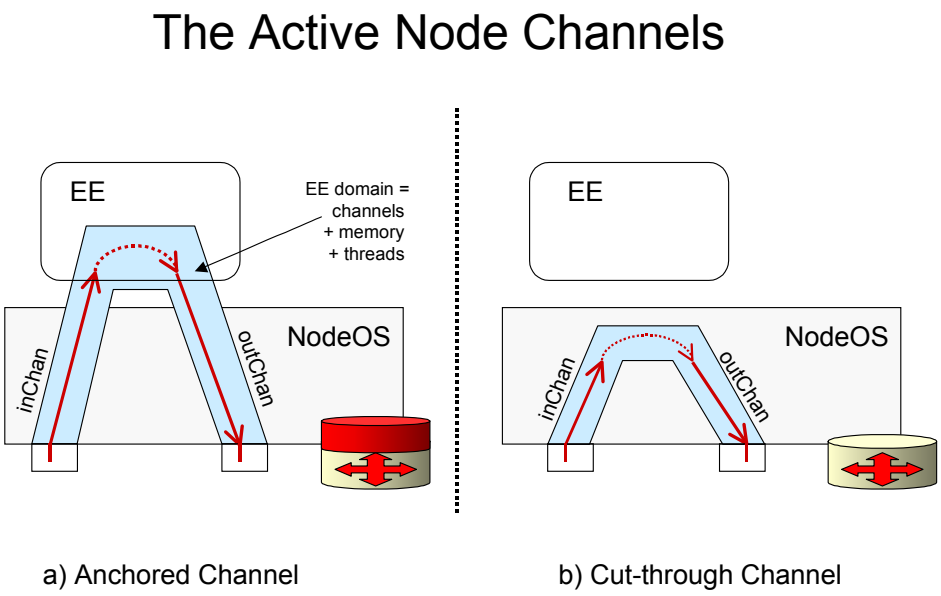
3.2.3.1 The Node Operating System

Many network vendors integrate an operating system, a *node kernel*, within their switches and routers to configure and maintain the communication system functions of the network nodes. These system functions include signalling, control and management processes, as well as forwarding, inter-process communication and download of new boot images. Because of their proprietary nature, such operating systems offer limited support to evolving network programming environments and are practically closed to end users and third party providers.

The Node Operating System (NodeOS) of an active node provides the basic functions from which EEs build the components of the network APIs. It manages the resources of the active node and mediates the demand for them basically by multiplexing the node's communication, memory and computational resources among the various packet flows traversing the node.

The NodeOS hides the details of resource management from the EEs, as well as execution environments from each other. The resources of the node consist of:

a) *threads*, computational resources such as CPU usage and data storage;
b) *memory* in terms of memory pools for packet buffers and holding the EE-specific state;
c) *files* for persistent storage; and
d) *channels* directing packets to the target handler (e.g., an EE, Figure 12).



a) Anchored Channel b) Cut-through Channel

Figure 12: An Active Node configured in a) active mode and in b) passive mode

The general computation model for programmable networks described in [Camp99a] (Figure 3, Figure 4) enables the programmability of the communication model. For instance, MIT's `Exokernel` [Eng95] system is a flexible NodeOS which securely multiplexes machine resources and allows a great degree of application-specific customization of OS abstractions.

The NodeOS uses *domains* to aggregate control and scheduling of the node's resources through the NodeOS interface. A domain is the primary abstraction for addressing admission control, scheduling and accounting for packets. The NodeOS communicates with the EEs through an interface which is based for evolutionary reasons on established interfaces such as POSIX. Its primary role is to support classified packet forwarding[39] at very high speeds. The secondary role of the NodeOS interface is to allow an extension or configuration of the underlying NodeOS itself.

For reasons of simplicity and efficiency the NodeOS interface specification [NOSIS] does not provide the means for an EE to extend the NodeOS directly. In their generalized model for programmable networks, Campbell et al. [Camp99a] propose a low-level programming environment which runs on switches and routers as a node kernel and provides a small set of node interfaces to support the manipulation of the node state (e.g. for access and control of the node's resources) and the invocation of communication services.

3.2.3.2 The Execution Environment

The Network Programming Environment (NPE) illustrated on Figure 4 represents the "middleware glue" between executing network architectures and the node kernels themselves. It is composed of a set of execution environments spread out throughout the nodes of the network to support the dynamic deployment of network services and protocols.

The NPE operates over a set of well-defined NodeOS interfaces and offers distributed toolkits for the creation of programmable network architectures while supporting different levels of programmability, programming methodologies, networking technologies and application domains. The services offered by an network programming environment range from simple Remote Procedure Calls (RPC) between distributed network objects to sophisticated dynamic downloading of mobile code and fast compilation into an intermediate machine-independent representation.

An execution environment defines the particular programming model for writing active applications. Each execution environment has an input channel for storing incoming packets to be processed and an output channel for placing outgoing packets, Figure 12. The execution environment provides a high-level interface for active applications, which are EE-specific. When a packet reaches an active node, depending on the target handler, it is either directed into an incoming channel of an EE, or into a cut-through channel. Usually, there is only one cut-through channel per node. This channel is used when the packet does not need to be processed which is the case e.g. with IP packets.

[39] as opposed to running arbitrary computations in common, terminal operating systems.

The packet in a cut-through channel bypasses all execution environments and is sent directly to the routing stack to be forwarded to the next node. Any packet in a channel anchored to an EE is processed by the EE. The results of processing a packet vary depending on the content of the packet and also on the execution environment. Usually a packet (the same one or a new one) is put in the outgoing channel of the EE. This packet is then going through the routing process and sent to the next node.

3.2.3.3 The Node Operation

An active node is operating as follows. When an EE requests a service from the NodeOS, the request is accompanied by an identifier (and possibly by a credential) for the principal (also, the EE itself) in whose behalf the request is made. The NodeOS presents this information to the *Security Enforcement Engine*, which verifies its authenticity and checks that the node's security *Policy Database* authorizes the principal to obtain the requested service or perform the requested operation.

EEs are allowed to implement their own policies to enhance these of the node, but they may not override the NodeOS own policies. For exchanging packets between the EEs, the NodeOS implements communication *channels* consisting of physical transmission links (e.g. ATM, Ethernet, etc.), and the protocol stack processing associated with higher-level protocols (IP, TCP, UDP).

When an active node receives a packet over a physical link, it classifies the packet based on the packet's contents (i.e. headers); each packet is either assigned to an existing channel or discarded. The mapping of incoming packets to channels is controlled by a pattern of headers (e.g. an Ethernet type or a combination of TCP port and IP protocol numbers) specified by the EE when it creates the channel. It is the responsibility of the security engine to ensure that a given principal is allowed to create a channel with a particular pattern. Input channels are scheduled for processing, whereas output channels are scheduled for both processing and transmission.

3.2.2.4 The Node State

The active node state is always given by the active[40] components present on the node (incl. the Active Applications, AAs), Figure 13. The notion of a *next*[41] active state depends on the action that is taking place. When a packet arrives at a node, the next state of the node is one where the packet is buffered in some channel, in addition to whatever was present in the previous state. If a packet is being executed, the next state is rather one where the packet has been removed from some channel, with the rest of the node not modified.

[40] Here we distinguish between *active* and *non-active* components on a node w. r. t. the concept of a virtual active node (VAN), a special feature of the WLI approach discussed later in chapter 7.

[41] The *Next* state construct is an essential part of the TLA formalism [Lamp94]. Therefore, we assume that TLA is ideally suited to model and verify active networks with real-time constraints.

Active states (memory) provide a mechanism for active packets and active applications to temporarily store information. The use of (active) states for describing the configuration of an active node makes the task of formally specifying the node easier. Since the NodeOS provides an underlying interface to each EE, it can be assumed as in [KAD00], that from the EE's perspective the network consists only of other execution environments of the same type. This filtered view is not only a good abstraction for formalizing the node model, but also the base for the WLI's *layered "shadow* network" (multiple overlaying) principle described later in chapter 6.

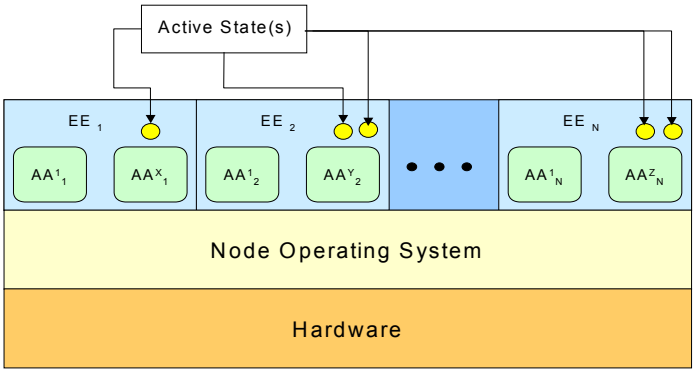


Figure 13: An Active Node application layer architecture

For instance, a 1G EE-centric node specification may look as follows in the PVS (Program Verification System) formalism [PVS] (and C, for comparison only):

```
ActiveNode: TYPE = [#              |     typedef struct ActiveNodeSpec {
   address:   Address,             |         address    Address;
   sendQ:     Queue[Pckt],         |         sendQ      Queue(Pckt);
   services:  set[SrvcID],         |         services   set(SrvcID);
   resources: set[RsrcID],         |         resources  set(RsrcID);
   policies:  set[PlcID],          |         policies   set(PlcID);
   data:      Resources            |         data       Resources;
#]                                 |     } *ActiveNodeSpec
```

The address field represents the address of the node on which the particular EE is running. Each execution environment has a send queue, `sendQ`, used for buffering the outgoing packets. The EE mechanisms comprise services, resources and security policies. Services represent the programmable functionality of an active node which controls the node resources on behalf of the NodeOS. They can be resident, installed on or removed from the EE depending on the AA executed. The service identifier, `SrvcID`, is used to identify a service that is available to programs running in the execution environment.

The resources of an EE are the node resources allocated to that EE by the NodeOS. The resource and policy identifiers, `RsrcID` and `PlcID`, denote the specific resource and security policy respectively. The construct `Resources` defines all types of resources that an EE can have such as memory and routing tables. The node resources cannot be accessed directly by packets and must be exploited through a special set of services which are guided by security policies. Each resource must have a unique resource identifier and data type. The exact set of resources that are available to an EE is given by the set of resource IDs.

```
SecureResource: TYPE

RsrcID: TYPE ={          |    Resources: TYPE = [#
      secureResource,    |       secureResource:  SecureResource,
      registeredHosts,   |       registeredHosts: setoff[Adress],
      routeTable,        |       routeTable:      [Address -> Address],
      registered         |       registered:      bool
}                        |    #]
```

Two types of policy IDs are of particular interest, `routePolicy` and `securePolicy`, granting access to the routing table and the secure resources correspondingly:

```
PlcID: TYPE = {          |        securePolicy: TYPE = [#
      routePolicy,       |           safetyPolicy:   SafetyPolicy,
      securePolicy,      |           livenessPolicy: LivenessPolicy,
      relaxedPolicy      |        #]
}
```

There are two groups of security policies. The first group is used to restrict the access to services and guarantee *safety* of the nodes, while the second one monitors the use of security policies by the user to avoid a user's policy monopolization of the node's services, and hence – the node's resources, and ensure *liveness* of the network. (The requirements for *safety* and *liveness* are mandatory constructs in temporal logic, [MaPn92]; they will be discussed in more detail in chapter 6.) A user's safety policy has to be designed in such a way, that it should not restrict the use of a resource to that user only. On the other hand, the liveness policies should control the interaction of users' policies in such a way that the running network is deadlock-free. These properties of the system can be easily modeled and verified by using a formal technique such as TLA [Lamp94].

An active packet can be described in PVS as follows, Figure 10:

```
ActivePacket: TYPE = [#
      ptype :          PacketType,          % protocol ID
      thandler:        TargetHandler        % ANEP ID, flow ID, etc.
      src :            Address,             % sender
      dest :           Address,             % receiver
      nhop :           Address,             % next node
      driver:          DeviceDriver,        % device driver ID
      payload :        Payload              % data and/or code
#]
```

The first six fields describe the header of the packet. The resolution of the exact payload content for a specific packet is obtained from the `ptype` variable in the header:

```
Payload: TYPE = [#
      getSecureResourcePayload: GetSecureResourcePayload
#]

GetSecureResourcePayload: TYPE = [#
      value : SecureResource,
      outgoing: bool
#]
```

3.2.4 THE OPEN SIGNALING APPROACH

Although active networking targets to reduce the amount of standardization in service provisioning, there is still some effort required to integrate active solutions within the available infrastructure. The primary interfaces in a First Generation AN architecture are the user-EE interface (network API) and the EE-NodeOS interface. The EE-NodeOS interface may vary from node to node; all that is required is to provide a standard set of basic services to EEs. Beyond this interface, there are only a few features that require standardisation, mainly encodings that must be understood by both the end user and the NodeOS such as ANEP, the syntax and semantics of principal identifiers and security credentials, and the measurement units for resource allocation.

The Open Signaling community (OPENSIG) represents the telecommunications industry approach to making the network programmable. It deals with methods for improving the network signaling system while providing "open"[42] access to switches and routers using a set of programmable network interfaces to the different layers of the network elements. Emphasis is given on service creation and QoS. There is also a clear distinction between the transport, control and management planes. Physical network devices are regarded as distributed computing objects (e.g. virtual switches, *switchlets* [Merw97b], and virtual base stations) with well-defined open programmable interfaces. These open interfaces allow service providers to manipulate the states of the network using middleware toolkits such as TINA and CORBA in order to construct and manage new network services. The OPENSIG's domain of interest begins with the Plain Old Telephone Service (POTS) and ends (currently) with real-time video services, Figure 14.

Supporting this space is a challenge to the network signalling system. This challenge means different things for different networks:

- For ATM networks, it means how to enrich its signalling capability to cope with multi-point and multi-media.
- For telephony networks, it means how to enrich transport to include say data and video.
- For IP networks, it means how to ensure the packet forwarding network supports QoS for voice and video.

[42] This term is quoted because OPENSIG distinguishes in fact between different classes of network users.

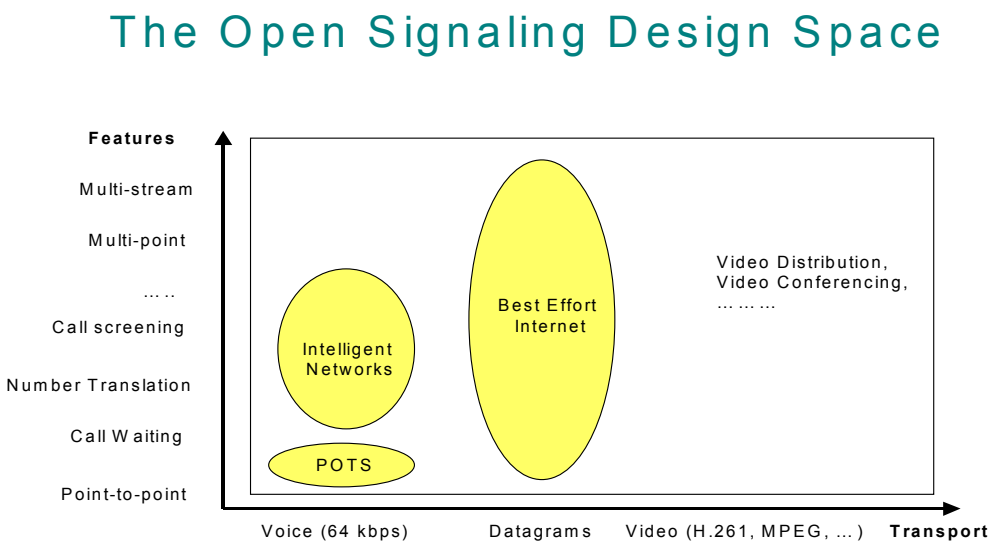


Figure 14: The OPENSIG domains of interest

The AN approach goes in fact beyond the OPPENSIG approach, especially when one considers the dispatch, execution and forwarding of packets based on the notion of "active packets". However, both communities share the common goal to go beyond existing approaches and technologies for construction, deployment and management of new services in telecommunication networks. Both trends include a broad spectrum of projects with diverse architectural approaches[43]. However, the OPENSIG approach clearly separates network control from information transport[44] and is primarily focused on programmable switches that provide some level of QOS support. In contrast, projects under the AN umbrella have historically been focused on IP networks, where the control and data paths are combined.

Recently, the IEEE standard project P1520 ([Bis98], [P1520]) on Programmable Interfaces for Networks is pursuing the OPENSIG approach in an attempt to standardize programming interfaces for network control and signaling on ATM switches, IP routers and mobile wireless networks. It proposes software application programming interfaces (APIs) for (active) networks based on IDL[45], in particular - for service and signaling control, in a much more integrated and elaborated fashion than SS7[46] in circuit-switched intelligent networks and ATM.

[43] For example, few AN projects consider every packet to be an active capsule and similarly few OPENSIG projects consider programmable network interfaces to be static.

[44] similar to the functional split between the SS7 net for signalling and the switching voice net in intelligent networks

[45] Interface Description Language

[46] Signalling System 7, [Fayn97].

P1520 is developing a reference model separating end-user application semantics, value-added services, network-generic services, virtual network devices, and hardware and/or low-level software support. The project aims to establish programming interfaces between the levels of the reference model which are closely related to the two levels of abstraction in an active node, e.g. as described in the PLAN [Hicks98] and CANE [CANES] approaches.

Figure 15 illustrates the different levels of the P1520 Reference Model for APIs for networks. In this model, there are levels, entities at each level, and interfaces between levels.

At the value-added services level (V interface), the entities are end to end algorithms that add value to services provided by the third and lower levels by means of user-oriented features and capabilities, such as real time stream management, synchronization of multimedia streams and other capabilities beneficial to value-added service providers and end users. At the network-generic services level (U interface) the entities are algorithms that deal primarily with the functioning of the network.

At the virtual network device level (L interface) the entities are logical representations (objects) of certain state variables of these entities in the first level. All three levels comprise the software interfaces that are abstractions of the physical resources in the Physical Element (PE) level. Finally, the entities at the PE level are physical elements of the network, such as ATM and IP switches, and local exchanges in narrowband circuit-switched phone networks.

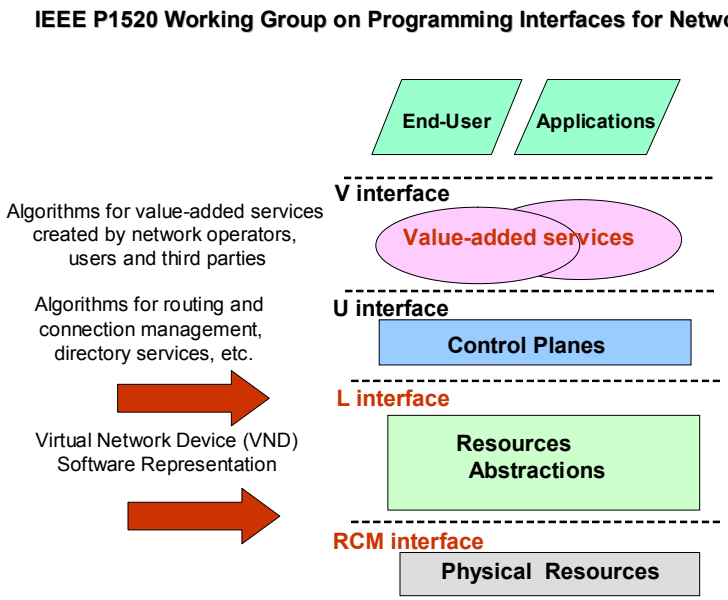


Figure 15: The IEEE P1520 network API layered architecture

The interfaces between these layers provide the APIs that users use to build their programs. The uppermost V interface provides a rich set of APIs for writing highly personalized end-user software, often in the form of value-added services. These services generally do not deal directly with the communications process per se, but provide convenient features such as the Service Independent Building Blocks (SIBBs) in Intelligent Networks that enhance the value and experience of using the basic communication service.

The second U interface deals with generic network services and allows users to make requests for connections. These connections may take the form of simple point to point connections, point-to-multipoint trees, or any general graph (as in the case of a VPN). The power of this interface comes from its generality, which in essence allows parameterisation of connection set-up requests independent of the algorithm used in the connection set-up procedure. This separation between interface and implementation in principle would allow multiple connection management schemes to coexist in a single network.

The third L interface defines the API to directly access and manipulate local network resource states. These could include VC/VP lookup tables in ATM or routing tables controlling IP forwarding. Finally, the bottom-most RCM (Reconfigurable Computing Machine) interface is not a programming interface but a collection of protocols that enable the exchange of state and control information at a very low level between a switch and an external agent.

The reference model described above provides a high-level framework for positioning programming interfaces for networks. It is necessary to map this high-level model into existing networking technologies to be able to recognize the point at which “useful” programming interfaces may be obtained.

The advantages of opening up application programming interfaces (APIs) for networks are many. First, there are the benefits associated with leveraging distributed object-oriented technology and modeling. These include benefits of object-oriented software engineering such as modularity, reusability, scalability, and reliability, which go a long way in reducing the service deployment cycle. Second, there are benefits of distributed computing, such as location-transparent remote access and dynamic binding, which make it possible for third-party service developers to write applications that perform third-party call set-up and management. Third, incorporating programming interfaces in networks for developing control and management applications allows unprecedented separation of software and hardware.

This, in turn, ensures that the end user gets the full benefit of competition in the marketplace. Fourth, there is now a separation of the signaling business from the transport business, permitting rich and flexible ways of dividing and segmenting the market. Finally, through gateways one can ensure that legacy interoperability will always be kept in the forefront.

3.3 BASIC ARCHITECTURES

The diverse architectures of the First Generation Wandering Networks can be distinguished by the applied networking technology, [Camp99a], i.e. according to *how activity is achieved at the network APIs* [OPENS] or within the EE and the active applications [DoDAN]. In the following sections, we review the most representative architectures in some detail.

3.3.1 ACTIVE PACKETS

Most of the early active networks architectures follow the active packets approach, which is fundamentally characterized by the fact that the code is carried by the packets. The nodes are also active because they allow computations up to the application layer to take place. However, *no active code* resides in them. Therefore, the reason for calling these technologies "active packets" (AP) technologies is that active code is carried by the packets either to be executed on the data of the same packet that carries the code, or to be executed in order to change the state or the behavior of the node.

The key question in all AP approaches is how to provide a rich and flexible programming environment without overloading the computing power of the managed node and without making the environment so complex that it cannot be secured easily. Examples of such architectures are the Active IP Option proposed at MIT [WeTe96], the Smart Packets project proposed at BBN Technologies [SmartP99], and the M0 [M0] architecture proposed at the University of California at Berkeley and at the University of Zurich.

The ANTS architecture is the prototype system for most active network approaches, and consequently of the representative AP technology. In the following section, we review the ANTS components in some detail.

3.3.1.1 On Capsules, Options and Protocols

The behavioral pattern applied for packet processing and forwarding within a network is called *protocol*. The ANTS architecture is based on the protocol as a single unit for protection and customization of network processing; *protocols are used by applications to configure the entire network*. The different behavioral patterns are associated with different protocols. Each protocol is composed of a set (a sequence or flow) of related capsules associated with the corresponding behavior. This relation is called *type* of the protocol/capsule, Figure 16.

A *capsule* (also called *active packet*), Figure 17, is the generalized replacement for (an IP) packet. ANTS considers its function as inclusion of a *reference* to a forwarding routine to be used to process the capsule at each active node. Of course, this function can be also extended, e.g. to carry short programs in a special language to be interpreted and executed at each node as is the case with PLAN [PLANet].

The capsule contains an identifier for the corresponding protocol based on a fingerprint of the protocol code. This identifier is used for de-multiplexing to a forwarding routine in the same sense as the Ethernet type and the IP version.

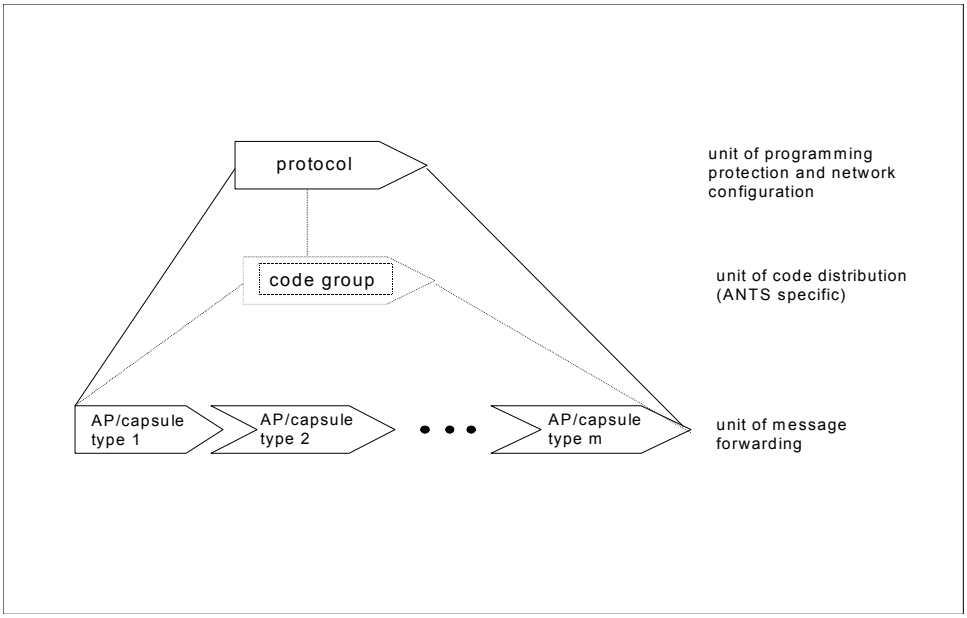


Figure 16: The capsule format

Deriving a capsule ID from the protocol code has the benefit that protocols and capsules can be quickly allocated in a decentralized manner. Thus, active nodes can easily verify for themselves (without any external trusted parties) whether the obtained code belongs to a capsule/protocol, it pretends to correspond.

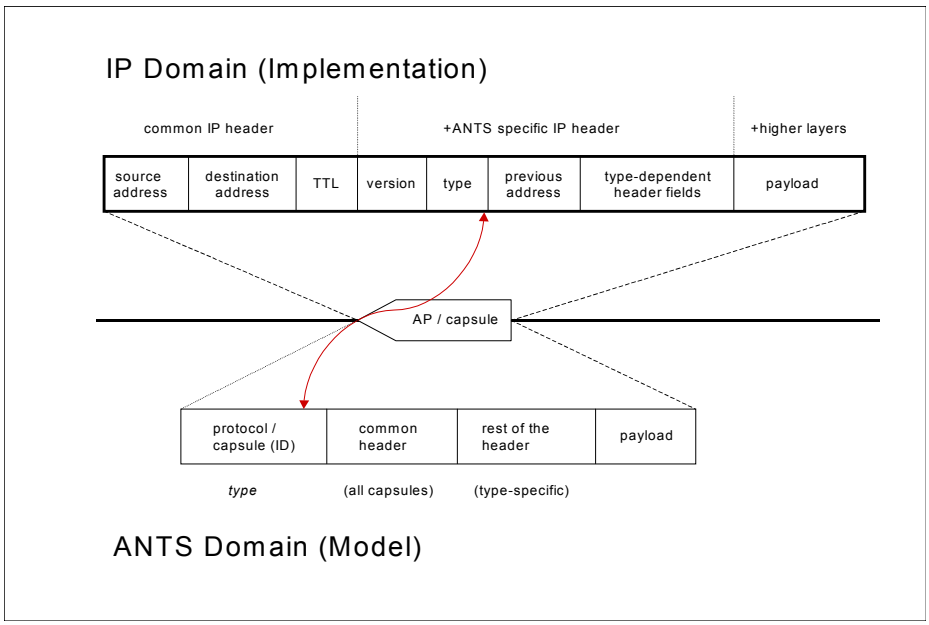


Figure 17: The concept of capsule and its IP implementation

3.3.1.2 Active IP Option

The ACTIVE IP Option ([WeTe96], Figure 18) was used by the ANTS project to implement the capsule concept in a standard IP packet (IPv4/IPv6). It describes an *extension to the IP options mechanism* that supports the embedding of program fragments in datagrams and the evaluation of these fragments as they traverse the network. Present day passive packets are replaced by active *capsules*, miniature programs, which are executed as they travel. Capsules can invoke predefined primitives that interact with the local node environment, and leave information behind in a node that they have visited. Subsequent capsules can carry code that depends on this information.

Active options can perform routing, copying, and merging functions. The processing environment allows ambient network conditions to be examined, the current datagram to be dispatched, and additional datagrams to be constructed and sent. The state of the node may also be modified.

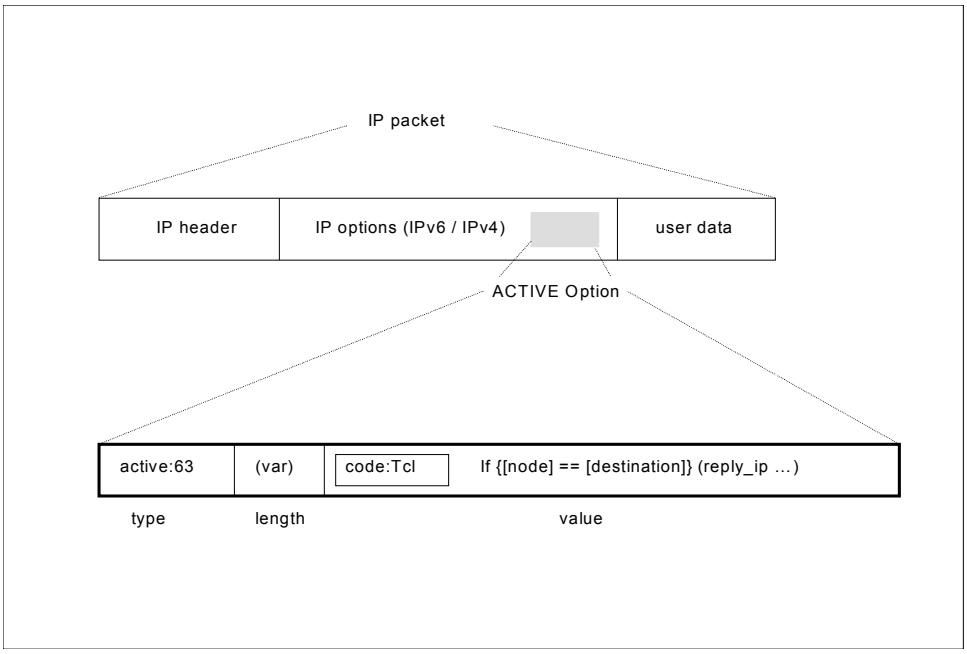


Figure 18: The format of the ACTIVE IP Option field

The capsule approach is an “in-band” approach in the sense that capsules carry the code along with the data on which it operates. Two options are defined. The first is used to carry program fragments, which may be encoded in a variety of languages. The second is used to query an active router for the languages it supports. Backward compatibility is automatically achieved because Internet hosts silently ignore options they do not recognize. Since the scheme is based in an extension of the IP Options mechanism, the capabilities of the technology are limited (for example, arbitrary protocols cannot be deployed). The language used in the first implementation of the architecture is TCL. The processing is done by a stripped-down TCL interpreter resulting in a restricted environment conceptually similar to that of Safe-TCL. This is the only means by which security and safety issues are addressed.

3.3.1.3 Smart Packets

The *Smart Packets* project at BBN and the University of Kansas aims the application of AN technology to network management and monitoring [SmartP99]. The project implements two important design principles. The first one is that *programs must be completely self-contained* to avoid the need for persistent states in a router. This means that *programs have to* fit entirely into one packet, so they cannot be more than 1 KB long, - and the packet should not be fragmented. The second principle is that *the operating environment must provide safety and security,* since packets containing executable code are potential hazards. In addition, a special cross-network protocol, called Active Network Encapsulation Protocol (ANEP) ensures the portability and interoperability among different active networks projects within the DARPA active networks program [Alex97].

Smart packets represent elements of in-band and out-band mobile code based on Java classes. They propagate state information of the nodes in the form of serialized objects and carry identifiers for authentication purposes. The mobile code is bound to and delivered with an IP data packet.

Active nodes offer a set of resource abstractions and primitives, which can be accessed by the smart packets:

- *resource controllers*, which provide interfaces to node resources;
- *node managers*, which impose static limits on resource usage; and
- *state managers*, which control the amount of information that smart packets may leave behind at an active node.

The active node supports a feedback-scheduling algorithm to allow partitioning of CPU cycles among competing tasks and a credit-based flow-control mechanism to regulate bandwidth usage. Each smart packet is allocated a single thread of CPU and some amount of node resources. The active code's lifetime at the destination node is only the time to execute it completely. The packets are not allowed to leave any state in the intermediate nodes.

Active nodes also include router managers that support both default routing schemes and alternative routing methods carried by smart packets. Before processing, Figure 19, packets are authenticated by taking a cryptographic hash of the non-mutable fields of the packet and comparing it to a certificate attached to the packet. Then, network management and monitoring programs generate smart packets, which are encapsulated within ANEP packets. The latter are encapsulated within an IP packet. Then smart packets are sent either to an end host or to each router in a hop-by-hop manner along the path to an end host. In the first case, the content of the smart packet is executed in the end host[47] and the results are returned back to the originating application. In the second case, the content is executed in all intermediate nodes. An ANEP demon located in each node is responsible for both injecting and receiving smart packets and for offering a secure environment, called virtual machine, for executing the programs.

[47] sometimes even without packet transmission: compare the straight line in Figure 19 denoting some signaling between the packet originator and the intermediate node which results in data processing at the originator side.

The smart packets' code can be written in either Sprocket, a high-level language much like C, or Spanner, an assembly language. Sprocket programs are compiled into Spanner code, which is assembled into a compact machine-independent binary encoding, that is placed into program packets. According to the authors, the reason why two new programming languages are used is that the already existing languages require space < 1 Kbyte and that none of them had compact, platform-independent encodings. The runtime system for Spanner includes support for accessing network management variables (MIBs) efficiently, but there is no access to system calls or memory outside the current packet or its declared dynamic variables.

As far as security is concerned, smart packets achieve the correct operation of a router and its configuration by evaluating programs conservatively (i.e., if a virtual machine does not know how to handle a situation it quits execution and sends an error packet back to the source of the program), by checking whether a program comes from an authorized user, by checking the data integrity of a program in each node, and by placing limits on the execution of programs, such as offering a resource-limited environment.

Applying the SmartPacket approach to network management has the following advantages:

- The returned information can be controlled and managed according to the needs.
- The management rules can be shifted from the management centres to the programs.
- The monitoring and control loop is shortened.

Smart packets capabilities are indirectly limited by two reasons: first, the programs must be at most 1 KB long; second, the functionality provided by the project is limited and tailored to network management applications. The positive part is that the performance of the technology should be good comparing to other active packets approaches. However, we are not aware of any results proving this yet.

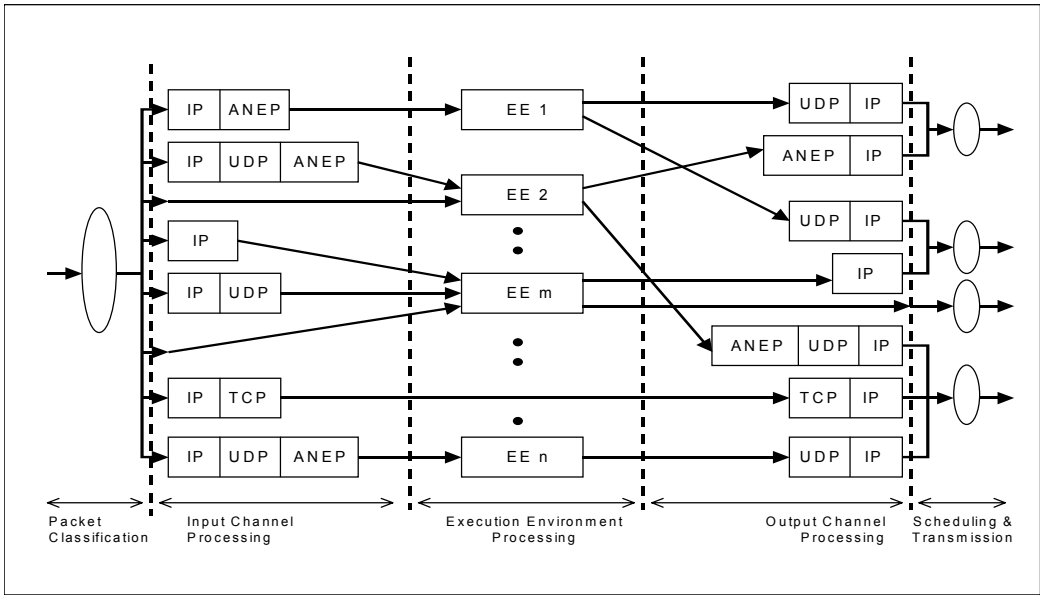


Figure 19: The logical flow of packets through an active node

The smart packets approach has been used to program enhanced HTTP and SMTP services that show some performance benefits over conventional HTTP and SMTP by reducing excessive ACK/NACK responses in the protocols. A beacon routing scheme supports the use of multiple routing algorithms within a common physical IP network based on smart packets.

3.3.1.4 The M0 Architecture

The messenger in the M0 system [M0] is similar to the capsule or the smart packet. Messengers are programs exchanged between M0 nodes. There are four elements inside the M0 node: concurrent messenger threads, a shared memory area, a simple synchronization mechanism, and channels toward neighboring nodes.

An independent and anonymous thread of control executes each messenger. These threads have their own private memory space; they are fully protected from each other. Messenger threads can deposit arbitrary data structures under self-chosen names so that other threads can access them. Thread queues are a way to serialize the execution of threads in order to avoid race conditions.

Channels enable messenger threads to send new messenger packets to neighboring nodes. The current M0 implementation maps messenger transmission to UDP, Ethernet, or serial-line communications.

Messenger code is written in the M0 language. M0 is a high-level language that inherits from PostScript the main concepts of operand, dictionary, and execution stack, as well as the main data manipulation and flow control operators. The M0 interpreter is written in C. M0 has no explicit code caching or code loading functionality. The code is shipped with every messenger.

This works quite well for small protocols where the code is only a few bytes long. For large code sizes, messengers implement their own caching method by storing the code in the shared memory area of a node under a chosen name. This option allows the deployment of *any protocol*, no matter how complex it is. Therefore, the M0 architecture appears to be more powerful than the previous two architectures.

Each M0 node manages its own resources independently of other nodes. All resources have price tags, which depend on the node's actual load for a given resource and on the demand from the running threads. Messenger threads are charged for their activities. When they run out of money, they are silently removed form the system. On arrival, each messenger thread obtains an account with some start money. The amount is sufficient to do some exploration inside the node and eventually send out another messenger. There is no authentication between M0 nodes, nor has a messenger any identity attached to it that would allow authentication. Safety-related questions on resource consumption have to be handled by controlling the flow of money. Access control for system resources is controlled by agreements between messengers and the system. M0 provides some basic cryptographic operators that can be invoked by a messenger.

3.3.2 ACTIVE NODES

In the active nodes approach, the packets do not carry the actual code, but instead carry some identifiers or references to predefined functions that reside in the active nodes. The packets are active in the sense that they decide which functions are going to be executed on their data, and they provide the parameters for these functions. However, the actual (heavyweight) code resides in the active nodes; the packets do not carry it.

This is why these technologies are called “active nodes” technologies. The motivation for such architecture is that the active packets approach suffers from performance-related problems because both safety and security requirements are huge, or capability related problems because the only way to minimize the security and safety issues is by restricting the programs that are carried in packets (e.g., Smart Packets or PLAN/SNAP packets). Examples of “active nodes” architectures are CANES proposed at Georgia Institute of Technology, the ANN/DAN architecture proposed at Washington University and at ETH Zurich, and the ANTS architecture proposed at MIT.

3.3.2.1 Composite Active Network Elements

The CANEs project at Georgia Tech and the University of Kentucky [CANES] aims to define and apply *service composition rules* as a general model for network programmability. A composition method is used to construct composite network services from components. It is specified as a programming language with enhanced language capabilities that operates on components to construct programmable network services. Attributes of a good composition method include high performance, scalability, security and ease of management.

Features of well-structured composition methods combine:

- control on the sequence in which components are executed;
- control on shared data among components;
- invocation methods, which are defined as events that cause a composite to be executed;
- division of functionality among multiple components, which may either reside at an active node or be carried by packets.

The CANEs approach aims to define a composition method that optimises all of the above points. The CANEs definition of service composition encompasses the OPENSIG approach to network programmability indicating how different approaches to programmable networking complement each other by addressing the same goal from different perspectives.

LIANE is a composition method proposed within the CANEs project and incorporates all of the aforementioned features. The key idea of LIANE is that services are composed from basic underlying programs that contain processing slots. Users insert programs for customisation in these slots. LIANE attempts to construct dynamic, trustworthy services from unreliable base services. It is not tied to a particular language, although its prototype implementation is in C++, but relies on a reduced programming model that gives a pre-decided amount of flexibility to the dynamic environment. The advantage is that the required security analysis can be limited. Some efforts are under way to apply LIANE to classes of multicast service and to provide verifiable safety assertions about the behaviour of composed services.

In the CANEs architecture, users control the invocation of predefined network-based functions through control information in packet headers [H1, H2]. Users can select from an available set of functions to be computed on their data and can supply parameters as input to those computations. The available functions are chosen and implemented by the network service provider, and support-specific services. Thus, users are able to influence the computation of a selected function but cannot define arbitrary functions to be computed.

This approach has some benefits with respect to incremental deployment, security, and efficiency. However, it seems to be slightly restrictive because only the functions that have been pre-loaded can be called upon. Each of the functions that a node supports has a unique identifier. Each packet has a set of headers, which specify the identifier of one or more functions to be applied to the packet and parameters to be supplied to those functions. When the packet is processed, the function identified by each header is applied, resulting in updating of the node's state and possibly modification of the rest of the packet. Thus, for each function *f*, and each parameter value *p*, there is a particular subset of the node's generic state information that is relevant to *f* and parameter *p*. Functions cannot modify or use parts of the node state that are not relevant.

The strength of active networking can be realized by incremental addition of user controllable functions. Each function is precisely defined to support a specific service. The introduction of a new active networking function involves specification of its identifier, of the parameters associated with it, and of its semantics. Once a function is specified, each provider or vendor is free to implement it in a manner consistent to the specifications. The CANEs definition of service composition comprises the OPENSIG approach to network programmability.

This approach has backward compatibility in that not all users have to be aware of the active functionality in the network, and not all nodes have to support the same functions. The scheme may have low flexibility and restricted capabilities but it achieves high performance because security can be easily addressed.

3.3.2.2 Distributed Code Caching

The packets in Distributed Code Caching for Active Networks (DAN) architecture [DAN] contain a finite sequence of function identifiers, and parameters for the functions. The functions are daisy-chained in the sense that one function calls the next according to the order of the identifiers in the packet. Depending on the type of node that the packet is processed upon and the packet's content, only a subset of the functions may be called. Thus, the packet may be interpreted as a sequence of function identifiers $fi_2 ... fi_N$,as shown in Figure 20, with a distinct set of parameters $P_1 ... P_N$. The first function is not indicated by any identifier but is derived from the context in which the packet processing starts (e.g., a packet received by an Ethernet card results in the calling of Ethernet input function).

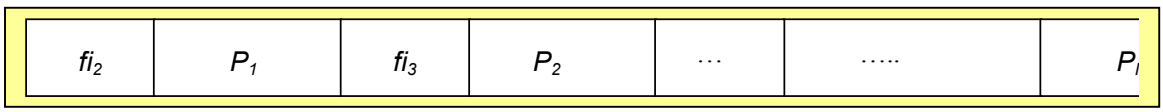


Figure 20: The format of a datagram

If the node is unable to locate a function, it temporarily suspends the processing of the packet and calls a “code server” for the implementation of the function. The code server is a well-known node in the network, which provides a library of functions for different types of operating systems from various developers. Once the module is downloaded, it is permanently stored locally on the node in order to prevent more downloads of the same module. Code servers are put in a hierarchy for the best possible distribution of active modules.

The option of downloading modules differentiates this technology from the previous one. DAN is more flexible because new functionality can be deployed and then just added to the code servers. If a node needs a new module it can easily download it. In the previous technology, the network manager should add to each node all the functions that they may need.

The active modules provided by the code servers are programmed in a high-level language such as C and compiled into object code. Once the node loads the functions, they are in no way different from the ones compiled into the network at run time. Thus, all functions run at high speed and the performance is good. However, downloading a function on demand causes some delay that reduces the overall performance. A possible solution is to download the modules before they are needed.

Security concerns are addressed by using well-known code servers, which authenticate themselves and give the node the possibility to check the module’s sources, and by providing digitally signed modules from well-known developers only.

The security problem is thus reduced to the installation of a *rule*, which enables the node to choose the right code server, and a database of public keys to check the developer’s signature. In addition, even if the module sources and the modules are authenticated, network administrators may restrict the set of developers from which they accept modules.

3.3.2.3 Active Node Transfer System

The Active Node Transfer System (ANTS) is an active network toolkit developed at MIT [WGT98], where arbitrary new protocols are automatically deployed at both intermediate nodes and end systems by using mobile code techniques. The ANTS approach is focused on standardizing the particular communication model, rather than individual communication protocols such as IP, UDP, etc. The major design goal is to build a system that allows the rapid transfer and deployment of protocol code across the network. The network is viewed as a distributed programming system. The model is designed to support many protocols simultaneously, in such a way that only the parties that use a protocol (i.e. not the entire networking community) have to agree on how it is built and used. ANTS has been slightly modified at MIT to set up the Practical Active Network (PAN) test-bed. ANTS uses Java as programming language, and the Java Virtual Machine as runtime environment. These features make ANTS suitable for a variety of applications. The architecture is based on three concept schemes: *capsules*, *active nodes*, and *code distribution system*.

In ANTS, capsules are the replacements for packets. They represent the most dynamic means of code and service deployment in the network. Their function is to include a reference to the forwarding routine to be used to process the capsule at each active node.

Therefore, references and forwarding routines in ANTS is the equivalent to identifiers and functions in DAN, respectively. Some routines are “well known” in that, they are available at every active node. Other routines may be application specific. Typically, they will not reside at every node but will be transferred to a node by a code distribution mechanism before the capsules of that type can be processed for the first time.

The code distribution system works on a Request-Response pattern, ultimately getting transferred to all the nodes that the first packet of a flow traverses. Subsequent packets can then use the code without the overhead of transporting it.

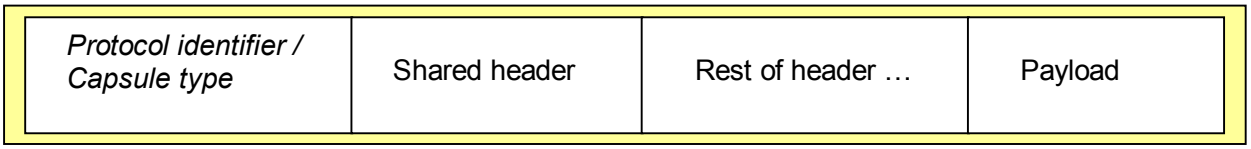


Figure 21: The format of a capsule

Related capsule types form a code group. The forwarding routines of a code group are transferred as a unit by the code distribution system. Related code groups form a protocol. Protocols are the units by which the network as a whole is customized by the applications. Capsules identify their type and the protocol to which they belong. When a capsule arrives at a node, a cache of protocol is checked. If the required code is not all present, a load request for the missing portion of the capsule type and protocol is sent to the upstream neighbor and the capsule put to “sleep.” When the upstream neighbor receives the request, it answers immediately (if possible) and sends the requested code. When the downstream requester receives the code, it caches it and if all the required code becomes avail-able, it “wakes” up the sleeping capsule. If requests for code remain unanswered, sleeping capsules are discarded. On-demand loading and caching is also used in DAN. However, while in ANTS loading takes place between neighbor active nodes, in DAN loading takes place between code servers and nodes.

The capsule format is shown in Figure 21**.** The capsule carries an identifier for its protocol and the particular capsule type within the protocol. The identifier is based on a fingerprint of the protocol code in order to reduce the danger of protocol spoofing and to allow protocols and capsule types to be allocated quickly and in a decentralized fashion. The remainder of the capsule has a shared header that has fields common to all capsules, a type-dependent header, and a payload. The shared header has the source and destination addresses, as well as the information about resource limits to be enforced by nodes.

The protocols need to be executed within a restricted environment that limits their access to shared resources. Active nodes play this role. During the processing, active nodes are responsible for the integrity of the network and handle any errors that may arise. Small tasks are not to be authenticated, but are to be protected by the safety mechanisms of mobile code technology. The use of primitives that manipulate shared logical resources, e.g., updates to the routing tables, must be authenticated. Each capsule has a resource limit that functions as a generalized Time To Live (TTL) field. The nodes decrease this field as resources are consumed and nodes discard capsules when their limit reaches zero.

Finally, forwarding routines are expected to run to completion locally and within a short time, and their memory and bandwidth consumption is bounded.

ANTS is based on Java. The security of the implementation lies in the Java system itself. The choice of Java has allowed the researchers to evolve their architecture quickly but at the cost of less control over resources usage and lower absolute performance.

Recently, an enhanced version of ANTS, known as PANTS, has been proposed at the University of Sidney [Fern00]; it opens the run-time configuration of the NodeOS resources to the EEs.

3.3.2.4 Liquid Software

The *Liquid Software* approach at the University of Arizona ([Hart96], [Hart99]) is designed to build up communication-oriented networked systems with the purpose is *to move* and *transform* data efficiently. In order to achieve this, liquid software implementations use a modified Java Virtual Machine that that allows liquid software to have fine-grained control over the allocation of the system's resources, such as CPU cycles, I/O buffers, and link bandwidth.

The Liquid Software architecture is based on Java as programming language. However, compared to the other components of the system, a Java runtime is much slower. To offset this speed differential and make the active code more efficient, liquid software uses the following two techniques:

- Java-to-C translators in conjunction with C compilers, thus avoiding the need to interpret code at runtime;
- compilers that run quickly at the point of execution.

These architectural implementations allow liquid software to be as usable and flexible as possible, while maintaining performance and security.

3.3.3 ACTIVE HYBRID ARCHITECTURES

Active packets can carry code efficiently only when the code is relatively simple and restricted. On the other hand, active nodes can efficiently provide any code. However, this code is predefined because it should reside in the active node or at least to one node from which it can be downloaded. In the active packets and nodes approach, active packets carry actual code and other more complex code resides in active nodes. Therefore, the merits of the two previous approaches exist in one system. Usually, such architectures allow users to choose either the one or the other approach according to the nature of their application. A typical example is the *SwitchWare* architecture proposed at the University of Pennsylvania. The *NetScript* architecture, proposed at Columbia University, follows its own approach toward programmable networks. However, it will be presented here, as it is relatively similar to the active packets and nodes approach in general.

3.3.3.1 The SwitchWare Architecture

SwitchWare ([Alex98a], [GNS98]) developed at the University of Pennsylvania attempts to balance the flexibility of a programmable network against the safety and security requirements in a shared network such as the Internet. It uses a layered architecture to provide a range of different flexibility, safety and security, performance, and usability tradeoffs. The three layers defined in SwitchWare are Active Packets, Switchlets, and Active Router Infrastructure. The first layer realizes the active packets approach, while the second one --- the active nodes approach. At the operating system level, an active IP-router component is responsible for providing a secure foundation that guarantees system integrity.

In SwitchWare, active packets carry programs consisting of both code and data, and replace both the header and payload of a conventional packet. SwitchWare offers a two-level programming interface to the user. At the lower level are mobile-code packets which carry lightweight programs that invoke node-resident service routines supported by active extensions. The active extensions themselves are always resident on the nodes, where they are loaded by the system administrators or authorized users. The first use of an Active Packet injects the code into the network, and the SwitchWare nodes execute the code in each packet along its delivery path. The mobile code decides the delivery path itself, but cannot explicitly leave state behind at nodes and can access state only through clearly defined interfaces furnished by active extension software. There is much less requirement for testing and verification in the case of active packets than for active extensions, given the confidence that lower level security checks have already been applied to active extensions.

The active extensions are loaded explicitly into each network processing element that will need them, all separate from data delivery. Active extensions are loaded into secure active routers through a set of security mechanisms that include encryption, authentication and program verification. The correct behaviour of active extensions can be verified off-line by applying 'heavyweight' methods, since the deployment of such extensions is done over slow time scales.

SwitchWare uses a domain specific functional scripting language, PLAN (Programming Language for Active Networks, [Hicks98]) for the mobile-code packets, and Caml, a general purpose functional programming language, for the active extensions. PLAN is a lightweight language[48] allowing resource limited computation without the need for authentication. However, it also performs authorized actions when required. Its programs are made secure by greatly restricting their actions. To compensate for this limitation, PLAN programs address routines, called *switchlets*, which can authenticate or use other more heavyweight mechanisms to provide security on an as needed basis. A PLAN program consists of code, plus an indication of which function should be executed first when the program arrives at a router, plus any data that makes up the arguments of that function.

[48] PLAN is a strongly-typed functional language with a similar syntax to those of ML [MTH90]. It supports standard programming features such as functions and arithmetics and features common to functional programming like lists and the list iterator `fold`. The restriction is that functions cannot be recursive and that there is no unbounded looping which heps to guarantee that PLAN programs are terminating. It is a part of a two-level hierarchy architecture SwitchWare PLAN programs/packets are similar to UNIX shell scripts which provide control over utility functions like `sort` and `grep`.

Switchlets form the middle layer of the SwitchWare Architecture. While active packets alone are deliberately limited in power for speed, but active packets combined with switchlets can implement arbitrary protocols or functionality.

Switchlets can be dynamically loaded across the network, but they execute entirely on a particular router. Thus, they are base functionality or dynamic extensions rather than “mobile code.” In the current implementation, switchlets are written in Caml. Switchlets can be subjected to heavier-weight security checks than active packets can. They are statically type-checked on arrival at a router and some may even carry cryptographic signatures. Because of the heavy checking procedures, switchlets can be given more latitude to access facilities in the router that active packets cannot. Thus, they can create or change the state of the router, as well as to directly access to the routers’ network interfaces.

The active router infrastructure is the solid base upon which active packets and switchlets are built. The security of the SwitchWare architecture as a whole is granted in this layer. Below that layer is SANE, an architecture which provides a minimal set of trust assumptions, the ability to securely bootstrap the remainder of the system when the trust assumptions are met, and authentication and naming service for code that is loaded.

The key point of the SwitchWare Architecture is the layered architecture with functionality partitioned between layers based on the flexibility and security tradeoffs required at each layer. Higher layers of the system provide more restricted functionality, but less security risk and very good performance. Lower layers provide arbitrary functionality but, due to the increased security issues, they are not very fast. In general, there is a good tradeoff among security, flexibility, and performance.

3.3.3.2 The NetScript Architecture

The *NetScript* project [NetScript] at Columbia University concentrates a functional language-based approach to program networks efficiently using universal language abstractions. Unlike other active network projects that take a language-based approach Netscript is being developed to support Virtual Active Networks as a programmable abstraction. It is a strongly typed language, which creates universal abstractions for programming network node functions. Abstractions can be systematically composed, verified and maintained.

A distinguishing feature of Netscript is that it seeks to provide a universal language for active networks in a manner that is analogous to postscript. Just as postscript captures the programmability of printer engines, Netscript captures the programmability of network node functions.

NetScript provides architecture for programming networks, architecture of a dynamically programmable network device/node, and a language called NetScript for building network software on a programmable network. NetScript uses also delegated agents to program and control the functions of intermediate network devices/nodes. It allows the mobile code to invoke remotely services resident on the active node.

The NetScript scripting language is being used to compose basic services into more complex ones. Using Netscript interpreters as the runtime environment, programmers can create a dataflow mesh of processes on select network nodes with explicit input and output ports.

The programming language is implemented in Java, and the dataflow abstractions are mapped to Java classes. The dataflow mesh becomes a network service that is applied to select data packets within the network. Within the runtime environment are primitives to control the instantiation of services on the remote nodes.

NetScript views a network as a collection of virtual network engines (VNEs) interconnected by virtual links (VLs). VNEs can be programmed by NetScript agents to process packet streams and relay these streams over VLs to other VNEs. The collection of VNEs and VLs define a NetScript virtual network (NVN). NetScript provides a language to program the NVN. A physical node may be executing many VNEs and a VL may correspond to a collection of physical links and nodes that interconnect VNEs. A VL can also interconnect any number of VNEs to handle broadcast links.

The architecture of the VNE is shown in Figure 22. The Agent Services layer provides a multi-threaded execution environment to support delegation, execution, and control of agent programs. It also supports message communication services among agents. The Connectivity Services module is responsible for interacting with the underlying physical environment to allocate and maintain VLs to neighboring VNEs. It provides a library of primitives used by NetScript programs to control the allocation of VL resources, and the scheduling and transmission of packets over VLs. Packets contain a minimal NetScript encapsulation header that identifies the stream to which they belong. When a packet arrives at a VNE, the run-time environment uses this header to pass it to the programs, which process this stream.

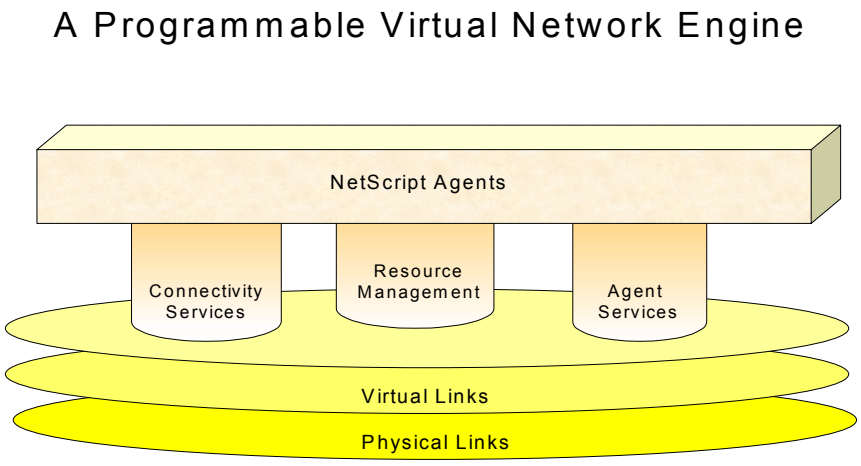


Figure 22: The NetScript programmable virtual network engine

The active packets of the scheme are the NetScript packets and the active nodes are the VNEs. Communication services provided by the VNE are local and permit interaction with neighboring VNEs only.

The NetScript language is a dynamic dataflow language designed specifically for communications-based tasks. It can operate on streams of packets. It is based on simple object-oriented principles, so that programmers can override default operators with customized versions of their own.

A NetScript program consists of a pool of communicating threads. These threads communicate through message streams that connect inputs to outputs of executing programs. Communicating programs can be geographically distributed. NetScript provides a universal abstraction of a programmable networking device because constructs hide the heterogeneity of networking devices behind simple abstractions.

The main difference between NetScript and other architectures is the focus on the programmability of networks. Here, the main assumption is that a single language based on the right model can greatly simplify protocol construction and allow flexibility in experimenting with appropriate programming features. Another difference is that NetScript treats the network as a single programmable abstraction rather than a heterogeneous collection of programmable intermediate nodes and end-nodes.

3.3.3.3 The Magician Architecture

Magician [Kulk98] is a toolkit that provides a framework for creating SmartPackets as well as an environment for executing the SmartPackets. Magician is implemented in Java. In Magician, the executing entity is a Java object whose state has to be preserved as it traverses the active network. Serialization preserves the state of an object so that it can be transported or saved, and recreated later.

Magician provides a model in which an active node is represented as a class hierarchy. Every protocol is derived from an abstract base protocol. Every active node provides some basic functionality in the form of certain default protocols (e.g., routing). Users may prefer to utilize these default protocols if they do not wish to implement their own schemes. To foster privacy and safety, a unique state is created for each invocation of the protocol. State that is common to all invocations of a protocol is inviolable and accessible only to users that have appropriate authorization. Providing each user with a protected copy of the state enables the user to customize his/her, state if necessary.

3.3.4 MODERATE APPROACHES TO ACTIVE NETWORKING

Among the different approaches to active networking, there are also a few architectures, which are based on a moderate philosophy of “limited activeness” and co-existence with the IP world. In the following, two of them are described.

3.3.4.1 The Darwin Approach for Resource Management and QoS Provisioning

The Darwin Project [Chan98a] at Carnegie Mellon University is developing a middleware environment for the next generation IP networks with the goal of offering Internet users a platform for value-added and customizable services. The Darwin project is focused toward *customizable resource management* that supports QoS. Architecturally, the Darwin framework includes Xena, a service broker that maps user requirements to a set of local resources, resource managers that communicate with Xena using the Beagle signaling protocol, and hierarchical scheduling disciplines based on service profiles. The Xena architecture takes the view that the IP forwarding and routing functions should be left intact. It *allows only restricted use of active packet* technology in the system.

Alongside the IP stack, Darwin introduces a control plane that builds on similar concepts such as those leveraged by broadband kernels ([CHLL96], [Laz97]) and active services, [Arb98]. The Xena architecture is programmable and incorporates active technologies in a restricted fashion. A set of service delegates provides support for active packets. Delegates can be dynamically injected into IP routers or servers to support application specific processing (e.g., sophisticated semantic dropping) and value-added services (e.g., transcoders). A distinguishing feature of the Darwin architectural approach is that mechanisms can be customized according to user specific service needs defined by space, organization and time constraints. While these architectural mechanisms are most effective when they work in unison, each mechanism can also be combined with traditional QoS architecture components. For example, the Beagle signaling system could be programmed to support RSVP signaling for resource reservation, while the Xena resource brokers and hierarchical schedulers could support traffic control.

3.3.4.2 The AS1 Approach to Active Services

In contrast to the main research stream in active networking, Amir et al. [AMK99] argue for the preservation of all routing and forwarding semantics of the present day Internet architecture by restricting the computation model to the application layer.

The Active Services version 1 (AS1) programmable service architecture enables clients to download and run service agents at strategic locations inside the network. Service agents called *servents* are restricted from manipulating routing tables and forwarding functions that would contravene the IP-layer integrity. The AS1 architecture is programmable at the application layer and contains a number of architectural components:

- *a service environment*, which defines a programming model and a set of interfaces available to servents;
- *a service-location facility*, which allows clients to ‘rendezvous’ with the AS1 environment by obtaining bootstrapping and configuration mechanisms to instantiate servents;

- servents are launched into the network by an active service control protocol (ASCP), which includes an announce-listen protocol for servers to manage session state consistency, soft-state to manage expiration due to timeouts and multicast damping to avoid flooding the environment with excessive servents;
- *a service management system*, which allocates clusters of resources to servents using admission control and load balancing of servents under high-load conditions;
- *a service control system*, which provides dynamic client control of servents once instantiated within an AS1 architecture;
- *a service attachment facility*, which provides mechanisms for clients that can not interact directly with the AS1 environment through soft-state gateways; and
- *a service composition mechanism*, which allows clients to contact multiple service clusters and interconnect servents running within and across clusters.

AS1 supports a range of application domains such as the MeGa architecture, an active media gateway service, where servents perform application-level rate control and transcoding techniques.

3.3.5 SPAWNING NETWORKS

In [Camp99b], the authors describe spawning networks, a new class of programmable networks that automate the creation, deployment and management of network architectures “on-the-fly”. The term “spawning” is analogous to the definition of spawning child processes in operating systems. Such processes operate typically over the same hardware as the parent process. This approach pursued at the Columbia University in New York envisions programmable networks as having the capability to spawn not processes but complex network architectures [Camp99c].

Thus, larger networks can spawn into distinct "child" virtual networks with their own transport, control and management systems (Figure 23). A child network operates on a subset of its "parent's" network resources and in isolation from other spawned networks.

Spawned child networks can support the controlled access to communities of users with specific connectivity, security and quality of service requirements [Camp99d]. The enabling technology behind spawning is the Genesis Kernel [Camp99b], a virtual network operating system that represents a next-generation approach to the development of network programming environments.

A key capability of Genesis is its ability to support a virtual network life cycle process for the creation and deployment of virtual networks through:

- *profiling*, which captures the “blueprint” of a virtual network architecture in terms of a comprehensive profiling script;
- *spawning*, which executes the profiling script to set-up network topology, and address space and bind transport control and management objects into the physical infrastructure; and
- *management*, which supports virtual network designing and resource control.

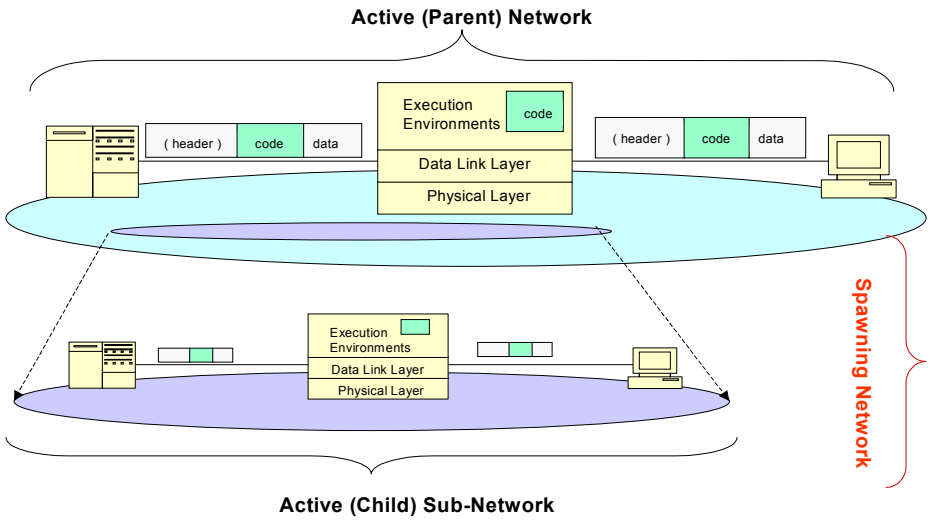


Figure 23: The emergence of a Spawning Network

Virtual networks, spawned by the Genesis Kernel operate in isolation with their traffic being carried securely and independently from other networks. Furthermore, “child” networks, created through spawning by “parent” networks inherit architectural components from their parent networks, including life cycle support. Thus, a child virtual network can be a parent (i.e., provider) to its own child networks, creating a notion of “nested virtual networks” within a virtual network.

3.3.6 SECURITY ARCHITECTURE

Active and programmable networks retain all of the security concerns from traditional networks. The users of the active network will be concerned with the authenticity, the integrity and the confidentiality of the data communicated through the network. But the movement of arbitrary computation into the infrastructure of the network introduces additional security concerns at each level of the architecture. Whereas the concern in traditional networks is focussed on the possibility of damage to user data and end-nodes, active networks must also be concerned with the possibility of damage to the robustness of the active network as a whole through the agency of the active code in the active packets. The active nodes in the infrastructure could be harmed by active code, either by modification of the node state or denial-of-service attacks through excessive resource usage. The execution environments themselves implement their own services and resources that are subject to damage from the active code. Consequently, the security protections must follow the movement of computation into the infrastructure, so that the network, each node and each EE is properly protected.

The network has no single entity to provide its protection; the protection of the robustness of the network as a whole must be built into the design of the individual nodes and EE's. For example[49], some existing EE's have defined a network resource that is consumed during an active packet's traversal of a node. This resource prevents a single active packet from placing an infinite demand on the network. Another example is SNAP (Safe and Nimble Active Packets), [Hicks01a], a special-purpose active networking programming language which has been designed for practical application and scores highly in safety, efficiency and flexibility. It is an expression-limited bytecode language which programs cannot compromise node integrity and cannot consume an unlimited amount of global or local network resources.

The key safety and efficiency gains of SNAP over PLAN come from its model of resource usage. There are no function calls, and all branches and jumps are constrained to move forward through the program, thus preventing looping[50] and causing the number of instructions to be limited by the program length. The execution of SNAP programs consumes time, space and bandwidth linearly to the length of the program.

3.3.6.1 Node Security

In contrast to the network, protection of an active node or execution environment can be *self enforced*. The active nodes protect themselves by ensuring that activity within the node on behalf of the active packet is safe and properly authorized. Safety can be ensured by the use of a variety of techniques, such as safe languages or software fault isolation. Authorization is expressed in terms of *access to portions of the node state*, either to protect its confidentiality (e.g., preventing unauthorized exposure of any keying material used to cryptographically protect the node's communications) or its content (e.g., preventing unauthorized modification of the node configuration parameters). Authorization is also expressed in terms of limitations on resource usage. The fundamental motivation for such limitations is to prevent denial-of-service attacks against the node (and thereby indirectly against the network). Validating the source authenticity and integrity of the active code are crucial in ensuring proper authorization. If the source can be spoofed, then the node can be deceived into taking actions, based on the apparent source, which will cause harm. If the active code can be changed, then it can be made to perform actions other than those intended by the author, possibly inducing harm as a result.

The execution environments offer services and resources, composed from the node services and resources, to the active code. Consequently, the EE will want to ensure that active code that uses or accesses its resources or services has the proper authorization. It will want to protect the integrity, the authenticity and even the confidentiality of those services and resources.

[49] The techniques presented below outline the basic research issues in AN security w.r.t. application level networking. A thorough examination of AN security at each network layer and their impacts on end-to-end security can be found in [Brown01].

[50] which permits to prove several safety properties about the language.

3.3.6.2 Authorization

Authorization in active networks is the fundamental security service that must be provided to protect either the active node or the execution environment. The difficulty is in providing a mechanism for authorization that is suited to the scale of active networks - anything from an enterprise network governed by a single administration with a homogeneous model and policy to a wide area network including a multitude of authorization models and policies.

For scalability and manageability, authorization cannot always be expressed in terms of unique entities and single services or resources. Aggregates of entities must be created, identified by some security attribute that policy relates to authorization for access to services and resources.

For further abstraction, the services and resources could be similarly aggregated and identified by security attributes, so that authorization is decided using the requesting entity's security attribute and the service's or resource's security attributes. These aggregates could be any of the aggregates that have been widely used in creating secure systems, such as roles, groups, and an ordered set of labels, domains or types.

3.3.6.3 Security Policy Enforcement

Users and other entities in the network (e.g. other nodes) are represented by an abstraction called the *principa*l. This principal may be some user, or the EE itself. Security policies are defined in terms of principals and the NodeOS is responsible for the enforcement of these policies authorizing the access to the NodeOS services and resources. Because the flows are the requesters of memory, threads, and channels, a principal must be associated with each flow as it is created. Yet, it is not necessary that each flow have a separate principal.

In keeping with the spirit of active networks, it should be possible to modify the policy dynamically. Dynamic modification of the policy requires a separation of the storage of the policy and the enforcement of the policy into a policy database and an *enforcement engine* (Figure 11) respectively. The enforcement engine in the NodeOS refers to the policy database to decide each controlled access. Thus, changes to the policy database, possibly through active code, can effect in an immediate change in policy.

Security policies are invoked as follows. When an EE requests a service from the NodeOS, the request is accompanied by an identifier for the principal on whose behalf the request is made. The NodeOS presents this information to the enforcement engine, which verifies the authenticity and checks whether the policy database authorizes the principal to receive the requested service or perform the requested operation. It is the authorization of this principal that decides the accesses and usage of a flow.

Security policies are implemented in a "nested" hierarchy scenario within the active network. Thus, execution environments may invoke their own policies to augment those of the node. For instance, an EE may offer specialized routing services and a routing table to the active code it executes. The execution environment has a policy governing who is allowed to use the routing table, who is allowed to update the routing table, and who is allowed to use the routing services.

However, each of these services and resources is composed from services and resources provided by the NodeOS and access to the NodeOS level objects must be decided in accordance with the Node OS policy. The NodeOS policy and the execution environment policy may conflict. There is no guarantee that the NodeOS policy for access to the memory pool that underlies the routing table, for example, meets the execution environment's policy. Therefore, the NodeOS policy must be enforced but it must be possible for the execution environment to enforce a stricter policy. The "local" EE policies are not allowed to override the NodeOS policies.

There are several ways to divide the responsibility for enforcement of policy between the execution environment and the NodeOS. For instance, an EE could choose to delegate all authorization decisions to the NodeOS[51]. Another, easier solution would be for the execution environment to enforce its own policy, with its own policy database and its own enforcement engine. Indeed, the execution environment cannot be prevented from performing authorization checks of its own for a service or resource it provides. If it denies access, then the access or usage is not provided[52]. After the execution environment has decided to grant access to a service or resource, it will attempt to access the NodeOS services and resources from which its own services and resources are constructed. The NodeOS can enforce its own policy when deciding access to the underlying NodeOS services and resources.

When making use of the NodeOS services and resources, the execution environment may or may not indicate the principal associated with the active code that induced the request. There is no way to force the execution environment to do so, as the execution environment may make requests unrelated to any active code. If it does not indicate the principal behind the request, then the NodeOS makes the decision based on the principal associated with the execution environment's flow. The execution environment would be accountable for the usage of NodeOS resources. However, the NodeOS cannot ensure that the execution environment enforces the NodeOS policy in this case. It must *trust* the execution environment to manage the resources granted to it in accordance with the NodeOS policy. This means that the NodeOS must communicate its policy to the execution environment on start-up, as an initial policy database.

3.3.6.4 Identification of Principals

The final problem remains the distribution of the representation of principals and their authorizations. In the trusted environment of an enterprise network, it may be possible to store identifiers and security attributes directly in the packet. In such an environment, the meaning of identifiers and the policy governing such identifiers are known everywhere and the source and integrity of the packets is trusted[53].

---

[51] This would require that the execution environment communicate to the NodeOS its policy governing access to those NodeOS services and resources that underlay each of its own services and resources. In response, the NodeOS would have to be capable to partition the resources it has associated with the execution environment according to the "local" EE's policy.

[52] In turn, a NodeOS policy cannot *guarantee* provision of an new service for a channel that is anchored in an EE. It must *trust* that the EE will perform the service.

[53] For instance, NFS packets carry the UNIX user ID and user's group ID used in making file access checks between the file server and the client. The user ID's and group ID's are trusted to have the same semantics, represent the same principal, throughout the NFS network.

However, in more wide spread and heterogeneous environments, identification of principals and their authorizations can not presumed to be widely known, uniform or trusted. A system of authorities must be established who can attest to the authorizations granted to a principal.

In an un-trusted environment, cryptography is used to create a binding of principal and authorizations and the packet. Normally, the principal is associated with a key that is used to provide some cryptographic service for the principal. An authority binds the keying material and some designation of the authorization of the principal into a *certificate* or *credentia*l.

The credential carries the *digital signature* of the authority, using a key bound to the authority. An infrastructure of authorities is created, with authorities attesting to the binding of keys to other authorities. Finally, the credential must be bound to the packet by the initiator to prevent its use in another packet. The usual mechanism is to employ the key represented in the credential to cryptographically bind the packet and credential with a digital signature.

One active packet can contain the credentials of multiple entities. The authorizations associated with the initiator of the packet are clearly important to security. The authorization of the author of the active code could be important in some node policies. Relevant attributes of the code, such as the proof for proof-carrying code or evaluations of its quality by independent authorities, might be part of some node policies. A significant challenge in active networks will be to create enough uniformity of security requirements that application developers can comply with these requirements and effectively use the security features of the network, while still maintaining the flexibility to tailor security services to the purposes of the applications and individual networks.

Finally, the source authenticity and integrity of the packet must be maintained to derive any assurance from the credentials contained in the packet. When the principal threat is from external intruders, protection of the integrity of the packet between neighboring active nodes can be accomplished with low cost symmetric cryptographic algorithms using shared keys. This ensures that the packet coming from a neighboring node has not been modified in transit. However, when it is not possible to trust implicitly every active node in the network, more assurance than just the immediate source of the packet is needed. Hence, the ultimate source and integrity of the packet must be ensured. A digital signature that binds a credential to a packet also provides this protection if the credential is associated with the source of the packet.

3.3.7 IMPLEMENTATION FRAMEWORK

3.3.7.1 Code Distribution Mechanisms

Code distribution mechanisms such as the Active Network Encapsulation Protocol[54] (ANEP, [Alex97]) provide the capability to route the active packets to a specific EE at a node. Thus, a packet needs to contain an ANEP header in order to be processed by an EE.

[54] Inside their headers, active packets are carrying identifiers for the target handler, the operating element they should be are delivered to. ANEP is used when the target handler is an executing environment, but it can be also an IP router state or a dynamic handler (e.g. a flow).

The ANEP header includes a *type identifier (TID)* field. If a particular EE is present at a node, packets containing a valid ANEP header with the appropriate TID will be routed to the appropriate EE. Some TIDs are assigned to specific execution environments. By setting up the appropriate channels, EEs can also process legacy traffic from end systems which are *not active-aware*.

3.3.7.2 Service Creation and Composite Network Services

An essential feature of the network API inherited from the IN domain is the ability to compose new network services from a set of service independent building blocks (SIBBs) known as *components*. The network API includes a "mini-compiler" for services, a *composition mechanism* which performs in a similar way as the Service Creation Environment for intelligent networks to produce a *composite service* from components.

The modular approach to service design allows price reduction (reusable software) and better proof of correctness of the overall service specification. A composite service may perform on a single active node or on a set of active nodes. The implementation of the network API can support a variety of techniques among which are:

- *Selection schemes:* the choice of a specific service from a fixed set.
- *Turing-complete language:* the generation of an infinite set of components using the sequence control mechanisms of a programming language, (ANTS [WGT98]; components installed in the active node as Java servlets called by the Java composite service).
- *Special-purpose language:* a service creation language restricted to preserving certain desirable properties of the composite service, (SwitchWare [Alex98a], PLAN [Hicks99] and NetScript [YeSi96], [NetScript]).
- *Event-based framework:* incorporating dynamic behaviour into a composite service by structuring the process as an event-driven computation and binding the code modules to specific events; this approach has been used for composing *micro-protocols* in the X-kernel [BS95], and for injecting customer specific programs into an *underlying program* of the active node which provides a basic service through a set of processing slots associated with execution points of the underlying program, LIANE [Calv98].

3.3.7.3 Quality of Service Policy

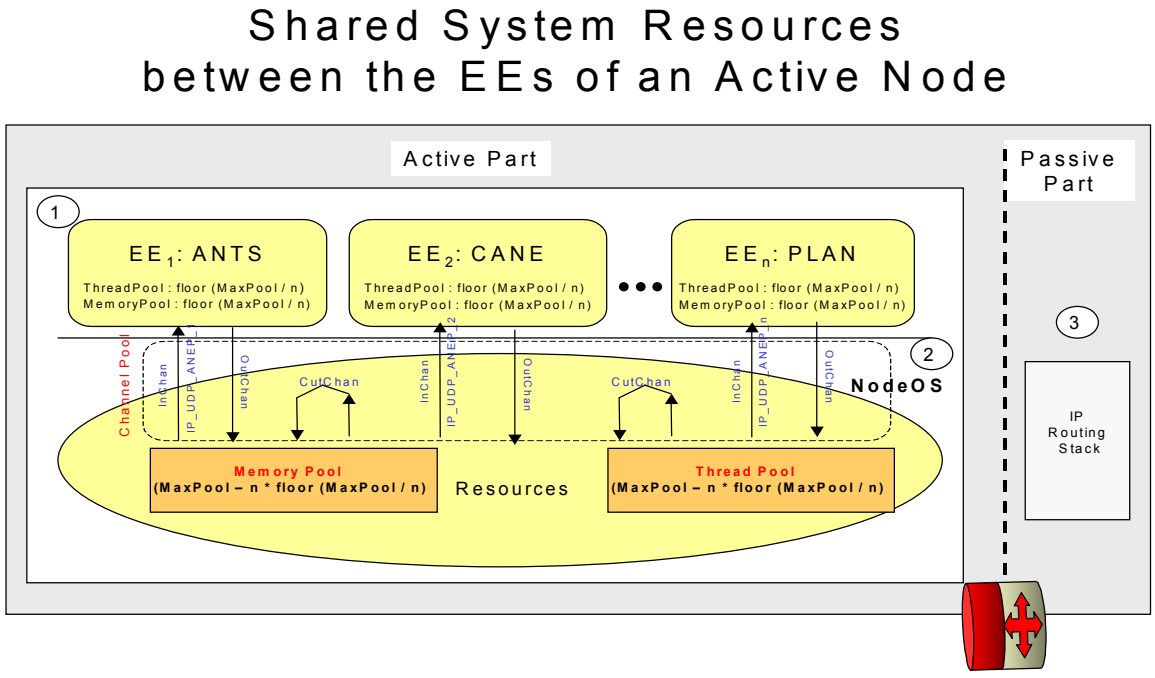
The provision of QoS is ensured by the NodeOS' scheduling mechanisms which control the access to the processing and transmission resources of the active node in such a way that each user's traffic appears to have its own virtual machine and/or virtual link. When channels are created, the requesting EE specifies the desired policy by the scheduler(s). Such policy may include the reservation of a specific amount of bandwidth for traffic on the channel, channel bundling for excess traffic, or isolation from other traffic on the channel and *fair sharing* of available bandwidth with other channels.

3.3.7.4 ABone

The Active Bone (ABone) [Bra98] is a research test-bed used for testing EEs and deploying Active Applications (AAs) which use the EEs. The ABone nodes are currently UNIX machines operating the *anetd* (active net demon) program which performs active network node management, such as initiating EEs. Currently, the ABone is flat, although Braden's proposal [Bra98] suggests that a hierarchy based on locally-administered *Ahosts* at edges and intermediate *Arouters* at the core may accelerate progress towards the goal of a 1000+ node Active Network. Currently, the ABone supports ANTS, NetScript and PLAN/Alien.

3.3.8 SUMMARY: THE PRO-ACTIVE ARGUMENTS

The architectures for AN execution environments described in this section have all been developed for different active applications and with different purposes in mind. They all represent the variety of software approaches in which programmability can be introduced into the network. The list is not exhaustive, but gives an overview of the various methodologies that can be adopted. Figure 24 illustrates how different EEs can co-exist on a single active node.



Legend:

EE	Execution Environment	NodeOS	Node Operating System
ANEP	Active Network Encapsulation Protocol	ANTS	Active Network Transfer System
InChan	Input Channel	CANE	Composable Active Network Element
OutChan	Output Channel	PLAN	Packet Language for Active Networks

Figure 24: Resource distribution among multiple different EEs on a single active node

3.3.8.1 Why are Active Networks needed ?

The dynamic control provided by active networks is potentially useful for a number of reasons. A few of them are described below:

- **rapid development and deployment of new services**. This is the biggest benefit of Active Networks from the viewpoint of the network service provider. The shared infrastructure of the network currently evolves at a much slower rate than other computing services. Therefore, the deployment of any new service requires a lengthy standardization process, whereby everyone involved should agree on a generic procedure for a solution framework (the standard). This has led to a huge backlog in network services waiting to be universally deployed. RSVP and IPv6 are two well-known examples. The ability to change the behavior of network nodes on the fly is expected to simplify greatly the process of deploying new network services.

- **dynamic customization of network services and resource allocation**. At a finer level of granularity, Active Networks might enable users or third parties to create and tailor services to their particular topology, applications and even to prevailing network conditions. This should make it possible to develop a much richer class of applications than the ones that are currently deployed.

- **open network management and administration**. Active Networks are open to deploy and administer. For researchers, a dynamically programmable network offers a platform for experimenting with new network services and features on a realistic scale without disrupting the regular network infrastructure.

3.3.8.2 Major Challenges for Implementing an Active Network

In order to be successful, any implementation of a programmable active network should satisfy the following requirements:

- The network services should be usable: any active programming model will impose new and unfamiliar programming models and constraints on the user. For ease of use, it is important that these constraints should be as limited as possible.
- The network should be highly flexible: Flexibility is the primary reason behind the Active Networks research. The network subsystem should be adaptable to a wide variety of tasks and applications.
- The implementation should be secure: Security is expected to be the major obstacle to large-scale deployment of active networks in the future.

The network should guarantee high performance: performance is usually the price that we have to pay for flexibility. Therefore, the deployment of Active Networks should not create new bottlenecks in the network infrastructure. In particular, the implementation should provide for fast and transparent path processing of non-active packets.

3.4 APPLICATIONS

Active networks enable the introduction of new applications that rely on network-based services. In the following, we present a short review of the most representative ones.

Active Network Management could track and repair network problems without having a working connection between the concerned component and the management server. The conventional approach is to poll managed devices from a management station (e.g. an SMP in Intelligent Networks), requesting the values of variables and checking for anomalies (e.g. via SNMP). This centralized approach of intelligence which is still suitable for telephony services, may result in processing and communication bottlenecks when multi-layered, multi-session applications such as video conferencing are deployed.

Further, the poll-and-check approach severely limits the ability to track problems in a timed and efficient manner. Active networking automatically distributes network management. Many network management tasks consist of collecting data, such as event counts, most of which are *local*. Hence, network components such as routers can take on a degree of responsibility for *monitoring themselves* (e.g., by injecting customized monitoring and diagnostic programs into their nearest neighbors). Furthermore, active technologies can be used to implement sophisticated approaches to network monitoring and event filtering. Thus, distributed intelligence can be used to filter out uninteresting events from exception indications in a differentiated fashion.

An active node may tailor management data and policies according to the local network behavior and delegate control depending on its own status. Finally, active networks provide the flexibility necessary to improve fault detection and to update the survivability policies, which govern the node response to correlated failures, such as those caused by earthquakes or malicious intruders. Several projects such as *NetScript* [YeSi96], *SmartPackets* [SmartP99], *Active Engine* [[RaSh00] and SwitchWare [MMN01] apply AN technology for improving network management.

Active Caching addresses self-organizing caches that determine dynamically where to place themselves depending on current demand for data. The caching of objects close to the clients is an important technique to reduce both network traffic and response time for Web applications. For example, web proxies that cache pages of information of a multi-user service could benefit from network-based computation and storage. A further argument in favor of using active technologies for Web caching is that a significant fraction of web pages is dynamically computed and not susceptible to passive caching. This suggests the development of schemes that support *active caches*, which store and execute programs that generate these pages.

Hierarchical caching schemes such as Harvest [ChDN96] can reduce the latencies experienced by individual users and the aggregate bandwidth consumed not only by relocating the cached information, but also the caches themselves. The cache nodes are presently configured manually and located near the edges of the network (static hierarchy), i.e. at nodes within the end-user organizations. These systems could be extended to adapt to dynamic conditions by allowing nodes of the hierarchy to be located at strategic points within the networks of Internet access providers and inter-exchange carriers.

Furthermore, active networking can deploy routing mechanisms for cache requests to pre-configured cache locations [LWG98]. In addition, caches can be made *aware* about the contents of nearby caches at each network node [BCZ98]. Finally, a scaleable, adaptive Web caching architecture can be realized using AN techniques and application layer protocols [Zhang98].

Active Security and Safety issues address user-aware network protection**.** Security problems may result e.g. in malfunctioning networks, loss of privacy, or attacks against other parts of the network. Protection of information means that the right information gets to the right people at the right place and time. Security and safety in networks are major concerns since programs run on sensitive components like routers. Despite the progress achieved in networking forums ([Bore94], [GMcG95], [ATab96]) the active approach may accelerate the design of an integrated mechanism that governs all network resources. This eliminates the need for multiple security/authentication systems operating independently at each communication protocol layer and allows the user to program in security policy for the network on a per-user or per-use basis.

The most promising techniques include:

- authentication of packets (author and ID of packet)
- monitoring & control restricting packet access to resources based on ID and other criteria
- limitation techniques allowing to specify time and range limits
- code carrying proof of the correctness when given specification
- fault tolerant systems
- encryption

Finally, a formal approach using rigorous specifications and language-enforced-type safety can be used to reason about the protection policies and the mechanisms of their implementation.

Other candidates for applying active techniques are such network services as:

Active Congestion Control (ACC). There will always be applications that prefer to use best-effort service and dynamically adjust rate. The *sender adaptation* model has worked well with IP networks. However, it has well-known bottlenecks: difficulty to detect congestion and determine the increase in available bandwidth, time required to detect congestion and adjust rate. The observation is that the application knows *how* to adapt to congestion, while the network knows *when* to adapt. Thus, we have to move advice about the adaptation into the network. It is reasonable to insert buffers along the network that work according to the available bandwidth, convert data, and provide connection-aware, application-aware and semantic data dropping (e.g. in MPEG, drop P and B frames).

Due to its flexibility and presence in the network, active networking offers considerable promise for improving congestion control ([Bhat96], [ASNS97]. This promise has been explored in the design of Active Congestion Control [Fab98], [Fab02] (ACC). ACC takes advantage of state and programmability to improve feedback congestion control by reducing the delay with which congestion is signalled to sending systems. ACC packets contain small (e.g., 4–8 byte) characterizations of the state of the endpoint's congestion feedback scheme (perhaps a congestion window size at the sender).

Category	Description
Firewalls	Firewalls implement filters that determine which packets should be passed transparently and which should be blocked. Although they have a peer relationship to other routers, they implement application and user specific functions in addition to packet routing. The need to update the firewall to enable the use of new protocols is an impediment to their adoption. In an active network, this process can be automated by allowing applications from approved vendors (e.g. McAffee) to authenticate themselves to the firewall and inject the appropriate modules into it.
Web proxies	Web proxies provide a user-transparent service tailored to the serving and cashing of Web pages. Cash nodes can be located near the edges of a network and close to the end user. With an active network, this system can be extended in a hierarchical way by allowing cache nodes to be located (moved) at strategic points within the network.
Nomadic routers	Nomadic routers are interposed between an end system and the network. These modules observe and adapt to the means in which the end system is connected to the network (ISDN, LAN, etc.). An active network can intelligently decide to perform more file caching or link compression when the end system is connected through a low bandwidth link and invoke additional caching security (encryption) when operating away from the home office.
Transport gateways	Transport gateways are nodes located at strategic points that bridge networks with different bandwidths and reliability characteristics (e.g. at the junctions between wireless and wired networks). An active network can support QoS for mobile access to wired networks by allowing TCP snooping to retain *per-connection state information* at wireless base stations.
Application servers	Application-specific gateways support services such as the transcoding of images among video conference users with different bandwidth constraints, as well as voice and handwriting recognition. In an active network, a server's functionality can be requested, configured, designed and injected by the user himself.

Table 1: Network elements and their "activation" [55]

When congestion occurs at a router, in particular, when a packet is dropped, the router (1) determines what congestion window size will result presuming the packet is dropped, (2) deletes packets that would not be sent with this new window size, and (3) informs the sender of the new window size.

[55] adapted from [Tenn97]

As noted by Faber [Fab98], nodes beyond the congestion point see traffic which looks as if the sender had reacted instantly. Thus, ACC is particularly powerful in network contexts which have large *bandwidth * delay* characteristics. ACC was evaluated by applying the basic model in the context of TCP/IP congestion control (thus ACC was presumed to be embedded in IP routers). The simulation studies showed that ACC can achieve up to an 18% improvement in throughput when traffic is bursty.

Other promising AN techniques for congestion control include the Active IP option, [WeTe96], and the Configurable Active Node Elements (CANEs[56], [CANES]).

Active Reliable Multicast (ARM). The traditional IP multicast service hides the details of the routing topology and the number and location of receivers from the user. For unreliable multicast this approach does certainly make sense, as it allows scaling to larger applications. However, this model is inappropriate when reliable data delivery to all receivers is required. For instance, losses typically affect all receivers downstream in the multicast tree. Active networking can reduce the delay and transmission resources and avoid overloading the transmitter by making receivers *aware* about their neighborhood. The responsibility for multicast retransmissions is spread out throughout the receivers. This can be achieved by using caches at the (active) network nodes [LWG98], or by including information about a node's state and processing in the transmitted (active) packets [PPV98] for controlled redirection. Furthermore, the active approach is useful for applying such multicast techniques as:

- suppressing negative acknowledgements (NACKs) for originators known to be repaired in a short time (duplicate NACK suppression to inhibit NACK implosion);
- using "best-effort" caching of multicast data: dynamically moving caches for multicast and repair packets to "strategic" routers (e.g. such before lossy[57] wireless links); and
- using "local" multicasts in the retransmission scheme to reduce bandwidth: selectively sending repair packets to only hosts that requested them.

ARM has the desirable architectural properties that not all nodes need be active, and that there are no "necessary" routers in the loss recovery scheme. The result is a robust scalable system for reliable multicast, which shows significant benefits (in recovery latency, scalability, reduced bandwidth required for recovery, *etc*.)

Active Merging and Distribution of Information. Existing systems are based on a service that provides an extremely limited function (i.e., the copying of IP packets) without support for application-specific distribution, let alone network-based storage or information fusion. However, the era of multi-user, multi-site applications has already begun with Mbone. Many applications require network-based services to support the merging and distribution of information.

[56] based on triggers indicating congestion control and examining the flow state to derive advice about how to reduce the amount of data

[57] e.g., the improved loss-recovery of the MIT's ARM architecture comes from 3 sources: a) duplicate NACK suppression which inhibits NACK implosion; b) best-effort caching of multicast data for recovery in case of retransmission; c) local multicasts in retransmission which reduces the required bandwidth for such data.

Active QoS Management. Network congestion or lossy links can significantly degrade the quality of application streams. The adaptation of a transmitter to the network conditions is featured by *latency* (the time for detecting the condition, react, and transmit the adapted data to the receiver) and *performance decrease* (uncontrolled packet loss and non-optimal throughput during the adaptation period, which can be compensated e.g. by an AN fusion server). By re-allocating the adaptive processing into the network, the appropriate type of adaptation can occur at the required node and time. Efforts in this area include transparent in-line protocol "boosters", e.g. by adding forward error correction over error-prone links [Bakin97], intelligent discard strategies [BCZ97], active plug-ins [Deca99] and end-to-end ANN feedback [SpMe99] for preserving the MPEG-2 video quality at network congestion points.

Protocol Boosters are protocol elements that can be inserted into or removed from existing protocols with the intention of building protocols dynamically and on request ([Feld98], [Marc98]). They do not require re-implementation of existing protocols and applications.

Protocol booster architectures, also called Performance Enhancing Proxies (PEP), integrate performance-enhancing functionality that can be located at the edges of the wireless part of the network. These protocol boosters operate *transparently* without the need to modify the existing IP suite. To enable efficient operation the boosters have to be designed for specific applications. For the case of TCP applications, IP booster architectures can double TCP throughput even under noise propagation conditions. Real-time video conferencing applications require other specific booster functionalities.

The boosting principle is illustrated on Figure 25. Host A and B act as server and client in an application scenario (e.g. video streaming) respectively. Since the network segment between the router R and client B is congested for some reason, the router acts as a bit rate adaptation proxy for the specified part of the network. Such architectures are well known and preceded research in Active Networks. Because the methodology for dynamic construction allows rapid development of specialized protocols from modules (servlets, *netlets,* etc.), this technique has the advantage of increasing both protocol performance and the rate of network technology evolution. Thus, Active Networks provide an ideal infrastructure for implementing protocol boosters.

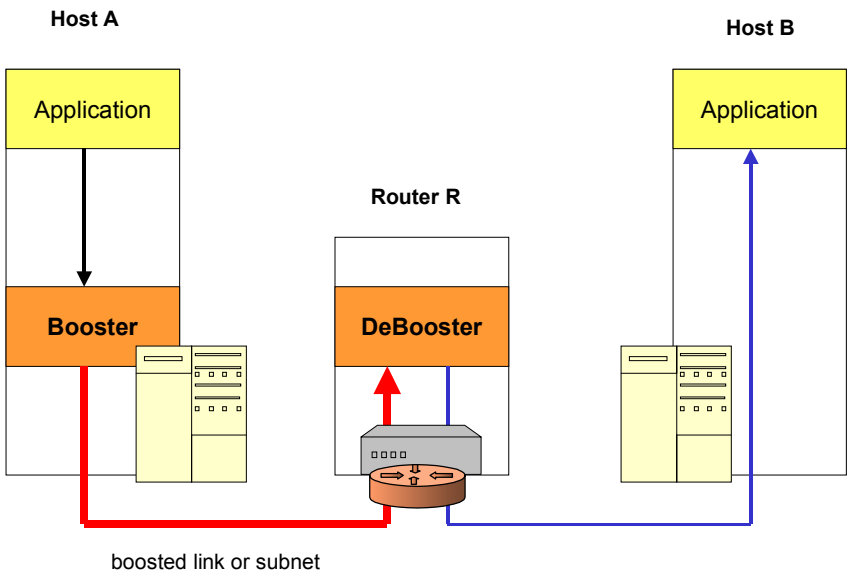


Figure 25: A protocol booster architecture

By design, the boosting mechanism is under control of a policy, which determines when augmentation is required. Thus, many portions of a protocol stack execute only as necessary, permitting significant increases in performance relative to general-purpose protocols.

Building protocols dynamically in this manner can aid both protocol performance and the rate at which the network technology evolves, because specialized protocols can be developed rapidly and implemented at desired locations in the network.

Protocol boosters have been initially developed for error and congestion control, but they can be applied for nearly any kind of local protocol stack enhancement or adaptation. Figure 26 illustrates the realisation of a DLL feedback mechanism as a protocol booster for multimedia traffic in a *performance enhanced proxy* (PEP, [BKGM00], [Sim00]) at the edge of a wireless access network, e.g. as a front-end to an UMTS radio node controller (RNC).

The idea behind is that the DLL protocols in the radio access network, -- packet data control (PDCP), radio link control (RLC), radio resource control (RRC) and media access control (MAC) -- , have much closer “look” at the network perturbances than the upper layer multimedia application (e.g. an MPEG-7 streaming video), so that important information about delay, frame loss and traffic congestion on the air interface can be much faster detected in a client proxy (than on the client itself) to be selectively reported to the application in the server enabling a real-time adaptive bit rate encoding compensating the (temporary) failures.

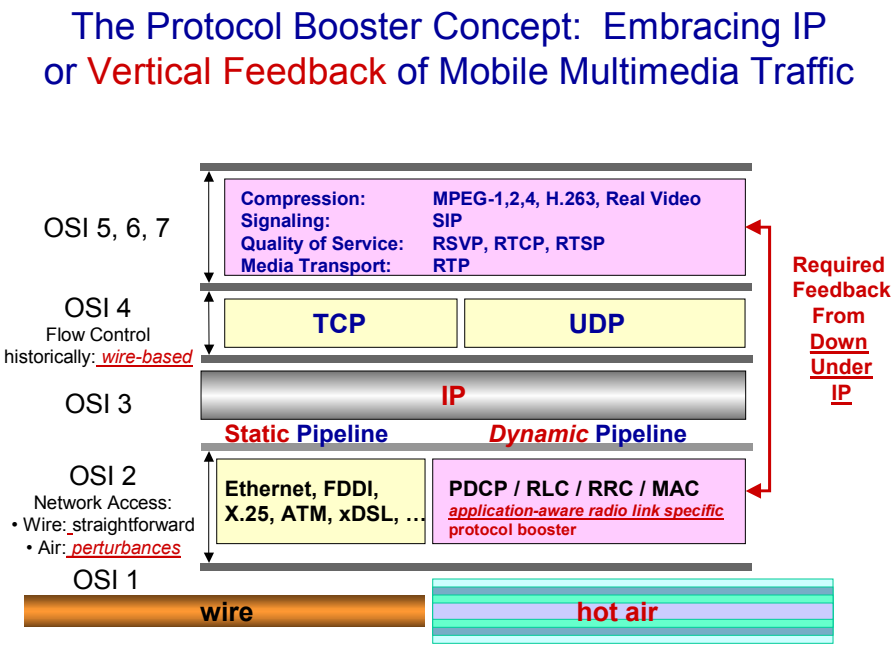


Figure 26: A protocol booster for error resilience of multimedia traffic in mobile wireless networks

Active Router Control (ARC) [SAMS98] represents an option for rapid deployment of active technology to enhance the existing Internet where active elements/routers serve as an enhanced programmable management and measurement system. An example for such a configuration is shown in Figure 27 using a set of routers as a logical “router in a room”.

The basic active “cloud” can be replicated throughout the Internet, with the active elements using the managed Internet routes as link layers. A set of active nodes can be grafted into a larger collection of routers and forwarders to create an active Internet. Researchers at the University of Washington have proposed a similar idea in their Detour [SAAB98] project, but using a high-performance workstation cluster to provide a different blend of computing performance.

Active Routers are obviously the key elements of future networks. Recently, some new approaches ([Kell00], [SHB00]) addressed routers directly to support applications and improve network performance at congestion points or at the edges (on a per-user basis).

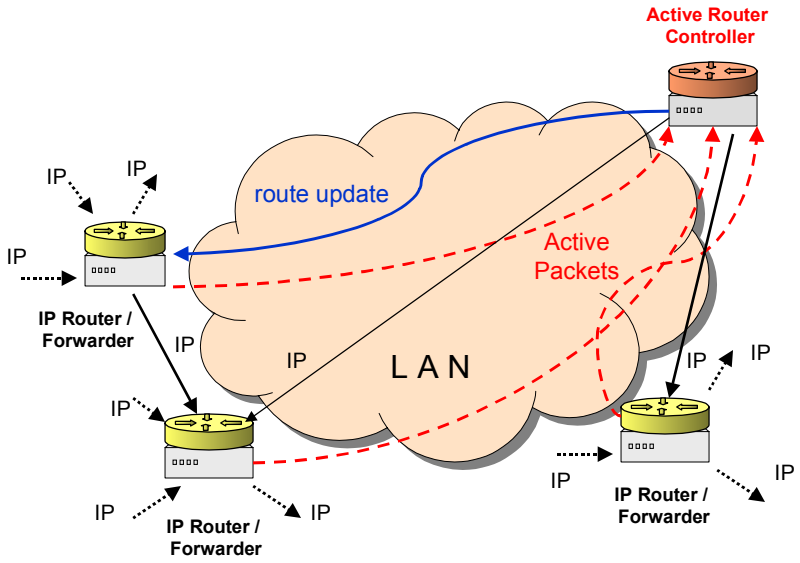


Figure 27: An Active Router Controller (ARC) managing a set of forwarder/ routers

Active Applications. Application Level Active Networking, ALAN [Fry99], is a framework for deploying applications on Active Networks, developed at the University of Technology, Sydney. The ALAN system consists of regular clients and servers, such as WWW browsers and servers, located on the Internet or Intranet. Communication between servers and clients is enhanced by *Dynamic Proxy Servers* (DPS) located at optimal points of the end-to-end path between the server and the client. There may be more than one DPS involved in an end-to-end path. It is possible to download protocol entities onto the DPS infrastructure. These protocol entities, *proxylets*, then act as filters or enhanced protocol functionalities that improve the level of service between servers and clients. Proxylets may be obtained from protocol servers, or Web servers, owned by network operators or value added service providers. ALAN implementations include such applications as WWW streaming audio and hypertext (html) transcoding.

Another approach to support active applications is based on specialized servers such as the *on-line auction servers* which collect bids from clients for items and provides the current price of an item on request. Given the time-scales with which such servers can operate (short) and the possible delays in packet arrival in an internet (long), outdated (i.e., lower than the current price) bids can arrive. Depending on the scale of the auction, these bids, which have become irrelevant, can generate considerable server load.

The active protocol suggested by Legedza, *et al.*, [LWG98], deletes low bids with active elements embedded in the network connecting client and server. When the server becomes busy, it enables these elements and periodically updates their notion of the current price. In dropping low bids, they thus serve as proxies for the server in its attempts to focus on relevant bids.

3.5 MAINSTREAM DIRECTIONS

The three basic Active Networking approaches outlined in Chapter 2, -- the "option" model, the "programmable switch" model and the "capsule" model , -- have been realized in a number of architectures with the main objective to establish some level of programmability in the network.

At the same time, other projects with open signaling and "pure" IP application background have been carried out into designs and test-beds pursuing with the same goal, yet expecting a more moderate impact on the present day telecommunications infrastructure.

In their survey of programmable networks, Campbell *et al.* [Camp99a] introduce four basic characteristics along which they classify the diverse research approaches with respect to their generalized framework architecture (Figure 3):

- *networking technology*, which implicitly limits the level of programmability that can be delivered to the higher levels. For instance, some technologies such are more "QoS programmable", (e.g., ATM), scalable (e.g., Internet) or limited in bandwidth availability (e.g., mobile networks);

- *level of programmability*, which indicates the method, granularity and the time scale over which new services can be introduced into the network infrastructure. This in turn is strongly related to language support, programming methodology or the middleware architecture. For example, distributed object technology can be based on RPC and CORBA [Vin97] or mobile code [WGT98] methodologies resulting in *quasi-static,* [OPENS], or *dynamically composed* network programming interfaces, [DoDAN];

- *programmable communications abstractions*, which indicate the level of virtualization and programmability of networking infrastructure requiring different middleware and network node support (e.g., switch/router, base station). For example, programmable communications abstractions include virtual switches [Laz97], switchlets [Merw97b], active nodes [NOSIS], universal mobile channels [LC98] and virtual active networks [NetScript]; and

- *architectural domain*, which indicates the targeted architectural or application domain (e.g., signaling, management, transport). This potentially determines certain design choices and affects the construction of architectures, and services offered, calling for a wide range of middleware support. Examples include, composing application services [Arb98], programmable QoS control [Laz97] and network management [SmartP99]).

Psounis [Pso99] presents a simple qualitative comparison of the programmable networks approaches with respect to the above characteristics aligned to the generalized *Columbia Model* for programmable networks, [Camp99a]. Aside from the above characteristics, these architectures differ from each other in the following details:

- **Language expressive power:** The "languages" that can be used to "program" the network nodes poses a wide range of expressive power and functionality. On the one hand, there are "languages" consisting of a simple list of fixed sized-parameters that select from pre-defined sets of choices for (re-)configuration. Such a system provides for easy-to-enforce security and simple fast-path optimizations of the traditional packets. The current IP Internet and active networks architectures belonging to the “option” school, use such a "language". On the other hand, there are also Turing-complete languages capable to specify and perform any programming task. Security is a major concern for such a paradigm, and strict type-safety and other checks have to be maintained.

- **Statefulness:** This refers to the ability of active packets to install state on the network nodes. The present *simple* IP network does not allow installation of state; however, most AN implementations allow user packets to install soft state on the nodes.

- **Granularity of control:** This refers to the visibility that modifications introduced to network nodes by active packets should have. The granularity ranges from making these modifications visible to all nodes on the way of a packet, to making them visible only to the packet making the modification. The first approach raises serious security issues. Therefore, it is not adopted by any implementation. The latter is implemented in the current IP architecture and some proposed Active Networks designs. Most Active Networks designs speak of implementing *per-flow* granularity, which is intermediate to the two alternatives mentioned above.

The rising interest of the research society ever since, and recent discussions at international forums such as IETF, testify that *active networking* is a promising approach for network evolution which allows the rapid introduction of new services and techniques for their deployment without the necessity of time-consuming standardization.

One of the major advantages of active networking is that distributed computing algorithms can be more easily implemented and quickly deployed in an active network environment. The goal of most recent AN research is the further development of the *distribution* and *virtualisation paradigms* in order to enable active networks (a) to generate some parts of their virtual architectures “on demand”[58] (e.g. a the Genesis project [Camp99b]) and (b) to predict their own behaviour [Bush00] while targeting some predictive optimisation strategy, such as traffic control [BKEG00], resource reservation ([Galt01a], [Galt01b]) or network management [BuKu01]. This research concentrates on the development of distributed computing techniques, such as logical processes (e.g. spawning, concatenation, etc.) by means of the design and analysis of new types of architectures and algorithms, e.g. Active Virtual Network Management Prediction, [AVNMP].

[58] We will see how the “on-demand” service approach will shift to *self-organization* in ad-hoc active mobile networks later in this work.

Projects	Characteristics								Node Kernels				Network Programming Systems			Programmable Network Architectures				
	arch. do-main	net. tech	prog. com-mun. abstr.	deg. of prog.	programming methodology				node IF & bind. mech.	res. mng. supp.	supp. for multi NW arch	sec. supp.	dyn. serv. comp. & ctrl	res. mng.	se-cu-ri-ty	app. level serv	NW mng	NW ctrl.	inf. trns	routg..
					comp. lang.	distr. obj. tech	mob. code tech	en-caps. tech												
ANTS (MIT)	cmps. NW serv.	Int.-net	IP set	dyn. ingr.	JAVA		X	X	X				X	x		x				x
Net Script (Columb. Univ.)	cmps. NW serv. & VNs	Int.-net	VAN	dyn. discr.	Net Script		X	X	X		x		X				x			
Switch-Ware (G.Tech)	cmps. NW serv	Int.-net	IP set	dyn.	PLAN & Caml		X	X	X				X		X					x
CANEs	cmps. serv.	Int.-net	cmps. serv.	dyn.	LIANE		X	X					X							
Smart Packets (BBN)	net. mng.	Int.-net	mng. Nodes	dyn., discr.	Sprok-ket & Spanner		X	X	X				x		X		x			
Smart Packets (U. Kan.)	cmps. serv.	Int.-net	IP set	dyn.	JAVA		X	X	X	x			x	x	X	x				x
ANN (U. Genv)	cmps. NW Serv	Int.-net	NW Node	dyn., discr	obj. code		X		X	x			x			x				x
Xbind (CU)	telco serv. create	ATM	multi-media NW	stat.	CORBA IDL	x			X				x			x		x		
AN NodeOS	NW prog.	Int.-net	NW Node				X	X	X		x									
DARWIN (CMU)	int. res. mng.	Int.-net	Flows	quasi static		x	X		X	x			x			x	x	x		
Mobi-ware (MIT)	QoS ctrl.	mo-bi-le	univ. mob. chan.	quasi static	CORBA IDL JAVA	x	X		X	x			x			x		x	x	
Tem-pest (U. Camb.)	Alt. Ctrl. Arch.	ATM	NW ctrl. Arch	quasi static	CORBA IDL	x	X		X	x	x		x			x		x		
XBONE	auto-depl. IP ovrly	Int.-net	IP Ovrly					X			x			x						x
Supranet	virt. NW serv.	Int.-net	virt. NW					x			x			x						x
Liquid SW (U. AZ)	mob. code tech	Int.-net			JAVA		X									x				
Active Services	app. layer serv.	Int.-net	app. serv.	Dyn.	Tcl		X		X				x	x		x				

Table 2: A comparison of the programmable network projects, [Camp99a].

This development let us conclude that Active Networking enters a new phase in the beginning of the 21st century which leads towards *actively adaptable self-organizing network architectures*.

3.6 ANALYSIS AND DISCUSSION

The primary goal of active networking is to accelerate network evolution by replacing the various ad hoc approaches to network-based computation with a generic capability that allows users to program and configure their networks at will. In programmable switches, separating the injection of programs from the processing of messages is particularly attractive when the selection of programs is made by network administrators rather than individual end users. In contrast, the capsule approach allows the embedding of user data within the capsules, thus providing user specific option for processing these data. In the following, we discuss the *usability* of the two basic approaches in active networking, the discrete and the integrated approach.

3.6.1 THE DISCRETE APPROACH

The processing of messages may be separated architecturally and conceptually from the injecting of programs into the node, with a separate mechanism for each function. This preserves the current distinction between in-band data transfer and out-of-band management channels well known in intelligent networks. Thus, users would first inject their custom processing routines into the desired programmable network nodes (routers, switches) and thus configure the network. Then they would send their packets through these nodes much in way as they do this with legacy nodes today. When a packet arrives at an active node its header is examined, and the appropriate program is dispatched to operate on its contents.

Separate mechanisms for loading and execution might be valuable when program loading must be carefully controlled. Allowing operators to load dynamically code into their routers would be useful for router enhancement, and thus lifetime network extension, even if the programs do not perform application- or user-specific computations. In the Internet for instance, loading code can be restricted to a router's operator who is equipped with a back door through which they can dynamically load code. This entry would at least authenticate the operator and might perform extensive checks on the code being loaded.

3.6.2 THE INTEGRATED APPROACH

A generalized view of active networks regards every message or capsule passed between the nodes of network as an *active packet*, i.e. a program (or at least an instruction) which may include embedded data. There is some guaranteed code present to each node. Some code is then being transferred between the nodes using a distribution protocol, such as ANEP. In addition, familiar application control mechanisms such as push and on demand loading, pre-fetching and caching are used as well.

When a capsule arrives at an active node, it is processed by a mechanism identifying its boundaries, possibly using the traditional link-layer framing mechanisms. Then, the capsule's contents are disassembled to a transient execution environment where they can safely be evaluated. The execution of a capsule results in the scheduling of zero or more capsules for transmission on the outgoing links.

The effect may be also a change of the non-transient state of the node. Programs run to a completion or self-suspension at a node. A node state can be one of the following:

- Soft state, cannot be relied upon
- Querying environment
- Capsule creation and manipulation
- Capsule control (e. g., forwarding)
- Capsule scheduling

Node states can be stored as data and communicated via active packets throughout the network, thus enabling states changes of other nodes along the route.

The main distinction characteristics between active packets and mobile agents are two:

- Mobile agents have data state (variables) and execution state (e.g. stack) whereas Active Packets do not have state included;
- Active Packets can be executed throughout almost all OSI layers whereas mobile agents are typically applications (layer 7).

Figure 28 illustrates a typical hybrid AN architecture supporting open programmable interfaces for value-added services, [P1520].

Despite the clear separation of concepts, layers and interfaces, active networks are considerably complex in their design which inevitably leads to failures at some point.

We strongly believe that deploying elaborate formal specification and verification techniques such as PVS used at NASA Langley [NASAL] will improve the test and accelerate the design processes of active networks.

Currently, the research community is divided concerning the usefulness of active networks. On the one hand, active networks provide a much more flexible network infrastructure, with increased capabilities to easily grow and introduce new services. On the other hand, they are obviously more complex than traditional networks and raise considerable security issues.

The argument against active networks is that the Internet is successful today because of its simplicity; by making the networks "active" things may get very complicated. The argument for active networks is that it is a very promising and innovative idea; a variety of useful network services that involve processing at inter-mediate nodes will be made possible and the use of such services is likely to lead to better end-to-end performance for applications [WLG98].

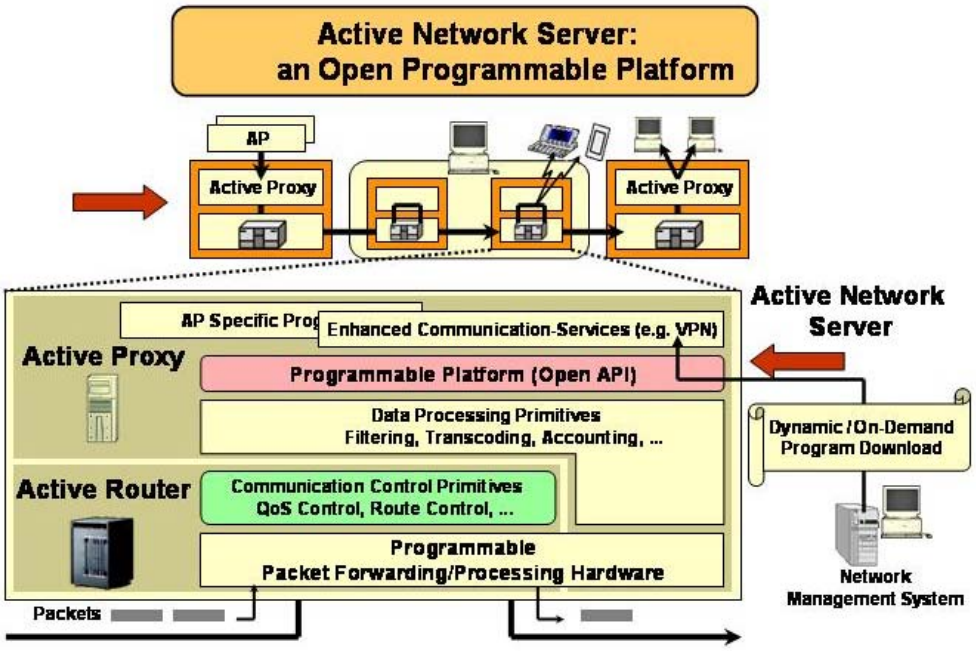


Figure 28: Active Network architecture realized on an open programming platform

3.6.3 Comparison of the Network Programming Approaches

3.6.3.1 Open Programmable Interfaces

The use of open programmable network interfaces is evident in many programmable network projects discussed in this survey. Open interfaces provide a foundation for service programming and the introduction of new network architectures. The `xbind` broadband kernel supports a comprehensive Binding Interface Base using CORBA/IDL to abstract network ATM devices, state and control. A number of other projects focused on programming IP networks (e.g., ANTS, Switchare, and CANEs) promote the use of open APIs, which abstract node primitives, enabling network programmability and the composition of new services.

Many network-programming environments shown on Table 2 take fundamentally different approaches to providing open interfaces for service composition. The programming methodology adopted (e.g., distributed object technology based on RPC, mobile code or hybrid approaches) has a significant impact on an architecture's level of programmability; that is, the granularity, time scales and complexity incurred when introducing new APIs and algorithms into the network.

Two counter proposals include the xbind and ANTS APIs. While the ANTS approach to the deployment of new APIs in extremely flexible presenting a highly dynamic programming methodology it represents a complex programming model in comparison to the simple RPC model. In contrast, the xbind binding interfaces and programming paradigm is based on a set of CORBA IDL and RPC mechanisms.

In comparison to capsule-based programmability, the xbind approach is rather static in nature and the programming model less complex. These approaches represent two extremes of network programmability.

One could argue that quasi-static APIs based on RPC is a limited and restrictive approach. A counter argument is that the process of introducing and managing APIs is less complex than the capsule-based programming paradigm, representing a more manageable mechanism for service composition and service control. Similarly one could argue that active message and capsule-based technologies are more 'open' because of the inherent flexibility of their network programming models given that capsules can graft new APIs onto routers at runtime. The xbind approach lacks this dynamic nature at the cost of a simplified programming environment. Other projects adopt hybrid approaches. For example the mobiware toolkit combines the static APIs with the dynamic introduction of Java service plug-ins when needed [BCK97]. A clear movement of the field is to open up the networks and present APIs for programming new architectures, services and protocols. As we discuss in the next section the field is arguing that the switches, routers and base stations should open up ultimately calling for open APIs everywhere.

3.6.3.2 Virtualization and Resource Partitioning

Many projects use virtualization techniques to support the programmability of different types of communication abstractions. The Tempest framework [Merw97a] presents a good example of the use of virtualization of the network infrastructure. Low-level physical switch interfaces are abstracted creating sets of interfaces to switch partitions called switchlets. Switchlets allow multiple control architectures to coexist and share the same physical switch resources (e.g., capacity, switching tables, name space, etc.). Typically, abstractions found in programmable networks are paired with safe resource partitioning strategies that enable multiple services, protocols and different programmable networking architectures to coexist. Virtualization of the network in this manner presents new levels of innovation in programmable networks that have not been considered before. All types of network components can be virtualized and made programmable from switches and links [Chan96] to switchlets [Merw97a], active nodes [NOSIS], routelets ([Camp99b], [Camp01]) and virtual networks ([Merw97b], [Camp99b]).

The NodeOS interface [NOSIS] provides a similar abstraction to node resources. The use of open interfaces allows multiple network programming environments (or execution environments using active networking terminology) to coexist within a common physical node architecture. In this case, the ANEP [Alex97] protocol provides encapsulation as a mechanism for delivering packets to distinct execution environments.

Using encapsulation in this manner allows for different overlay execution environments (e.g., ANTS, Switchware, or Netscript) to execute on the same router using a single, common node kernel. The notion of virtualization is not a new concept, however. Similar motivation in the Internet community has led to the advent of the Mbone. New directions in the virtualization of the Internet have prompted the proposal for X-bone [ToHo98], which will provide a network-programming environment capable of dynamically deploying overlay networks.

Other projects such as Supranet [DeFe97] advocate tunneling and encapsulation techniques for the separation and privacy among coexisting, collaborative environments.

3.6.3.3 Programmable Virtual Networking

The dynamic composition and deployment of new services can be extended to include the composition of complete network architectures as virtual networks. The Netscript project [YeSi96] supports the notion of Virtual Active Networks [NetScript] over IP networks. Virtual network engines interconnect sets of virtual nodes and virtual links to form virtual active networks. The Tempest framework [Merw97b] supports the notion of virtual networks using safe partitioning over ATM hardware. Tempest offers two levels of programmability. First, network control architectures can be introduced over long time scales through a 'heavyweight' deployment process. Second, 'lightweight' application-specific customization of established control architectures takes place over faster time scales. The abstraction of physical switch partitions within the Tempest framework has led to the implementation of multiple coexisting control architectures. The Tempest strategy aims to address QoS through connection-oriented ATM technology and investigates physical resource sharing techniques between alternative control architectures. Both Darwin [Chan98a] and Netscript [NetScript] projects support the notion of sharing the underlying physical infrastructure in a customized fashion as well. As discussed in the previous section, the NodeOS [NOSIS] project also provides facilities for coexisting execution environments.

3.7 OUTLOOK

The network programming abstraction provides a powerful platform for user-driven customization of the infrastructure, allowing new services to be deployed at a faster pace than can be sustained by vendor-driven consensus and standardization activities. The ability to download new services into the infrastructure will lead to a user-driven innovation process in which the availability of new services will be dependent on their acceptance in the marketplace. Active networks present an opportunity to change the structure of the networking industry from a mainframe mindset, in which hardware and software are bundled together, to a virtualized approach, in which hardware and software innovations are decoupled. However, as it will be shown later in this work, *configurable computing* can provide an additional flexibility of the programmable active network nodes, which can virtually couple again software with hardware on demand.

3.8 SUMMARY

The emergence of new technologies supporting encapsulation, transfer, safe and efficient execution, and interposition of programs and program fragments is one of the reasons why it is now possible to build active networks. At the same time, in the fields of operating systems and programming languages, issues relating to mobility, efficiency, and safety have been addressed. From all the above one can conclude that there is a user "pull" and a technology "push" towards a new way of thinking about the network ([Tenn96a], [Tenn97]): the user "pull" stems from the paradigms that "violate" the traditional properties of the network while the technology "push" stems from the fact that until recently it was not technologically possible to treat programs as a set of encapsulated and moving code fragments.

Conventional (passive) nets rely on agreements about protocols. The following problems with conventional protocol architectures led to the emergence of active networks:

- Long standardization process,
- Too high protocol stacks
- Poor performance due to redundant operations at several protocol layers,
- Emerging need of computations within the network,
- Difficult integration of new technologies(e.g. IPv6),
- Difficult accommodation of new services and applications in the network.

The active packets approach suffers from performance-related problems because safety and security requirements are huge. In an effort to reduce the security burden and thus increase performance, some researchers have decided to restrict the functionality of the programs carried by the active packets, resulting in architectures with decreased capabilities. M0 is the only architecture within the active packet approach that can provide arbitrary functionality, thanks to its novel caching technique.

The active nodes approach has good performance because security issues are much less than in the previous approach. However, the flexibility of the relevant architectures is limited. In an effort to increase flexibility, DAN and ANTS architectures have adopted a scheme where code is downloaded on demand and is cached for future use. As a result, these two technologies can easily deploy any new arbitrary protocol. Nevertheless, downloading code on demand causes some delay that reduces the overall performance.

The combination of both approaches seems to be very appealing. SwitchWare architecture realizes this idea by the use of a layered architecture, and manages to provide a range of different flexibility, safety and security, performance, and usability tradeoffs. Finally, NetScript architecture proposes a novel viewpoint where the network is treated as a single programmable abstraction. *Activity* of nodes and packets offers a considerable flexibility in networking allowing new functionality, better performance and faster deployment of protocols and services, yet, at the cost of diffuse separation between the OSI layers, unclear security and performance decrease when flexibility is not needed.

Active networks are consistent with the end-to-end argument of service provisioning:

- they provide *a generic interface*, available to all users; the cost of the interface is one-time cost (infrastructure).
- they allow users to select network services more precisely; the cost of providing a service is paid only by those applications using it.

Active networks were expected to replace traditional IP networks, [Gutt97].

The following few paragraphs summarize the essential characteristics of the AN approach:

- *Active*[59] *Packets* (*APs* or *capsules*) replace the packets of traditional IP networks; they carry data and code responsible for the processing of this data at specialized *(active)* network nodes. A series of APs sharing common information within the network are grouped into a *protocol.* The protocol provides a network service; it is the basic element for network customization and protection.

- *Active network nodes (ANN)* replace selected nodes within the IP network and its terminals (routers, switches, hubs, bridges); they execute the active packets of a protocol and maintain the protocol state. Unlike ordinary nodes, active nodes provide an API for capsule processing routines, and execute those routines safely by using operating system and programming language techniques.

- A *code distribution mechanism*[60] ensures that AP/capsule processing routines are transferred automatically and dynamically to the active nodes where they are required.

The above model supports the programming, the introduction and the deployment of new network services by a generalized form of packet forwarding based on the following premises for *safety and security* [WGT98]:

- Active forwarding routines, like traditional ones, are expected to run to completion locally and within a short time. Because of the risk for malicious usage, they are intentionally limited[61] in their capabilities such as global memory and bandwidth consumption, which are bounded by a lifetime or Time-To-Live (TTL) period.
- The forwarding routine of an AP/capsule is set at the sender and may not change as it traverses the network; nor may capsules belonging to one protocol create capsules or access state belonging to a different protocol within the network. Thus, transmission channels are isolated and one user may not control the processing of another user's capsules in unintended ways.

[59] In the classical definition, an AP/capsule is also "*passive"* (i.e. not autonomous) during the end–to-end transport over the network; its *activation* proceeds at the destination node or at predefined intermediate nodes. The WLI approach, however, reveals some additional capabilities of the active packets.

[60] This component does not exist in traditional IP networks, and is handled by the AN system, not by the service provider.

[61] A possible solution to this problem could be e.g. the periodical transmission of an "authentication" capsule applying an already negotiated coding scheme in previous transmissions.

- Depending upon the node's available resources and security policies, some active nodes may not execute particular forwarding routines, but rather perform “default” IP-like forwarding. In addition, some forwarding routines may self-select nodes at which it is useful to perform their specialized processing depending on the location of the node and its capabilities.

According to the AN model, a new network service is defined as a protocol structure in terms of the different AP/capsule types or *types* (data) and their processing methods or *routines* (code). An application can immediately use the specified service by supplying the protocol definition to the local node, starting sending, and receiving the appropriate AP/capsules.

3.9 CONCLUSIONS

3.9.1 THE AN APPROACH

Active networks are a “*quantum jump”* in the evolution of packet-switched networks from simple packet forwarding engines to elaborate processing and communication environments. By providing programmable interfaces in network nodes, they expose to their resources, mechanisms and policies and allow the dynamic modification of the network behavior on a per-node, per-packet and per-user base.

Active networking has a wide application range and the potential for solving many of the problems identified in current “passive” networks. There are various applications where active networks can be beneficial. Network management, congestion control, reliable and efficient multicasting, and active caching are some of them. However, current research in active network architectures and their applications assist solely the examination of their usefulness, applicability, and efficiency. Yet, no existing active system is flexible enough to anticipate and accommodate the future needs of the network. The scope of future updates in present active networks is too restricted.

Most systems offer *plug-in extensibility* abstracting future changes through a pre-defined interface. Allowing network nodes to be extended by dynamically loading code which addresses only pre-defined future changes is not sufficient enough to enable true network evolution (ANN: [ASNS97], [YeSi96]; AP: [WGT98],[Hicks98]).

For instance, because capsules are the only plug-in, much of the ANTS system [WGT98] is not subject to change. This includes the code distribution service, the entire node-programming interface, the packet arrangement and disarrangement code, the packet code cache, the security enforcement policies, etc. If some aspect of the node API, or the distribution protocol, needs to be changed, then the entire node architecture have to be changed, recompiled and redeployed. In general, this reasoning applies to other AN systems such as *SmartPackets* [SmartP99], and *PLANet* [PLANet99].

Recently, Hicks [Hicks99] proposed a new technique, the *dynamic software updating*, which allows to add new functionality to a predefined structure and change *any functional part* of the system or its type *at anytime* (incl. runtime) without anticipation by the programmer. Such developments appear very promising.

However, the implementations that have been built at the various research sites have not been tested in large-scale networks so far. It is not very clear how the available experimental results would generalize to a large-scale network like the Internet. Before deriving any provable conclusions, it is necessary to deploy and test the performance of the diverse active network technologies under real conditions in ABone[62], analogous to the Mbone for multicasting.

The problem with large-scale experiments is that they also take too much time and require a lot of coordination work. Besides, testing any new incremental idea requires almost the same and often even more and more time and effort. For instance, what about hardware upgrades in active networking ? The network is changing all the time. We think that ideas have to underlay some "mental" tests in a scientific manner before being offered as RFCs. Formal methods are providing a good scientific base for engineering research. In this work, we impose an additional requirement for evolutionary active networks, namely the one of *dynamic hardware upgrade and reconfiguration* of active nodes. We claim that the Wandering Logic Intelligence (WLI) is a suitable formal technique for specifying and verifying such systems.

There are two basic tradeoffs in active networking:

1) between security levels and performance, and
2) between usability/flexibility and complexity.

A lot more can be done to address the security and programmability issues in active networking. However, it is very difficult to say now where the security-performance tradeoff can be optimized. "Conservative" approaches toward active networks may yield satisfactory results, particularly in combination with the emerging mobile software agent technology. As for the usability/flexibility issue, a compromise could be reached. Nevertheless, the complexity has to be manageable. The optimal line of the flexibility–complexity tradeoff is also hard to draw. ABone may be used to find optimum solutions for both tradeoffs. However, a try-and-error approach usually takes too much time, especially in a large-scale environment. This is where formal models can help to accelerate the experimental research and find the real boundaries of applying AN technologies.

The AN approach does not simplify service programming *per se*, as it requires heuristic treatment of well-known problems such as packet and protocol state loss, changing routes and concurrency. Therefore, users are not expected to program their services, but rather to construct and configure them following the Intelligent Network SIBB model through selection between protocols offered by service providers or third party software vendors. Furthermore, it is not the task of protocol designers to consider how to *distribute* the processing routines throughout the network, nor worry about *interaction* with other protocols (except in case of resource reservation at the active nodes). Active networking pursues the just-in-time (JIT) introduction of new network services and applications.

[62] a DARPA-sponsored active network in which active nodes communicate via tunneling through the Internet using UDP [Abone].

Therefore, the business model assumes that network operators are supposed to be interested in generating traffic in their infrastructure, and thus open their architectures for use, despite the enormous security and safety overheads.

Finally, there are several tradeoffs to be made when introducing new Internet services with the active networking paradigm, [WLG98]:

- *Protection* is based on the inability to specify processing for another user's packets and the encapsulation of protocol state by the associated capsule processing routines.
- *Allocation* is based on the limited resources that will be granted to each packet by the network nodes.
- *Performance* is based on the simple event-driven processing model and the ability to tailor processing to the diversity of heterogeneous networks, including those in which only some nodes may be active.

3.9.2 THE FORMAL APPROACH

Active networks represent complex structures and mechanisms imposed by the hard (but necessary) technology challenges described earlier in this chapter (Motivation). In addition, evolutionary models and requirements for dynamically adaptive software increase the difficulty in designing such architectures because of the inclusion of new technologies. Therefore, the application of methods and tools capable to specify and validate the desirable semantics of such architectures in the early design stages, before implementation, are highly desirable and recommendable.

Formal techniques, such as process algebras, graphical notations, state machines, set theory and models in logic proved to be the appropriate means for this task ([HeMa96], [Harry96]). In particular, programming languages such as LOTOS, Estelle and SDL were specifically developed for telecommunications [Turn93].

In fact, the gap between AN models, specifications and the actual code at system level could be quite big. Therefore, formal techniques are expected to bridge (or essentially reduce) this gap by providing adequate models and a good mathematical reasoning basis for system assurance using logic languages assisted by mechanical specification and verification tools such as computer algebra systems, graphical design browsers, proof checkers, and automated and interactive theorem provers. The goal of these formal approaches is to prove that the desired static and dynamic properties of the system (code and architecture) are correct and safe.

Basically, a configurable network is characterized by both node and packet *programmability* and *adaptability*. In addition, active packets transport *mobile code*. In this work, we postulate that *active nodes can be also mobile*[63], for instance as mobile platforms such as vehicles or wearable user terminals.

[63] In fact, assuming that routers can be mobile is a true challenge for routing protocols even at the presence Mobile IP and IPv6. Yet, this is inevitable step ahead in the entering the Cosmos. Perhaps the recent NASA initiative of Vint Cerf, the *InterPlanet Protocol*, is already addressing this issue.

The formal approach requires that all these properties of the active system be described in a formal way, such as a calculus or logic. Furthermore, the formal language should support generic design features such as openness, distribution and object-orientation. Active networking requires that programs be transmitted across the communication substrate and loaded into a range of platforms. This implies the development of formal specification techniques and common models for the encoding of network programs, the built-in primitives available at each node, and the description and allocation of node resources.

However, irrespective of anyone's concerns about the future, active networks enter its second age. Currently, there are two driving forces in AN research:

- the possibility for automatic upgrade of network protocols, and
- the possibility to program and deploy the intermediate node functions through simple, open, and rapid interfaces and processes without the need of standardization committees or vendors' lobbies.

Given these two opportunities, it is well worth trying. In any case, perhaps the problem is not whether networks should be programmable or not, but deciding on paradigms that will program them efficiently.

In this chapter, we have discussed the state-of-the-art in programmable and active networks. We have presented the conceptual paradigms, the reference model, the basic architectures and the mainstream directions in research.

The generalized architecture of a programmable network comprises communication and computation models into the architecture and opens interfaces to the underlying hardware.

Active Networks devise a paradigm shift in network design towards higher levels of network programmability following the principles of:

- *separation* of hardware from software;
- *management* of hardware as software;
- *availability* of open programmable interfaces to all levels of network granularity;
- *virtualization* of the networking infrastructure;
- *tools for rapid creation* and *deployment* of new network services; and
- *safe resource partitioning and coexistence* of distinct network architectures over the changing physical networking hardware.

Active programmable networks provide a foundation for architecting, composing and deploying virtual network architectures through the availability of open programmable interfaces, resource partitioning and the virtualization of the networking infrastructure. The key challenge in this research is the synchronous development of programmable virtual networking environments based on configurable hardware architecture.

* * *

CHAPTER 4: RECONFIGURABLE COMPUTING

"*Reconfiguration* after failures reflected STAR [project] concepts...
One advantage of distributed systems is that interfaces can be
simpler than in a system using a central computer.
Each local computer is responsible for its own timing and control."

GALILEO – TRUE DISTRIBUTED COMPUTING IN SPACE,
DISTRIBUTED COMPUTING ON BOARD VOYAGER AND GALILEO
http://www.hq.nasa.gov/office/pao/History/computers/Ch6-3.html

"Over the weekend, the *reconfiguration* of the spacecraft following the
completion of the Attitude and Articulation Control Subsystem
in-flight mode was completed on Saturday."

February 1, 1993
GALILEO STATUS REPORT
http://seds.lpl.arizona.edu/ftp/spacecraft/GALILEO/gs02.01.93

4.1 OVERVIEW

Reconfigurable computing is a relatively new field of research, which emerged in the late 1980's to fill the gap between hardware and software by achieving much more performance than software while maintaining higher level of flexibility than hardware. It is an aggregation of "hard-wired" computation, one that performs series of routine computations, e.g. signal processing, using *Application Specific Integrated Circuits (ASICs)*, and the *flexible*, or "soft-wired", i.e. the programmable one, which performs computations using general-purpose *(micro)processors* capable to execute series of instructions provided entirely in software.

The most common devices used for reconfigurable computing are *Field Programmable Gate Arrays (FPGAs)*. FPGAs consist of a matrix of logic blocks and an interconnection fabric or "mash" (Figure 32). Both, the functionality of the logic blocks and the connections between them can be modified, i.e. configured or programmed by downloading bits of configuration data into the hardware. FPGAs allow designers to manipulate gate-level devices such as flip-flops, memory and other logic gates. However, FPGAs have also some inherent disadvantages such as bit level operation and inefficient performance for logic operators and ordinary arithmetic. Therefore, many researchers have focused on a more general and higher level models of configurable computing systems such as *PADDI* [ChRa92], *rDPA* [HaKr95], *DPGA* [Tau95], *MATRIX* [[MiDH96], *RaPiD* ([Ebel97], [ECF97]), *Raw* ([Wain97a], [Wain97b]) and *Garp* ([HaWa97],[CHW00]).

Reconfigurable computing is different from the von-Neumann paradigm of computing and requires computational models different from conventional models. The ability to implement an application in hardware provides an opportunity to exploit the inherent concurrency of digital circuits. That is, the device can be configured, or partitioned, into multiple subsystems - all of which could run concurrently with each other.

One major goal of reconfigurable computing systems is to exploit the fine grain and the coarse grain parallelism in applications, which allows a functional split and adaptation of the hardware to specific computations in each application to achieve higher performance than software. Applications are mapped onto reconfigurable architectures by analyzing the computations performed. The partitioning of an application's computations between the microprocessor and the reconfigurable hardware is performed manually or by means of (semi-)automatic tools. The obtained results are compiled into executable code on the microprocessor and hardware configurations on the reconfigurable hardware.

Before the computation can be executed, the reconfigurable hardware needs to be configured using the configuration information. Thus, configurations can be updated at run-time to execute a different set of operations from the application.

Currently, hybrid architectures, which integrate programmable logic and interconnect along with a microprocessor on the same chip, the so-called "systems on a chip" (SoC), are being developed. On-chip integration of reconfigurable logic reduces the memory access costs and the reconfiguration costs. SoC systems are used in almost any network device.

4.2 SCOPE

With the dawn of multimedia and wireless communications, reconfigurable computing is often associated with the ability to modify a computer's system hardware architecture during an application's runtime, and in particular – in real time. Current computers are fixed hardware systems based upon microprocessors. With each new generation of microprocessors, the application's performance increases only incrementally. Traditional fixed hardware may be classified into three categories: *logic* (Gate Arrays, PALs, etc.), *embedded control* (controllers, e.g. ASICs and custom VLSI devices) and computers (microprocessors).

Reconfigurable computing systems are those computing platforms whose architecture can be modified by the software to suit the application at hand. To obtain maximum performance or throughput an algorithm must be molded into a hardware (ASIC, DSP. etc.).

Thus, dramatic improvements in the performance of the system are achieved through the "hardwiring" of the algorithm. In a reconfigurable system, these hardwiring takes place in an "interpreter"-like manner on a function-by-function basis as the application evolves.

Recently, advances have been made in the integration of *compilers* within reconfigurable architectures such as Garp, [CHW00], and PipeRench[64]. In this way, digital circuits can be programmed and swapped into a reconfigurable computing system *on demand* by a software application during execution time. Reconfigurable systems take advantage of *spatial* parallelism while reducing *temporal* overhead of load store, branch operations and instruction decoding.

Since this work is concerned with the design of new generations networks, the subject or reconfigurable (or programmable hardware) is reviewed from two perspectives: a) *micro-architectures*, related to the VLSI (gate) level up to a PC board or a handheld mobile terminal level, and b) *macro-architectures*, addressing configurable servers, switches, routers, etc. What follows is a short review of applications and computing models related to the "micro" level. The "macro" perspective was already discussed in the previous chapter (Active Networks).

4.2.1 APPLICATIONS

Over the last decade, *reconfigurable computing machines (RCM)* have demonstrated significant potential for a range of applications. Many of these tasks (e.g. real-time signal processing) are computation-intensive and have high throughput requirements. Other applications are inherently complex (e.g. real-time speech recognition). In general, conventional microprocessor-based architectures fail to meet the performance needs for most of the *locally centralized*[65] applications in the realms of long multiplication [Vuill96], genetic algorithms ([GrNe96], [SMP99]), image processing [AtAb95], signal processing [TeBu00], cryptography [Dand00], genomic database search [LeMe95], etc. Automatic target recognition, feature extraction, surveillance, video compression are among those applications that have shown performance improvements of over an order of magnitude when implemented on configurable systems [Man97]. Other applications are parallel computations, stream processing, template matching, image filtering, etc.

Co-processors	**Applications**
Intelligent embedded controllers (e.g. laser controller)	High-bandwidth graphics and communication processing (e.g. wireless multimedia)
I/O channel processors (e.g. protocol converter, database accelerator)	Bit-slice (and bit-tweed) functions
Image processing; enhanced video/ multimedia (e.g. image processor, transcoder)	Complex algorithms processing streams of integers
DSP function acceleration	Image/audio/video processing

Table 3: Configurable computing machines and their usage

[64] http://www.ece.cmu.edu/research/piperench/.

[65] i.e. performed on a single processor or a cluster of processors on a desk-top PC, a workstation or an embedded system.

Configurable computing systems have been considered to be suitable solutions for dedicated signal and software processing, Table 3. As we will see later in this chapter, they may also provide the technology for the next generation *dynamic networking*.

4.2.2 COMPUTING MODELS

We distinguish between two computing models, *fixed* (Figure 29) and *flexible* (Figure 30), historically w. r. t. hardware, and thus defining the corresponding market segments, but basically reflecting the three layers in development both in software and hardware, and hence defining the application areas: computers, embedded control and logic. In this chapter, we refer to *applications* as to specifically hardware applications.

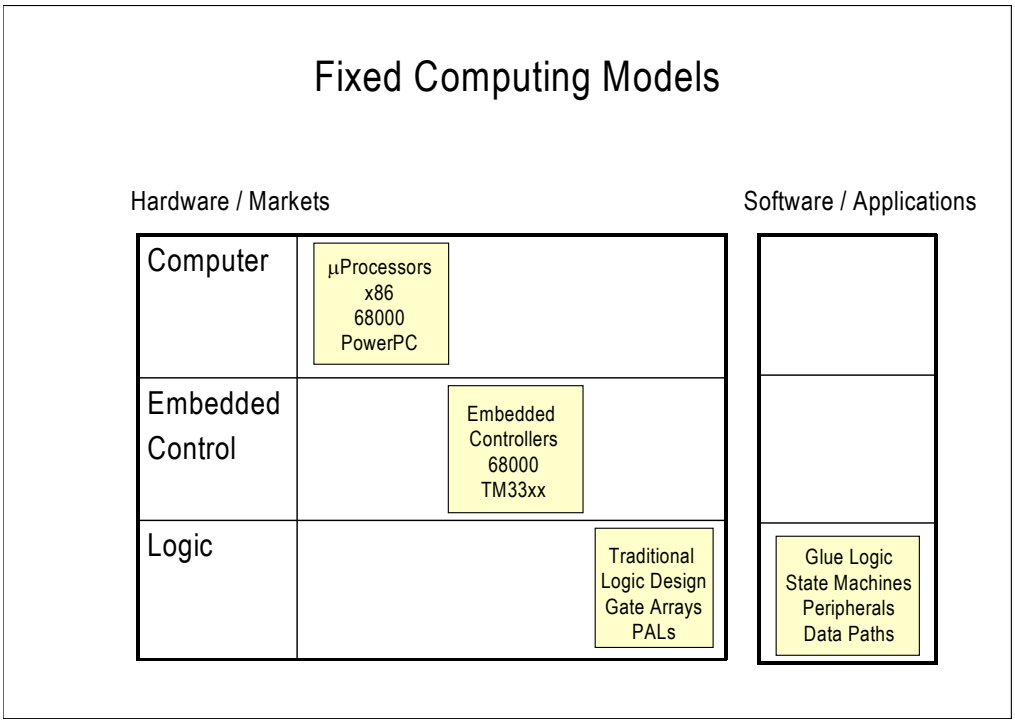


Figure 29: Classification of computing developments within *fixed* models

Further, within the flexible computing model in Figure 30 we classify reconfigurable hardware into three categories:

- Logic: FPGA
- Embedded Control: Reconfigurable co-processors, tightly coupled FPGAs to traditional systems
- Computers: completely reconfigurable computing platforms using FPGAs in a system designed for general purpose computing; high-end reconfigurable computers address the switches, servers and routers of the flexible hardware macro architectures.

In addition, we expand this model to a fourth level or category, the *networking* dimension, by addressing configurability of sets of network nodes, which are composed of configurable elements of the other three layers. This model is going to provide the "hardware base" of the *wandering logic intelligence (WLI)* referred later in this work. The WLI's software base, which comprises the basic principles of active networking, has been discussed in the previous sections.

The leading idea behind WLI is that we wish to devise a set of guiding design principles unifying the numerous ad hoc approaches in network evolution in a thorough logical framework providing *the best available flexibility* in software and hardware technology at a certain level of development. We wish to give a possibly direct answer to the question of how *to make a network* like the future Internet and how to let it develop in order to provide the desired performance, quality, security, etc. features of importance to all users.

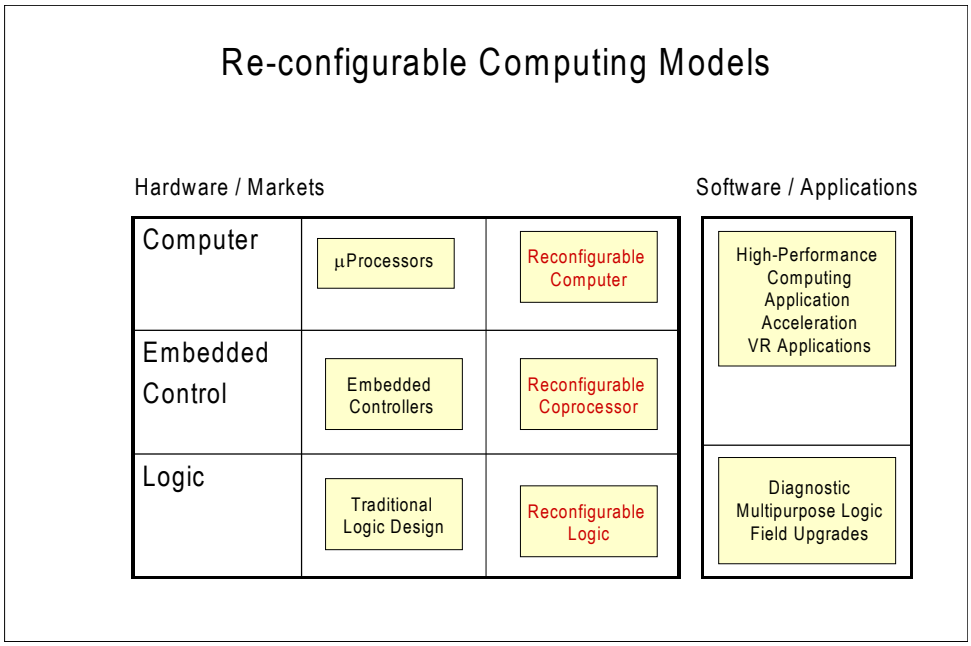


Figure 30: Classification of computing developments within reconfigurable models

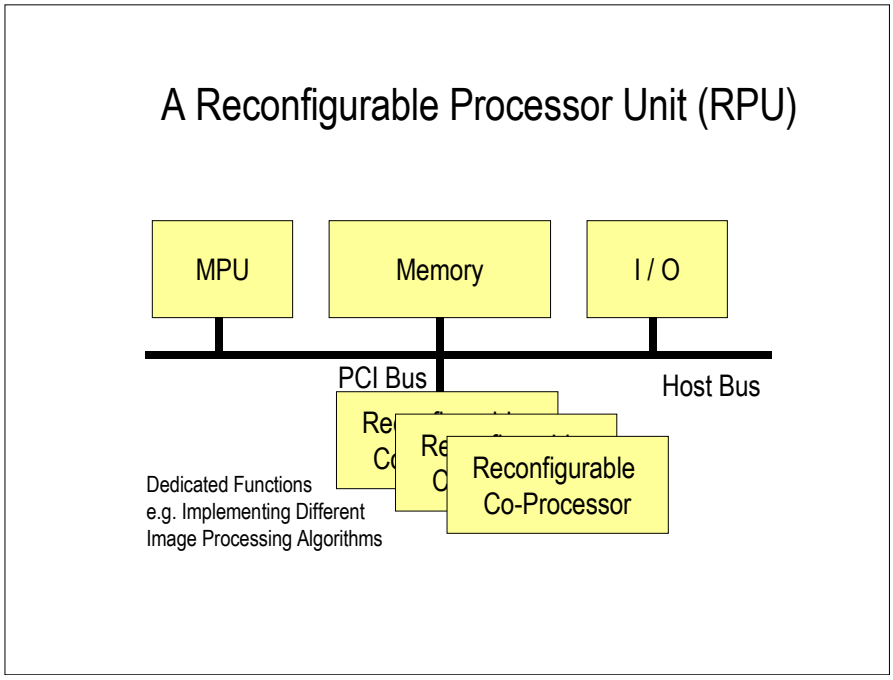


Figure 31: Typical architecture of a reconfigurable multiple co-processor unit

Networking differs from local, centralized processing in that load and performance, in terms of computational intensity and application specifics vary in both space and time. Increasing resources and their capacity and performance is, of course, the long-term goal in network development. Yet, available resources are often *over-dimensioned* in the very beginning for usage at some peak rate or number, and later, when this rate or number is reached, they are *not re-usable* with advancing technology (in terms of physical limits). In fact, networks are expensive, because of not being exploited in an effective way *over time*.

However, there is another option. *To move.* Moving people across the network to guarantee QoS and effectively use the resources is outdated. What remains is *to move code and data, and to re-configure resources at runtime, i.e. to mobilize them (*in some way) to be close to the user.

Figure 31 depicts the starting point of our investigations in terms of hardware, the familiar example of a reconfigurable processor unit (RPU). What we focus on is the expansion of this model to networking *over time*, both in terms of software and hardware: the adaptive, evolutionary networking.

4.3 MICRO-ARCHITECTURES

4.3.1 FIELD PROGRAMMABLE GATE ARRAYS (FPGA)

With the introduction of the *Field Programmable Gate Array* (*FPGA*, configurable logic chips) in the early 80's, hardware engineers were empowered to implement chip level designs without having to fabricate a chip. As these devices and their software tools matured, the use of FPGAs expanded from testing and verification of digital systems to in-system design.

FPGAs perform the function of a custom LSI circuit (e.g. a gate array, ASIC) and are user-programmable. They were originally designed as alternatives to mask configured gate arrays, the bit processing elements implementing the logic gates, and the programmable interconnect replacing selective gate wiring [Trim92].

Currently, they are used as glue logic for design of adaptable systems and coprocessor devices. FPGAs are also used to emulate other component architectures, and are applicable for rapid prototyping. Increasingly, FPGAs have been applied as *spatial computing* devices for recursive and parallel processing (Figure 33). They have proved themselves a feasible alternative to the *temporal computing* model (ALU) as being the fastest or most economical way to solve highly parallel and recursive problems such as signal and image processing, DNA sequence matching and cryptographic search. SRAM based configurations can be reprogrammed on the fly by downloading different configuration bits into the SRAM memory cells. The future SRAM based FPGAs will invoke a completely new generation of computing applications.

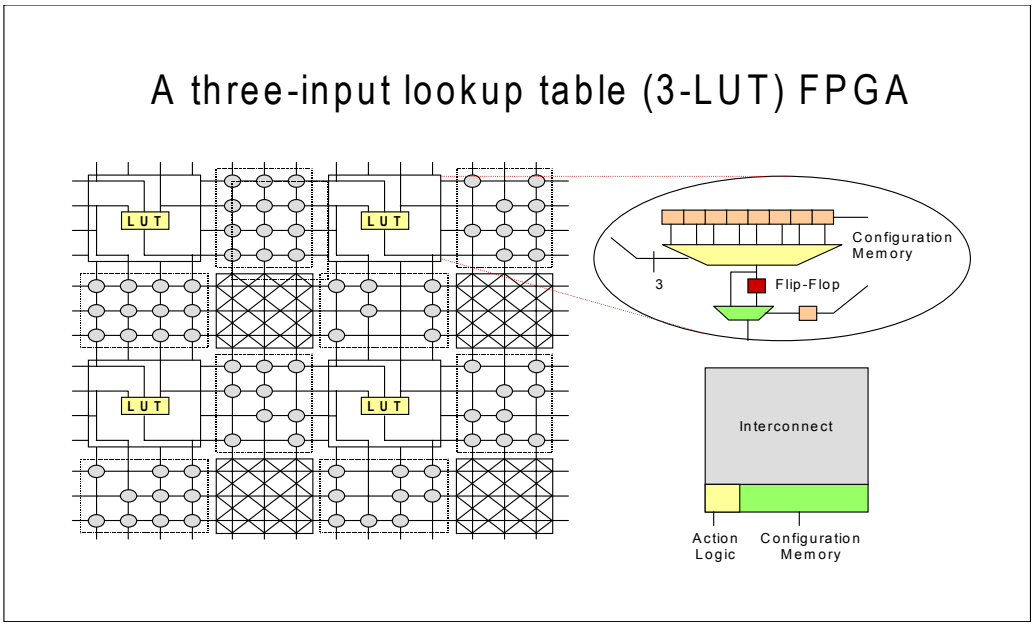


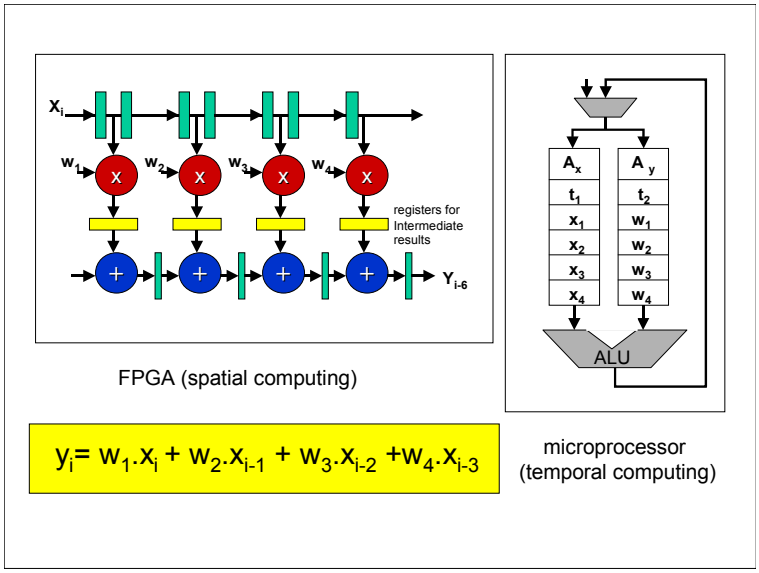
Figure 32: A three-input lookup table (3 LUT) FPGA

An FPGA is an array of bit processing units whose function and interconnection can be programmed after fabrication. Most traditional FPGAs use small lookup tables to serve as programmable computational elements. As shown on Figure 32, the look up tables are wired together with a programmable interconnect, which accounts for most of the area in each FPGA cell. Commercial devices usually have 4-input lookup tables (4-LUTs) for the programmable processing elements because they are area efficient [Rose90].

4.3.2 RECONFIGURABLE COMPUTING

Computing with FPGAs is called *configurable computing* because the computation is defined by configuration bits in the device that controls each gate and interconnect how to behave. Like processors, FPGAs are programmed after fabrication to solve virtually *any* computational task that fits in the device's finite state and operational resources. This impermanent, post-fabrication customizability distinguishes processors and FPGAs from custom functional blocks (e.g. ASICs) which are operationally set during fabrication and implement only one function or a very small range of functions.

Unlike processors, the primitive computing and interconnect elements in an FPGA hold *only a single device-wide instruction*[66]. Without undergoing a lengthy reconfiguration, FPGA resources can be reused only to perform the same operation from cycle to cycle. Thus, tasks are implemented by *spatial composition* of primitive operators, i.e. by linking them with wires. In contrast, traditional processors temporally compose operations by sequencing them in time, using registers or memory to store the intermediate results, Figure 33.



$y_i = w_1.x_i + w_2.x_{i-1} + w_3.x_{i-2} + w_4.x_{i-3}$

Figure 33: Example: spatial vs. temporal computing

[66] Here, the term "instruction" refers to the set of bits controlling one operational cycle of the FPGA.

The single-instruction-per-active-computing-unit limitation in FPGAs provides an area advantage, at the cost of restricting the size of computation described at the die at any point in time. Thus, a single reconfigurable device often can compute, in a single cycle, a desired computation such as a filter operation that takes a processor or a DSP hundreds of cycles and MHz power to evaluate even simple filter taps. The FPGA might require tens of cycles latency to compute a result, but because it performs the computation as a spatial pipeline composed of many *active* computing elements, rather than sequentially reusing a small number of computing elements, it achieves a higher throughput.

An FPGA has a better performance for two key reasons, both due to the spatial organization:

1. *FPGA exploit more parallelism per cycle;* with less instruction overhead, the FPGA performs more active computations onto the same silicon die area as the processor.
2. *FPGAs can control operations at the bit level*, but processors can control their operations at the word level. As a result, processors often waste a portion of their computational capacity when operating on narrow-width data.

Thus, through pipelining, a FPGAs extracts more computational capacity from a silicon die than, e.g. a RISC processor. When operating on short data items, FPGAs have a potential for a second order-of magnitude advantage in computational density over processors [DeHon00]. However, the problem with FPGA is that it is difficult to pipeline a particular design in an adequate fashion to achieve such a high clock rate. Conventional FPGA architectures and engineering methodology make it difficult to contain interconnect delays and reliably target clock rates near the device's peak capacity. This is in turn an opportunity for formal models and simulation techniques to help finding the most appropriate split between software and hardware designs.

4.4 MACRO-ARCHITECTURES

4.4.1 THE ROAD AHEAD: AN ADAPTABLE NETWORK

The experiences collected with recent developments in *run-time reconfigurable* (RTR) hardware and hardware/ software co-design techniques enable the design of dynamically reconfigurable high-performance switches, which are expected to improve both the performance and functionality of future network routers. The combination of speed and flexibility offered by these devices has been the key to much recent work in networking hardware ([HaSk96], [McH97], [Hess99]). With suitable computation and interface modules, *stream processing* has the potential to enable hardware-level configuration and packet processing at line speeds.

The achieved results provide the basis for a *hardware-reconfigurable network*. A prototype of a reconfigurable router was proposed in [LHAM99]. In [Hess99] the authors present its implementation using RTR FPGAs. The following section shortly describes this architecture, which is used later as a reference model.

4.4.2 THE RECONFIGURABLE ROUTER ARCHITECTURE

Figure 34 shows the basic architecture of a typical router. It consists of a set of input and output ports that are connected via an interconnection architecture, which is controlled by a routing engine running on top of the router operating system. The interconnection architecture can deliver either packets or pointers to packets to assist in transport from port to port.

To improve performance, each port may have one or more processors. The routing engine is used to process routing protocols and to perform a route lookup. The result of the lookup function is a forwarding table, which is used by a forwarding engine to determine where the packets should go. Forwarding engines can be located within the input/output port devices or as separate port devices.

The activities towards designing a flexible router operating system can be categorized into two areas: open signaling and active networking. In open signaling, a set of application programming interfaces (APIs) is defined so users can access, in a standard way, information stored within a router. In active networking, as we have seen in the previous sections, new protocols and services can be *injected* at run-time. An active network has the advantage that it not only provides a uniform (software) signaling interface, but also a platform-independent mobile code. Thus, programs written for the vendor A's router hardware can, by itself, migrate and run on vendor B's router hardware. In fact, we consider open signaling as a part of the larger goal of an active programmable network, [Camp99a].

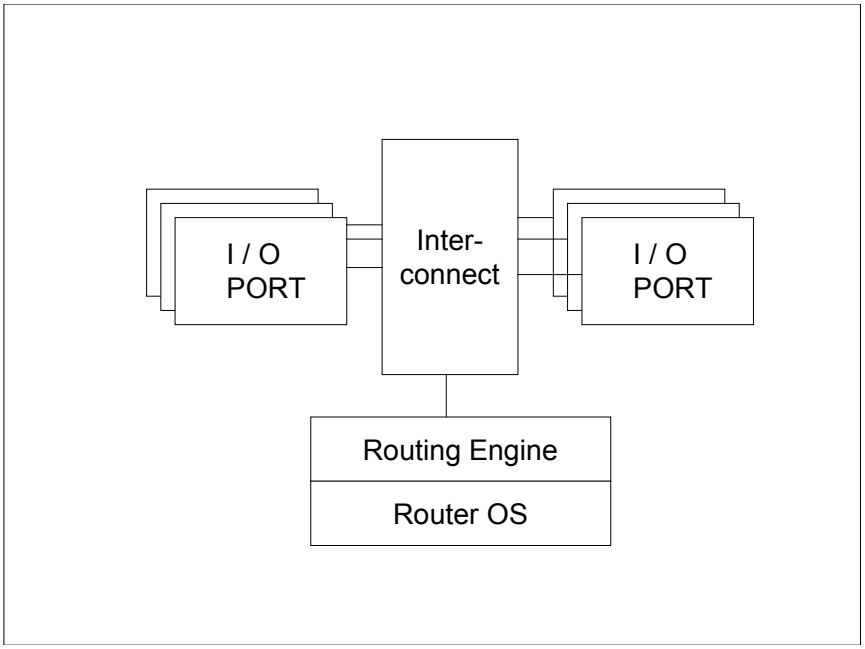


Figure 34: Basic router architecture

4.4.2.1 The Out-of-Band Reconfigurable Router

Figure 35 shows the general model of the reconfigurable *out-of-band* router, i.e. a router that is separately programmed from the packet flow passing through it (the OPENSIG approach, [OPENS]).

The router operating system provides the resource management and route protocol processing, and it would have the capability to receive, authenticate, and install any new protocols as required. Once a new protocol code is loaded into the operating system, it is partitioned and scheduled according its functionality for execution. The latter can be performed on a general-purpose processor for the application-specific code and on reconfigurable hardware (e.g. an FPGA) for the computing-intensive code. For example, the routing engine of a typical IP switch may receive up to 50 route updates per second, [IBM96]. A general-purpose CPU processes this information. Furthermore, there may be hardware assistance to perform the actual route lookup using custom VLSI implementations that are based on *content addressable memories (CAMs)* or some hardware processing of a trie structure [[PeZu92]. A VLSI implementation typically only supports IP route lookups. Software route lookups are completely performed using *trie* structures ([WVTP97], [DBCP97]) – the software can be modified to support other protocol types.

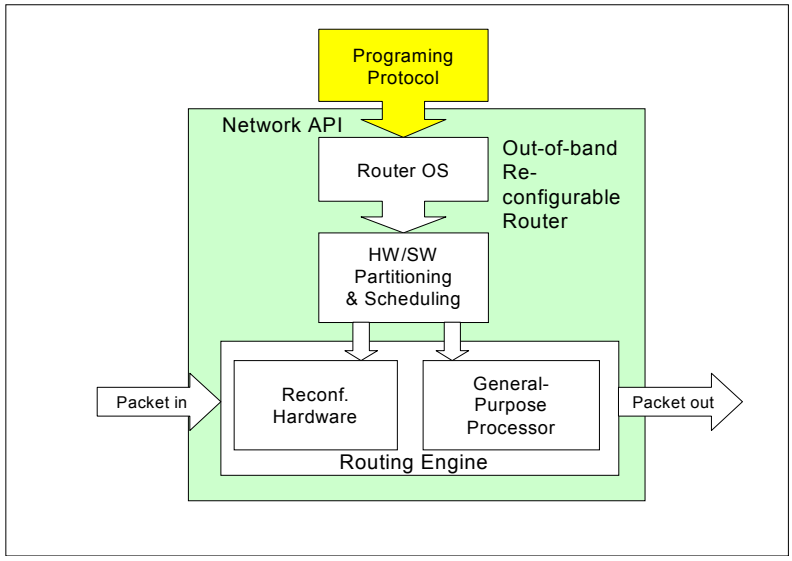


Figure 35: General model of an out-of-band reconfigurable router

Thus, a *routing engine* may consist of:

- a *dedicated hardware to perform route lookups*,
- a general-purpose CPU to process the routing protocols and provide network control,
- *special-purpose hardware to support the packet processing* ([KVE95], [DCP95]).

The hardware used by the routing engine on Figure 35 only processes the headers of IP packets. The reconfigurable approach of [Hess99] allows processing of *both* headers and data.

4.4.2.2 The In-Band Reconfigurable Router

The in-band, or *strea*m-based router (Programmable Active Network approach, [Camp99a]) is reconfigured by programs sent along with the data to be routed through the same network interfaces. Processing streams is performed in a somewhat similar manner as processing capsules in active networks; the difference is in the internal architecture of the router (active node) which *reflects* the arrangement of the stream packets, the flow of capsules or both of them (in the hybrid version), Figure 36.

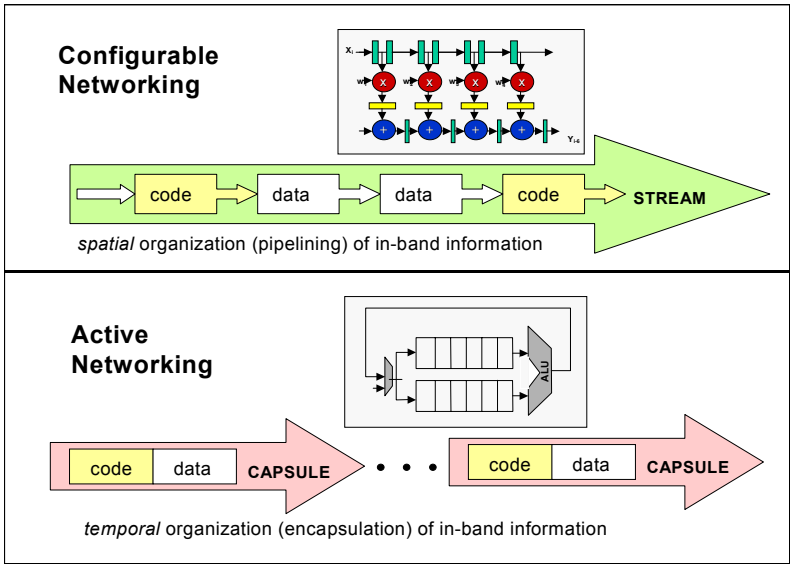


Figure 36: Spatial vs. temporal organization of the in-band information

The stream programming information configures a RCM for the computation task on the *user data that follows*, whereas active network capsules contain code (or a reference to it) which is destined to process the data in the capsule itself (and some of the capsules to follow). Therefore, generally stream programming is associated with *spatial* computing, whereas capsule programming – with *temporal* computing in the network node.

Stream-based computing is based on the use of self-guiding streams of programming information and user data to perform a computational task [BiAt97]. Stream-based approaches have been used in other configurable hardware applications in the realm of signal processing [SwAt99]. Within a reconfigurable router, the stream programming header can be viewed as a protocol, and the data as the packets. **The advantage of the stream-based paradigm is that the information stream can be viewed as a network flow with switching and other features defined by the programming header (!).** In addition, the stream-based programming concept can be extended so that the packets arriving at the router are stream-based, i.e. by a stream-based active network. **The Wandering Logic Intelligence model gives preference to the hybrid approach.** Thus, *shuttles* (active packets in WLI) have a dual nature. A RCM node or an active one can e.g. interpret them. In turn, *netbots* (network elements) are in general both RCM and active nodes in terms of hardware and software respectively.

In addition, the model *postulates that active nodes must best match the structure of the active packets at the time of delivery*. WLI uses the abstraction of a *Logical Computing Element (LCE)* to address the two schemes in computing, *spatial* and *temporal* (as a distributed resource).

Because the stream processor is similar to a fine-grained, super-pipelined processor, results are produced on each clock pulse. If the clock speed is equivalent to the line speed, then stream processing can operate at line speed, with a finite amount of latency. This simplifies design by allowing packets destined to be discarded, to be dropped at the end of the pipeline; otherwise, a mechanism must be developed to discard the packet while it is entering and being processed by the pipeline. Because of the pipeline design, the stream-based processor relies on a modular and configurable set of functional units. Furthermore, stream modules can be strung together to provide a desired functionality.

In a fully configurable stream processor, one can add, remove, and modify modules and interconnects on the fly. In addition, modules can be developed independently which greatly simplifies the implementation of a design. If a module later in the stream needs results from a module earlier in the stream (e.g. the result of the route lookup in the case of the reconfigurable router), packets may be accompanied through the processor by a small amount of additional data, which contains calculations from various modules on a separate data bus. Because the algorithm is known in advance and the amount of additional information is known, a *stream interface controller (SIC)* within each module can ignore, pass on, or use the information as needed.

4.5 SUMMARY AND CONCLUSIONS

Recent developments in networking technology testify the trend for ever-increasing user demand for more functionality from their network devices, including quality-of-service and policy routing, multicasting, firewalls, network address translation, mobile nodes, etc. *Run-time flexibility* in network devices is clearly a problem that must be solved, as future networks will be multi-functional, heterogeneous, and scalable in nature.

As future network architectures are supposed to become *highly adaptable* [MNK97] and *active* [Tenn97], they will require development of *flexible routing and nodal processing architectures*. In addition, if route-processing time exceeds 0.27 ms for IP switching, the number of packets that can be switched drops significantly [LinMcK97]. *As line speeds increase towards terabit switching, the upper bound on route processing time restrictions will become more stringent, which will eventually cause problems for systems that rely only on software algorithms to perform active network reconfiguration.*

FPGAs proved the flexibility of a general processor at near-ASIC speeds, yet at a cost much lower than that of a custom ASIC. Computations and data that would fit on a single chip only by sequentially reusing a single CPU a decade ago can be fully implemented in a spatial data pipeline on a single FPGA today. As available silicon continues to grow, we can fit even more computational problems into single dies using spatial data flow, thus off-loading processors and increasing the range of feasible applications.

The competition between the *"active school"* and the *"configurable school"* in network design, as we observe in the case of routers will continue in the years to come. However, combining both approaches brings more benefits, especially when the referee is formal logic.

The widely quoted "90/10 rule" states that 90 % of the program runtime is consumed by 10% of the code. The rule reflects the fact, that *only small portions of the software become the performance bottlenecks* consuming the largest part of the total computation time. Balancing code is necessary for completeness, but its execution speed does not limit performance.

Consequently, a *hybrid approach* (in terms of hardware) such as the GARP architecture [HaWa97], couples a processor with a configurable computing array. The array computes the application's performance-limiting portions (10% code/ 90% time) with high parallelism on densely packed spatial operators. The processor, in turn, deals with the computation's non-critical portions (90% code / 10% time) into minimum space. Yet, nothing argues against feeding the processor with some "active" portion of data, i.e. extending the hybrid approach to software.

Finally, why should not we use the same advantage in the network domain ? The compromise that can be derived by means of formal logic could involve some performance test benchmarking (in both software and hardware) for the user perspective of the best fit of the application. As reported in [SiMi94] and [Sim94a], this proved to be a suitable approach in meeting praxis with some *hybrid* theory model (at least for video communications). Yet, in order for architecture to be checked that way, it has to provide all possible degrees of flexibility from the very beginning.

The *reconfigurable router* is an important step in this direction, because it enables the creation of *both `actively reconfigurable networks*. We define a *reconfigurable (adaptable) network* as those network elements which stay in direct relationship to each other and use the reconfigurable aspects of the computing system in real time following automatically a certain management policy requested on demand when an application (or a group of them) decides which network configuration is required.

Are reconfigurable networks the right solution to future communications ? Definitely --- not alone.

Reconfigurable networking can essentially contribute to develop active networks, scalable multicast packet caching algorithms, and new quality-of-service switching algorithms. Yet, it is not flexible enough, at least not as much as thought and software.

> In fact, spatial and temporal hardware computing and networking designs are mutually complementary; they depend on technology developments and define the performance and the quality of software implementations. Therefore, *fast hardware reconfiguration* and *efficient forwarding algorithms* along with an *integrated software/hardware application life cycle support* comprise a realistic complementary approach to active networking to meet future network demands.

We consider active networking and configurable computing as very promising research approaches in adaptable systems design for the next few years to come. Both of them have some drive towards convergence.

Personally, we believe in the unification of the two system approaches. However, the requirements on future communications systems, especially in the wireless domain and in the era of mobility and multimedia, will rise continuously with an increasing rate of complexity. Without a formal base, we cannot get a clear picture of what is really going on in networking today. For a couple of decades, formal description techniques, such as SDL, LOTOS and Estelle, have proved to be valuable tools for the developer in designing and verifying network protocols and services. They are broadly accepted in industry and are still used along with modern programming environments such as UML and Java in different areas. Using formal methods will even gain on weight in planning and verifying practical network solutions in future.

4.6 OUTLOOK

In the mid 1990s, thanks to increasing transistor and interconnection densities, a new form of reconfigurable element emerged: *the reconfigurable data path* came in response to the call for more parallelism and higher performance in the now larger ICs. Reconfigurable data paths have coarser-grained reconfigurable units than the FPGA; they can accommodate reconfigurable nibble, byte, or wider ALUs. They also provide a slightly more regular, but still configurable interconnection fabric, while maintaining plenty of registers and local memory.

In this way, the fabric lets programmers map regular computation-intensive operations (e.g. encoding of multimedia data) in a spatially pipelined manner onto the device, while clock speeds remain close to those in CPUs with custom ALUs. In near future, coherent application-development environments will let a program's small but time-consuming kernels map to a reconfigurable coprocessor, while the remaining part of the program is executed to run on a traditional instruction-set processor.

4.7 DISCUSSION

The rapid growth of networking has created an ever-present demand for higher performance and throughput. Meanwhile, the network standards are in constant flux, and must continuously evolve to provide the latest support for security, integrated services, and other enhancements (such as the latest version of the Internet Protocol, IPv6). Thus, network designers have to choose *a technology that is flexible enough* to keep up with the latest standards while providing the highest possible performance.

Much research has recently been done in active networks, which increase network flexibility by allowing the routers to be reprogrammed, often at the cost of lower throughput. A reconfigurable router implemented on a *Reconfigurable Computing Machine (RCM)* can provide the *flexibility required for active networking* while approaching the high throughput of inflexible application-specific integrated circuit (ASIC)-based routers. Active networks [Tenn97] and open signaling [OPS97] are the recent attempts to reconcile the seemingly contradictory goals of flexible and fast routing. While these techniques have shown much promise in improving the flexibility of networks, there remains some concern about their possible effects on performance.

The flexibility requirements of active networks suggest the use of *general-purpose processors* for implementing the software reconfiguration algorithms, but there is little indication that contemporary general-purpose processors will operate fast enough to meet throughput requirements. On the other hand, application-specific integrated circuits (ASICs) that can be relied upon for maximum router throughput cannot meet any of the goals for network flexibility.

In the previous two chapters, two recent trends in network research were identified:

- The outwards, hardware-driven approach: expanding reconfigurable computing from the locally centralized processing areas (desk-top and workstation computing) to distributed processing in the network domain while address network elements, such as reconfigurable routers;
- The inwards, software-driven approach: active networks and software processing technology in the physical layers, such as software radio.

These two approaches are in fact complementary and are driven towards convergence. The following criteria/tradeoffs are becoming characteristic for the design of the descending network architectures.

4.7.1 MIXING ACTIVE AND PASSIVE FLOWS

If we deploy an active network over the current Internet, do we need also a specific addressing and routing scheme for active packet processing and transfer separate from the Internet traffic ?

The answer is that one of the most exciting possibilities of the Active Networks is their ability to *mix* active and passive flows in a transparent manner. Using a specific addressing and routing scheme could result in having only a "configurable network", rather than an active network. On the other hand, dealing with active packets in a passive network remains a challenge, especially when we consider fragmentation or dynamic routing.

There is some ongoing research[67] on this topic and submitted proposals toward a possible solution, e.g. by embedding AN information in routing protocol schemes. However, some active applications may require an, - at least -, static route, so that the first implementations will most probably deploy virtual circuit like mechanisms.

4.7.2. FLEXIBILITY VS. SECURITY

How the wireless network can support the deployment of new services and applications ?

This is a non-trivial question, since it addresses the kernel of the design problem, the trade-off between flexibility and expectations: a new networking architecture must be open and extensible enough to accommodate new services, but at the same time also restrictive enough to meet performance and security requirements.

[67] http://search.ietf.org/internet-drafts/draft-galand-an-routing-00.txt

4.7.3 CONFIGURATION VS. ENCAPSULATION

Lets us consider a reconfigure router being used within an active network and receives a packet that is a capsule. The router will decode the packet, if it contains software code, and then send the code to a general-purpose CPU for compilation and hardware/software partitioning. After the partitioning is performed, appropriate portions of the code can be installed in the hardware and the code can now process the next incoming packet.

Alternatively, the packet can contain *a hardware reconfiguration description.* It can be directly installed into an available reconfigurable processing engine. This strategy provides complete reconfiguration of the router *without human intervention* and little, if any, knowledge of hardware design is required.

Using a reconfigurable router, all the operator has to do is:

1) send another active network packet with new code to the router, or
2) install the new code using a terminal interface.

The change is all at the hardware layer, is all performed at run-time, and is performed without requiring detailed knowledge of how the operating system kernel works.

These features are especially important to regular users who do not care to know any details about how the network operates. In this, we could have a customizable network in much the same manner we configure our Linux desktop environment with the KDE toolkit.

The basic premise behind deploying a configurable router in an active network is, however, the support for *dynamic hardware/software co-design* within the switch architecture, which allows traditional software-based routing code to be automatically mapped onto the hardware *when* and *as* needed. This feature is an important first step towards the realization of active networks. We believe that formal methods can do a lot in the pre-design of such systems.

The reconfigurable routing and switching technologies can also be applied to base stations and radio network controllers for mobile networks.

* * *

CHAPTER 5: THE WANDERING LOGIC INTELLIGENCE

"The meshes of the net of Heaven are large;
far apart, but letting nothing escape."

LAO TSE

5.1 ARCHITECTURAL BASE

Like most networks, an *active network (AN)* consists of an interconnected group of nodes executing a common runtime. The nodes are linked across a LAN/WAN via point-to-point or shared medium channels. The system is built on the link layer services of the channels to provide network layer services to distributed applications.

Unlike IP, the network service provided by an active network is not fixed, but *flexible*; in addition to providing IP-style routing and forwarding, the AN allows *applications to introduce new protocols into the network* by specifying the routines to be executed at the intermediate network nodes instead of the simple IP forwarding service. As a result, the applications may delegate a portion of their processing into the network.

The active network approach pursues *three goals* in network protocol design. All of them describe more innovative forms of engineering than currently achieved on the Internet: a) distributed development and use of new protocols by mutual agreement among the interested parties, rather than the centralized (standardized) one; b) dynamic (runtime) introduction and deployment of new protocols; c) simultaneous use of a variety of different network layer protocols.

In the following paragraphs, we introduce the desired characteristics of a new type of *hyperactive* network architecture, which represent the base of the *WLI model* described in this chapter.

- Active nodes can be *mobile* and *reconfigurable* (in terms of software and hardware) during runtime. Reconfiguration could be just another type of network service.

- Depending on the *class* of service to be installed onto an active node, a special manipulation of the capsule in the execution environment may be applied (including cashing of its contents and/or state) to ensure *awareness* about the flow. The *class* of service is a new concept in WLI used to describe *multiple code systems*, either at the "byte level"[68], or at higher-layers associated with different service functions such as encoding/decoding of MPEG video streams.

[68] as e.g. the support of native Intel x86 object code and JavaVM code in a heterogeneous active network (which this is the case of PAN [Nygr99])

- A*ctive nodes can adapt (themselves) to communications in such a way to best-match the structure of the active packets (shuttles) at the time of delivery.*

- Capsules could carry code and data not only for the execution within**,** but also for the upgrade/degrade and re-configuration of active nodes in terms of software and hardware. In this way, active nodes can be *modified*. For this reason, the capsule APIs and the execution environments could be extended by special routines allowing the accommodation and execution of code that changes a netbot's configuration and resources. In this way, new functionality could not only be delivered to and injected into the active node, but also distributed and optimized throughout the node itself.

- In addition to the data related code, capsules could also carry *genetic code* which can be selectively invoked by special routines at the active nodes to perform structural changes in the node/network architecture (e.g. to spawn/collapse a virtual (sub-)network, to expand/derive a new reachability (sub-)tree for routing, chapter 6, etc.).

- It could be useful to allow the creation of new capsules (or the replication of "old" ones) in the intermediate active nodes under the supervision of the node operating system (NodeOS). Furthermore, a special class of capsules could be capable to replicate themselves and to *create/remove/modify other capsules and resources* in the network. The term "resource" could be extended to an entire *virtual* active node.

- The network protocol itself could be particularly embedded within the capsule. Furthermore, code distribution throughout the network and inside the nodes *can be maintained by the capsules themselves.* A capsule approaching a netbot could *re-configure itself* becoming a *morphing* packet to match the netbot's processing requirements.

A capsule could also have some additional *activity* properties for code distribution and code execution. The *Wandering Logic Intelligence (WLI)* ([Sim99e], [Sim99f], [Sim02a]) is a new approach, which introduces an extended concept of a secondary encapsulation level to model evolutionary active and configurable mobile networks.

The medium-term goals of the WLI approach which are not part of this work, but are going to be pursued in a future research project, are summarized as follows:

- to provide a formal means for the specification and verification of the generic temporal properties of active mobile nodes and packets;
- to support the reflexive dynamic adaptation of both mobile code (software) and node architecture (software and hardware);
- to provide the formal means for specification and verification of dynamic QoS properties in wireless networks at both application (service) and packet level;
- to assist the formal transformation of the systems properties into mobile code,
- to simulate system behavior previously to implementation.

The following sections introduce the WLI framework.

5.2 THE WLI APPROACH

5.2.1. THE COLLISION OF THE INTELLIGENCE PARADIGMS

There is a definite distribution of functions and roles among network components within outbound-signaling circuit-switched (telephony based) Intelligent Network. (Of course, there is also a distinct distribution of roles and functions in present day packet switched networks: servers, switches, routers, proxies and terminal equipment. The example is taken for demonstration purposes.)

Thus, the standard SCP-IP relationship is e.g. not simply the client-server one (typical for data networks), but rather a processor-coprocessor delegation of service logic (e.g. playing announcements and collecting user input) to a subordinate unit, the Intelligent Peripheral, which delivers back some result to the Service Control Point. The SCP then continues to control the flow of the service in the well-known *main ()* routine manner. Even in a distributed IN, Service Nodes are not supposed to serve each other in the common ''data packet'' sense, but rather to have a local (centralized) control over certain resources, services and subscribers. In addition, ongoing IN standardization do not consider alternative network technology integration paradigms, but rather extend the SS7 model by ROSE functionality, such as e.g. invoking functions (ETSI INAP CS-2, [Fayn97]).

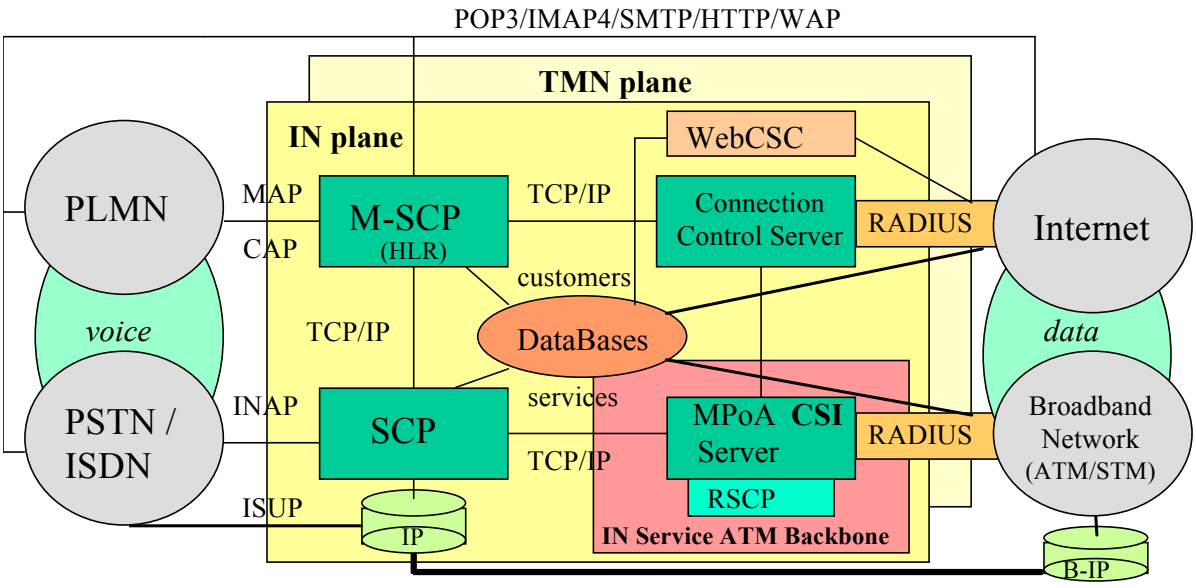
On the other hand, the packet data world is handling intelligence not in a call flow but rather in an application-oriented way while addressing variable bandwidth technology and OSI protocol *stack permutations* such as e.g. running SS7 over[69] TCP/IP.

Such developments are a clear sign that the layered architectures and the *fixed* treatment of IN functions, databases and services can no longer exist in a heterogeneous distributed network under rapid development. Even *sculptured object paradigms* such as CORBA, TINA, or DCOM and powerful IN architectures based on ''clustered, multi-protocol SCPs" (Figure 37) cannot handle all aspects of the network evolution as far as they remain *rigid*. One major aspect of the network has been neglected until now: it is *temporal* and *living,* i.e. *changing.*

For example proprietary IPs can grow up to autonomous multiprotocol ''virus-like'' CTI point solutions hiding micro data networks in themselves.

A time for change in network design thinking has come. Now, we have the chance to adopt a *stepwise approach* for building the kernel of a new communications cell … Where could we start? Perhaps flexible *nomadic intelligence* may be an option.

[69] http://www.ttiweb.com/corporate/press/ssip-pr.htm



Legend:

PLMN	Public Land Mobile Network	ATM	Asynchronous Transfer Mode
PSTN	Public Switched Telephone Network	STM	Synchronous Transfer Mode
ISDN	Integrated Services Data Network	RADIUS	Remote Access Data Interface User Service
INAP	Intelligent Network Application Protocol	TCP/IP	Transport Control Protocol / Internet Protocol
ISUP	ISDN User Part (Protocol)	MPoA	Multiple Point of Access
SCP	Service Control Pont	HLR	Home Location Register
M-SCP	Mobile SCP	POP3	Post Office Protocol, Version 3
RSCP	Remote SCP	IMAP4	Internet Message Access Protocol, Version 4
IP	Intelligent Peripheral	SMTP	Simple Mail Transfer Protocol
B-IP	Broadband Intelligen Peripheral	HTTP	HyperText Transfer Protocol
CSI	Customer Services Interface	WebCSC	Web (based) Customer Services Control
MAP	Mobile Application Protocol	CAP	CAMEL Application Protocol

Figure 37 A clustered SCP configuration for converged networks

In the following, we will show how flexible intelligent network elements can be used for functionally integrated multimode logic control in advanced *nomadic intelligence* architectures that offer completely new perspectives for communication services in converged networks.

5.2.2 Nomadic Services and Their Logic

The idea of migrating service functionality comes out of traditional network convergence solutions such as the One Number Service (ONS) for fixed and mobile networks. Figure 38 and Figure 39 illustrate two possible variants to realize this service architecture. The essential difference between them is the location of the Service Resource Function (SRF). We can simply ask what would come out to being if we *let this function roam* between the network elements for the reasons of evolution, effective resource utilization, adaptability and QoS guarantee.

To assist new services and customers, intelligent networks require permanent upgrades in hardware and software components, which are not always optimally exploited at their location (Figure 40). Even a distributed architecture, which dedicates certain resources for local usage, is not adapting (not yet) to variable load and service usage.

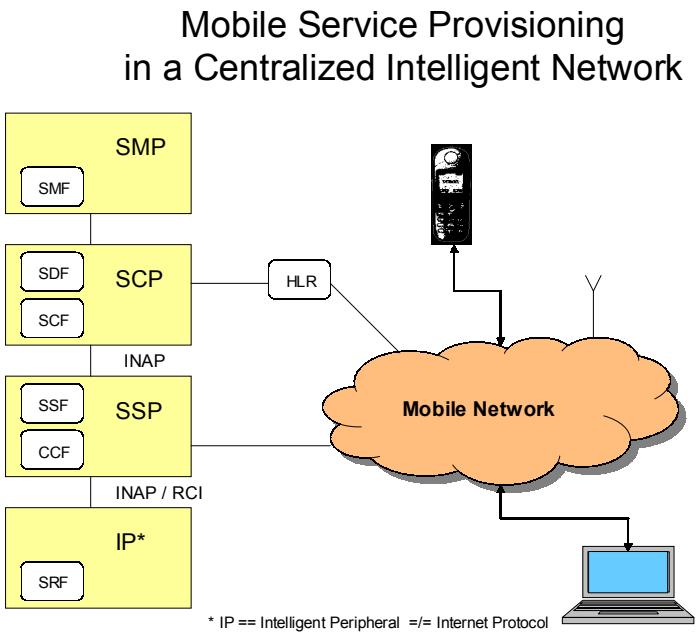


Figure 38: A centralized architecture for the realization of a mobile IN service

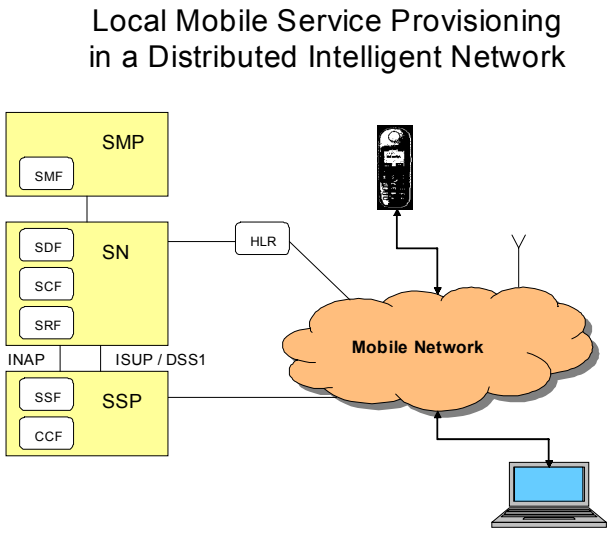


Figure 39: A distributed architecture for the realization of a mobile IN service

Thus, next to *HOW*, the question of *WHERE* to physically implement an IN function has always been crucial for network designers to provide for more efficient and effective services. Now, in the era of platform independent object-oriented technology this subject obtains a new thread: *mobility*. Therefore, we have two choices:

- a non-deterministic worst-case-dimensioned service within a rigid network architecture requiring software and hardware upgrades, or
- a deterministic service within a dynamically configurable network architecture, *FINE,* based on autonomous nomadic services taking care for software updates and optimal utilization of hardware resources to a certain grade that can be proved by algorithms.

SCF + SDF : Number Translation, VPN, ...

SCP + SDP + SRF : Voice Recognition, UMS, ...

SCP + SDP + SRF + SSF : Local Operators, ...

SCP + SDP + SRF + SSF + GWF : Internet Telephony ...

Figure 40: Overall trend – increasing complexity of node related IN functionality

In some cases, it seems reasonable to temporarily enact and allocate/move different services and their resource areas depending on the eventual customer demands, instead of permanently increasing the functionality and the capacity of the network elements (Figure 41).

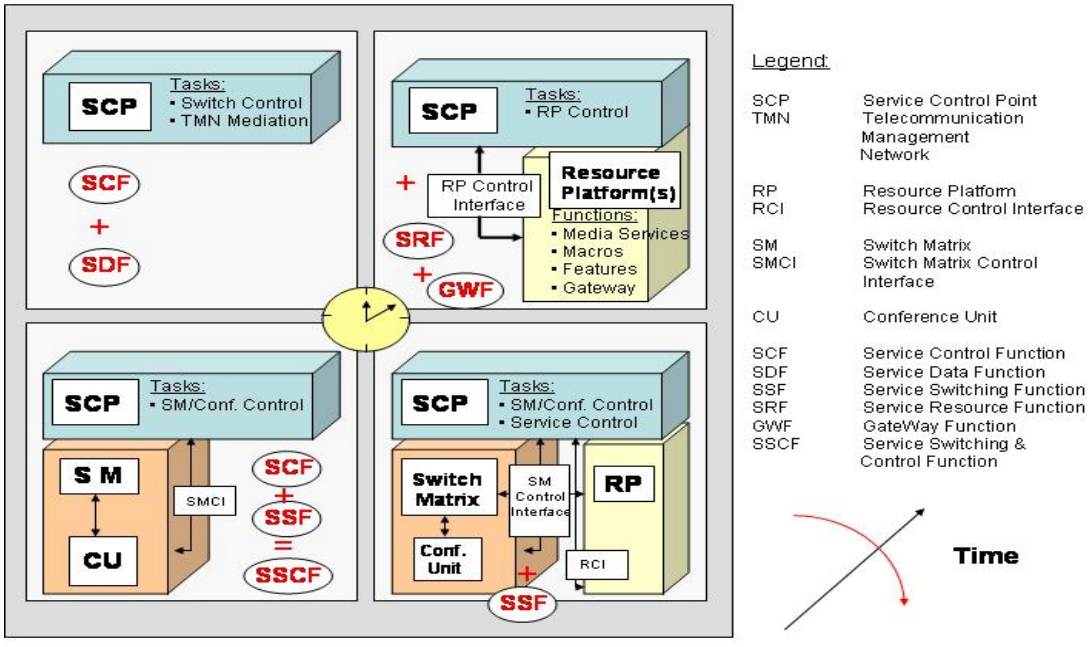


Figure 41: A possible node utilization cycle

This implies an event-driven, i.e. deterministic, redistribution of roles among the network elements (in particular, client, server and agent) to satisfy network performance and user demands on service configuration, availability and QoS (Figure 43).

In the following sections, we provide a few definitions which are going to be used in this work.

5.2.3 WLI DEFINITIONS

The following definitions provide the base for *SmartNet* [Sim98], a new generation IN architecture we used to define the WLI formalism, Figure 42:

> **WLI Definition 1:** A *Flexible Intelligent Network Element (FINE)* is a configurable multi-mode active network element. It can be deployed in a single, dual or triple mode of operation as *independent* (I), *dependent* (D) and *autonomous* (A) unit w. r. t. communicating services, i.e. as *server*, *client* or *agent (proxy)* accordingly. All three modes can be active at the same time at different layers of functional abstraction as far as this does not conflict with basic system design principles. A FINE may be a physical entity (active node), a piece of software (service component, active packet) or a virtual sub-network by itself.

There is no distinction between network internal and peripheral elements. Even network terminals can be regarded as FINEs operating in some of the three modes.

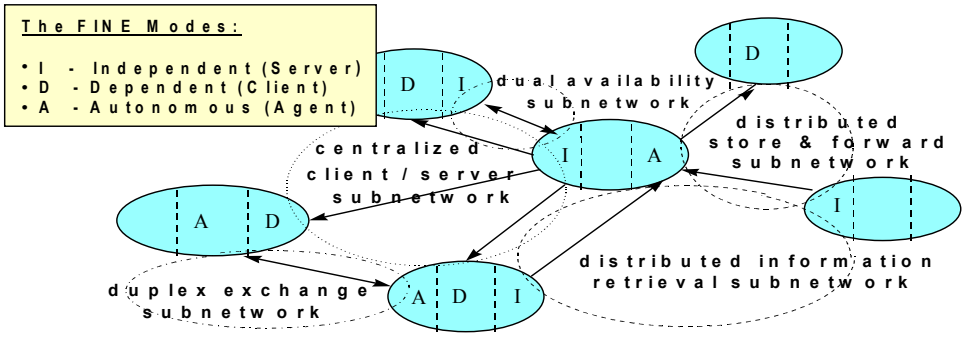


Figure 42: An example of a FINE configuration

> **WLI Definition 2**: *The FINE Architecture (FINEA)* is a versatile network intelligence construction of FINEs and *links* between them. Each link may be unidirectional or bi-directional and carry one or more *channels*. The FINEA has a dynamic, temporal character; all elements and links between them are temporal functions. FINEs and links can be created, re-configured or removed in every new state of the network. FINEs can *temporarily* adopt certain roles in the network. It is required that
>
> - agents be assigned to negotiate about the distribution of roles among FINEs at a certain level of abstraction (active network), and
> - data, functions (service logic) and roles (modes) migrate upon request from one FINE node to another, thus allowing for dynamic network configuration.

> **WLI Definition 3:** *Nomadic FSLs* (flexible service logic) are the smallest autonomous service elements (SDUs) known as features (functions) including data (PDUs), also as parameters (variables) and database contents[70], and being transferable from one (fixed or mobile) node of the network to another to provide *intelligence on demand*.

Thus, the FINE model leads to a self-configurable, adaptable, and domain-aware intelligence architecture where network properties such as database and feature mobility are parts of the service itself. Although the FINE approach has an IN telephony origin, it can be used to virtually model *any* type of network.

This simple model offers a unified and structured evolutionary approach to intelligent network design and configuration which addresses the two basic architectures, centralized and distributed IN and allows for a flexible, but *deterministic* (event-driven) *spread of intelligence* across the network matching user demands wherever and whenever required.

This approach allows for effective treatment of problems such as feature interaction (multi-protocol capability), service mobility, database updates, etc. The FINE model can be easily implemented using approaches such as JTAPI (Java Telephone API, [JTAPI]) for applications and the Agent Transfer Protocol [ATP] for transferring service logic as mobile agents between networked computers.

The FINE architecture presents a new service view at IN (in ODP terms) when compared to the classical vertical and horizontal layering of intelligence in networks. It can be centralized, distributed or both at the same time depending on the network size, architecture, service configuration, performance, scheduling and QoS requirements, as well as on the subscriber and service migration flow.

This paradigm can be used as a *unified, vivid object-oriented IN* model where the roles of the network elements (SSP, SCP, SMP, SN, IP, servers, routers, CPE, etc.) are temporary and on demand. Within the FINE paradigm of a dynamically configurable intelligent network, services and their logic are gaining a new value by becoming *autonomous and truly mobile entities* throughout the nodes of the network reaching even the terminals to turn them temporarily into agents, clients or servers whenever and whenever required.

Thus, service logic can be transferred, installed and mounted on demand among the FINE Controllers and the terminal equipment to provide optimal resource utilization and QoS. This is in fact one of the major objectives of active networking.

5.2.4 INTRODUCING THE WANDERING LOGIC INTELLIGENCE (WLI)

Recent research on active networks approaches the problem of effective service provisioning not by moving services along network nodes, but rather by using the means of (mainly Java driven) object-oriented encapsulated software technology up to the deepest layers and elements of network protocols.

[70] Encoding, security and data-split are considered to be part of an FSL's contents scheme.

No matter of whether we talk about a *NetScript* language [YeSi96] to program functions for sophisticated packet stream processing in network nodes, evolving to form *capsules* or datagrams carrying code [TeWe96] by reference, or "smart packets" [SmartP99], *switchlets* or "active packets" [Alex98a], *CANEs* or "Composable Active Network Elements" [Zegura96], -- in all these works the search for data and process customization beyond the limits of performance, up to the application, up to the user needs is so evident, as it was with the definition of different QoS layers matching the OSI-centric communications model in the mid 90[ies]. The belief creates the reality. This holds until the next change of the paradigm. Yet, we still miss the quantum jump, the profound networking uncertainty equation.

For the time being, there are two approaches to improve an application's performance: a service oriented and a network oriented, both considered as *objects*, both strictly separated in hardware and software. Programming is mainly a *soft* method that leaves certain options and degrees of freedom to adjust a model's behavior towards a desired result. If a high-level abstraction program performs well, it is then translated into a "lower" language and eventually split into modules and instructions that fit the hardware and thus better perform. Finally, if this works well, the entire program is being molded into silicon to match the "click- to-switch" or "be there" requirement and perform for the best. Further improvements are only due to the physics. Unfortunately, there is no way back … (yet)…to check if the things could work the other way up.

If you have a new idea, you need a new model that most probably needs a new language to express its axioms, predicates etc. logic constructs. The trouble with modern computer science, however, is that it still lacks the freedom of the Greek philosophy era. Every problem is digitalized and reduced to a set of objects and functions (both *corpuscles*) that express the same formalism in the broad sense. Be in hardware or software, it is the same logic and the same *programming* language, no matter whether we unfold services or inject pro-active capsules into network nodes. What we miss is the analogous *wave* in the digital world, the sparking *process* that describes the quantum nature of evolving multiple realities, some kind of a free, *Wandering Logic Intelligence.*

5.2.4.1 The WLI Model

The WLI model represents an instantiation of the FINE architecture mapped to the description of an active network with adding the details of *mobile* components, [Sim99e].

> **WLI Definition 4:** A Wandering Network is a FINE architecture[71] defined by the following characteristics:
>
> 1. A network element can exchange its functional modules[72] and thus perform different tasks in different deployment modes / roles. Each mode is characterized by an operational state. The network element can perform multiple roles simultaneously. There are three generic deployment modes:
>   - Independent (Server),
>   - Dependent (Client),
>   - Autonomous (Agent).

[71] Definitions 4-6 in the previous section.

[72] The functional modules can be aggregated either in software or in hardware, or both in software and hardware.

2. Each element can acquire a certain role or a functional task in the network with reference to a particular sort of transmission or network constellation. The transmission sort defines a distinct form of executing the information transfer in a network. It is related to a certain subset of network elements which participate a transmission session, e.g. a conference or a multicast group, following a structured functional scheme (the constellation), called *morpheme*. As soon as a new element joins or leaves the morpheme, a new transmission session is started. The roles of the network elements can be re-distributed or re-defined within the new session. The transmission sorts can build hierarchical relationships among each other.
3. The network constellation, the morpheme, represents a configuration of network elements, which are related in a certain way to each other. It reflects the *active* network topology w. r. t. a particular transmission sort for the correspondingly active transfer session. A morpheme is uniquely identified through a variable parameter or a function, called *state* of the network. Since transmission sorts can be hierarchically constructed, network constellations and their states can be hierarchical as well.
4. A network element can participate in one or more morphemes and thus perform multiple roles or deployment modes simultaneously. Each one of these roles of the network element is described through a state vector, called *index* of the network element within the particular constellation (morpheme). The constituents of a morpheme can smoothly exchange their roles during a transfer session. The storage and the transfer of state indices accompany this role exchange. The state index of a morpheme, the *state*, is composed of the state indices of its constituents.
5. The network element, called *netbot*, is an aggregate of roles and functions, which are described with its state vector. A netbot's role or a deployment mode can be enhanced or modified with new functions as time passes. These changes are registered as new information, called *facts*, in the state vector of the network element. Netbots are capable to exchange facts.
6. The organization of knowledge quanta inside a netbot or within a shuttle flow is called *logic* in WLI. The new type of network is called *wandering*, since shuttles enable the transmission of knowledge quanta, and hence the network evolution. *The intelligence*[73] *of a wandering network at a certain point of time is an aggregation of its logics and their interplay.*

The Wandering Network approach differs from the well-known research frameworks of Open Signaling and Active or Programmable Networks essentially through two characteristics:

(i) *adaptable function migration,* and
(ii) *pulsating metamorphosis* (to be discussed later).

In this way network, functions can change their hosts (netbots), wander and settle down in other hosts, thus creating a valuable statistics about the frequency of usage of wandering functions in the network[74]. The results obtained after a careful evaluation of this data can be used for the design of new network architectures and topologies.

[73] We provide this explicit definition of the term *intelligence* in context of a Wandering Network to avoid further speculations and interpretations about its meaning as this was noted by Bateson, [Bate72].

[74] e.g. in connection with the maintenance of a Virtual Home Environment for end users.

Figure 43 illustrates the migration principle of the Wandering Logic[75]. It shows two network elements known as Service Nodes in Intelligent Networking, SN1 and SN2.

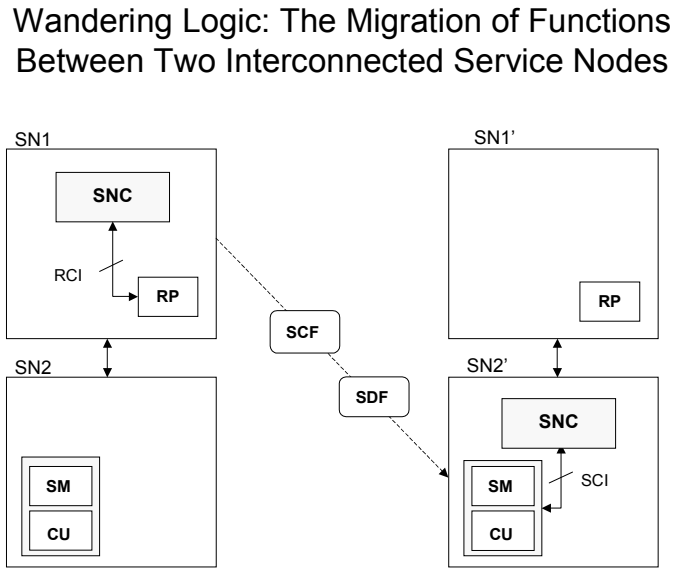


Figure 43**:** The Function Migration Principle of the Wandering Logic

The first network element contains the functionalities of a Service Node Controller (SNC) and a Resource Platform (RP), whereas the second element contains a Switching Matrix (SM) and a conferencing unit (CU). Both network elements are interconnected. At some point of time, the Service Control Function (SCF) and the Service Data Function (SDF) can wander from SN1 to SN2.

This can be performed[76] upon request, through signaling or as a result of some kind of an automatic mechanism in response of an application event of some change of the operation environment to adapt to some transmission/user criteria such as congestion, cost, QoS, etc. In this way, a network element can require additional functional modules for it is depending on the service, the user location and the actual environment. This theoretical framework does not claim to be final; rather, it can be regarded as a stepwise, upgrading approach of creating and using dedicated domain libraries to treat a particular subject of interest.

[75] The term ”logic“ has an IN heritage. It usually stands for the so-called Service Logic (SL) , which denotes the components of a network service. An SCF can control several different service logics. Through the migration and the new creation of the SCF in another network node, the corresponding SLs wander throughout the network as well. The classical IN model does not consider the interoperability of the IN functions with their environment, i.e. with the host and its resources. This is possible now with the new WLI model. Nevertheless, we have limited ourselves first to the plain migration and execution of IN functions in different network nodes, which can be compared to the operation of mobile agents or AN capsules in a kind of “sand box“ on the new host.

[76] either via software transport (e.g. a java mobile agent), or activation/deactivation of resident software on the nodes.

In WLI, active nodes and active packets are called *netbots* and *shuttles* correspondingly. The main distinction from other AN approaches elsewhere is that the programmable active nodes (netbots) and their components are considered *mobile*[77] and *reconfigurable* in terms of both software and hardware. Netbots may move in some direction with a *constant* velocity.

Furthermore, shuttles differ from capsules in the property that they can carry *genetic code* (in addition to the "passive" executable code) which is capable to invoke structural changes in the network (e.g. to create overlays or generate computing elements).

The WLI approach represents a next step of network virtualization and evolution based on the Wandering Network Principles (Section 5.3), an *autonomous hyperactive network*.

Next, we distinguish between *decks* (a netbot's resource areas) and *docking ports* (the execution environments for shuttles). The last nautical definition is reserved for the NodeOS as a *Cockpit*. A sequence of shuttles moving in one direction along a channel is called *flow.* A flow may contain multiple control schemes, i.e. protocols defining network services. A set of netbots sharing the same flow is called a *fleet* of that flow. Following the FINEA approach, the elements of the WLI architecture have a temporal character, i.e. they can be created, configured and removed (on demand) upon *actions*. The WLI model comprises a mobile active network with both nodes (netbots) and packets (shuttles) being *active* (i.e. executable) and *reconfigurable*. Thus, a netbot processing shuttles in a docking port can change its state and re-configure its decks and links for further actions. It can also change the state of the shuttle (provided, an adequate permission is given[78]).

On the other hand, a shuttle approaching a netbot can re-configure itself becoming a *morphing* packet to provide the desired interface at the docking port and match a netbot's requirements. This operation can be based on the destination address and on the class of the netbot included in this address. The assumption in this case is that the sender netbot was not taking care about arranging this for the shuttle (e.g. in a broadcast session).

According to the classification in the previous section, the Wandering Logic Intelligence (WLI) represents a hybrid approach to active networking. It is an open, hierarchical and dynamically structured formal model which allows to address specific problems in communications architectures, services and applications with a great degree of differentiation and flexibility that can be tracked down to the gate level. In the following, we discuss in detail the WLI framework.

[77] We introduced this feature mainly for the reason of satisfying service requirements in wireless networks such as resource management for QoS. This interferes of course with the definition of today's "mobile networks" which consider that only terminals are mobile, but not routers or base transceiver stations.

[78] Note the generalized properties as compared to the capsule approach, [Tenn97]. In terms of WLI, IP packets and capsules are passive carriers of data and code. A truly active packet, should be somewhat autonomous. Therefore, *shuttles* are more likely to be classified as mobile micro-agents.

5.2.4.2 Fundamentals – The Netbot-Shuttle Dualism

Netbots and *shuttle*s are the main elements of the WLI model. On the one hand, they include and refer the basic components of common [passive[79]] packet switched networks, packets and nodes correspondingly. On the other hand they are intended to reflect the fundamental dualistic substance of *net-working* per se as both a complementary and a redundant spatial-temporal entity-relationship diagram of programmed ongoing events in terms of information processing and distribution schemes.

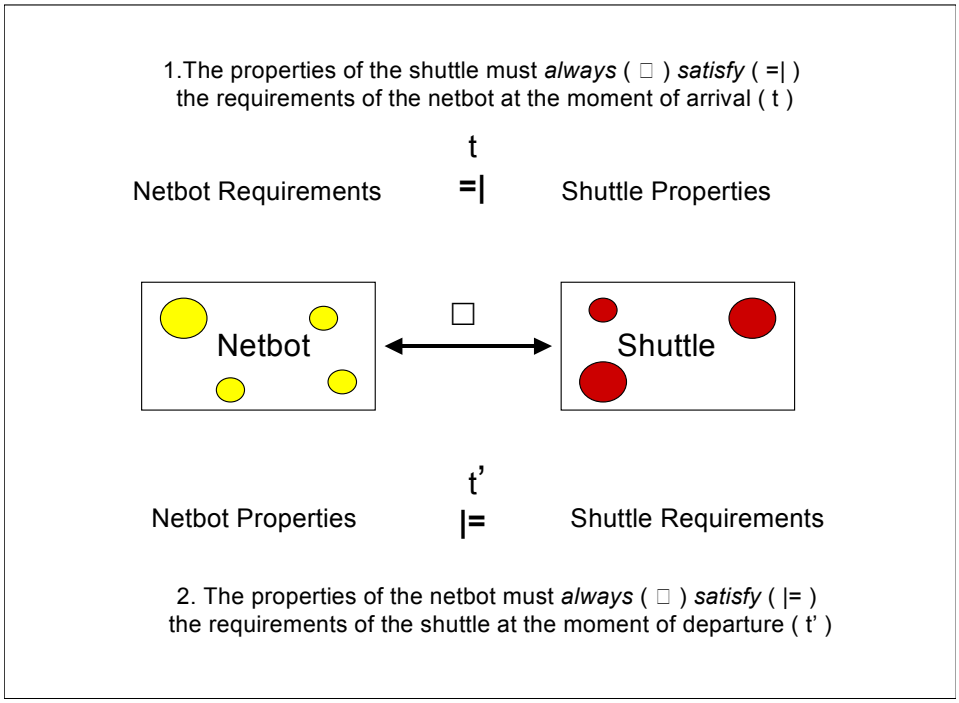


Figure 44: The WLI's basic assumptions

The philosophy behind our approach is that both netbots and shuttles are representing the very same THING, the NET. Ultimately, we can "pack" a netbot into a shuttle and "unpack" a shuttle into a netbot. Figure 44 illustrates the WLI's basic temporal formulas. Under properties and requirements, we understand an element's characteristics as a function of time.

We regard shuttles as portions, bits or *quanta* of information. The same holds for the netbots. They simply have another representation and function in the network. This is essential for the WLI approach.

[79] Again, using the term "*passive"* in the context of networking is somewhat controversial; this is the reason why we use it in brackets. Transport layer protocols (TCP, XTP) provide already flow control mechanisms. Higher layer protocols such as RTP, RTCP and RSVP even address resource reservation and QoS maintenance issues which is, of course, some kind of activity.

We selected this view to describe the system and reason about it with passing time. This is the domain of temporal logic. As we deal with communications, we consider not exactly the method of delivering the information between two spatially remote points, but rather the goal and the result in terms of properties satisfying requirements. For instance, in transmitting video, our goal is to deliver within the timing constraints possibly the same quality (and thus, induce the same sensual perception) at the receiver as it was at the sender.

It is not our concern *how* to fulfill this requirement: by applying transport control mechanisms, integrating error-resilient algorithms in the decoder, implementation issues (DSP, ASIC or FPGA), or simply by triggering a "play" function on a previously stored video file at the receiver[80]. The method is defined[81] as property of the system.

In this way, we can check different options previously to *en*coding, including the ones with configurable hardware and functional split between software and hardware, if we consider the required details. In TLA, we can even include performance measurements as properties of the system ([Sim94b], [SiMi94]).

In the following, we describe the main properties of netbots and shuttles in terms of their composite elements and classes of functions. As we will see later in this work, these properties can be refined, if required, in a particular problem treatment.

5.2.4.2.1 The Netbot

A netbot's functions or services are usually allocated in and performed by the *Cockpit* (NodeOS). However, depending on the system hypothesis to be verified, they may be also distributed throughout the *decks* (resources).

5.2.4.2.2 Common Primitives

The services built into each netbot might include several categories of operations/actions:

- primitives that allow the (active) packet itself to be manipulated (e.g., by changing its header, payload, and/or length): type `processShuttle`;
- primitives that provide access to the node s environment (e.g., node address, time of day, and link status): type `accessNodeEnv`; and
- primitives for controlling packet flow (e.g., forwarding, copying, and discarding): type `controlFlow`.

Additional primitives might provide access to node storage and scheduling (type `accessNodeStorage` and type `accessNodeScheduling`), for example, to facilitate rendezvous operations that combine processing across multiple packets.

[80] This will, of course, not work with live video conferencing; the example is only to illustrate the goal.

[81] This implies some hypothesizing and an opportunistic style in system design. Then we use the means of formal logic to verify the different alternatives with the problem constraints.

5.2.4.2.3 Resources and Their Allocation

Beyond encodings and primitives, there must be *a common model of node resources* and the means by which policies governing their allocation are communicated. The resources to be modeled include physical resources (type `PhysResources`), such as transmission bandwidth, processing capacity, and storage, as well as logical resources (type `Logical Resources`), such as routing tables and the node s management information base. Safe resource allocation is an area that requires considerable attention.

Active nodes will be embedded within the shared network infrastructure, so their designs must address a range of sharing issues that are often brushed aside in the design of programmable systems destined for less public environments.

5.2.4.2.4 The Shuttle

The following list provides some important characteristics of the shuttle concept.

- A shuttle can be turned into a component, or an even netbot, or a fleet upon processing by the NodeOS and interactions with other shuttles and netbot services.
- The shuttle is of temporary nature. Its life span comprises creation, assembling, transfer (incl. re-routing), activation and execution.
- A shuttle represents a *morphing packet*, i.e. its contents can be filtered out, re-configured or adapted to a netbot's actual state previously to processing.
- The *tail* is optional. It can be used for policy differentiation, e.g. in sorting shuttles.
- After entering the node, shuttles transfer the execution control to the cockpit, unless there is some code with a particular mission[82] that has to be granted by the cockpit before executing itself.

The main advantage of the WLI approach is that it unifies the macro-world of active networking and the micro-world of configurable computing in a generic model, which reflects the intrinsic nature of intelligent communications. It facilitates the application design and allows sophisticated network growth, adaptation and rapid introduction of new services while supporting both engineering approaches by making only minor changes on the available infrastructure.

In this way, a WLI architecture represents a *vivid, scaleable object-oriented model* for intelligent service provisioning and control when compared to the traditional horizontal and vertical OSI-like layered network architectures.

5.2.4.3 The Evolution of the ANTS Reference Model

In the following, we review the ANTS reference model for a programmable network architecture and discuss its expansion within WLI framework. Table 4 summarizes the basic characteristics of the ANTS architecture, [WGT98], along with the available options for extension (in *italic*).

[82] The only type of "viruses" which are allowed in WLI are the *pilot* shuttles which have to authenticate themselves as being administrator agents. Of course, future model enhancement may consider multiple types of pilots, if found for appropriate.

Active Nodes	Active Packets
• Have structure that *could be re-configured with time.* • May accommodate some residential program code for processing packets. *Could support multiple code schemes[83] to define classes of services.* • Do processing on packets. *Could be processed by packets. Could do some processing on themselves.* • *Could be mobile.*	• Have structure that *could be re-configured with time.* • May carry program code, but do not execute it. *Could support multiple code schemes. Could carry some code for AN configuration.* • Are processed by nodes. *Could do some processing on nodes[84]. Could do some processing on themselves[85].* • Are mobile.

Table 4: Possible enhancements to the concept of active networks.

WLI generalizes capsules in *shuttles* as relatively autonomous components including both programs and data possibly encoded in some special language with corresponding references to nodes and other shuttles within the same or a different flow (protocol). The "special language" is also a generalization, allowing us to address in a uniform way even hypermedia content information such as MPEG-4 along with the corresponding encoding/decoding routines or references to them in some active network nodes or protocols (Figure 45, [Sim00]). Furthermore, we consider security as a part of the flow encoding mechanism. Using code derivatives for identifying shuttles to guarantee *per-protocol* protection (as in ANTS' capsules) is simply one method using the standard security mechanisms of Java programming (sandboxing and Java bytecode verification, [Weth99a]).

This approach is probably sufficient in the case of "static" nodes deploying a particular protocol and generating the corresponding flow of capsules. However, in a dynamically reconfigurable netbot architecture, it may happen that two or more different sources, being (virtually!) active nodes, participate (i.e. generate capsules of) the same protocol that has been possibly initiated by one of them; the latter node even may does not exist anymore (in terms of protocol deployment). Furthermore, the shuttle sequence of a protocol may be split upon departure from a node and before reaching the "next hop" for some efficiency reason such as building a new virtual active node. All this requires a better authentication procedure of the generated code.

What is appealing in the WLI model is that we allow (and if used, we require!) a *per-shuttle* code protection when interfering with the protocol states is also allowed (opposite to ANTS), e.g. for *pilots.* This "individual" encoding is embedded within the code of selected shuttles during their assembly in the active nodes. The distribution of the decoding keys for pilots follows a special scheme, which is not subject of the present work.

[83] In fact, PAN, the successor of ANTS, already supports this at the byte code level, [Nygr99].

[84] provided, that they are allowed to.

[85] previously to entering the nodes.

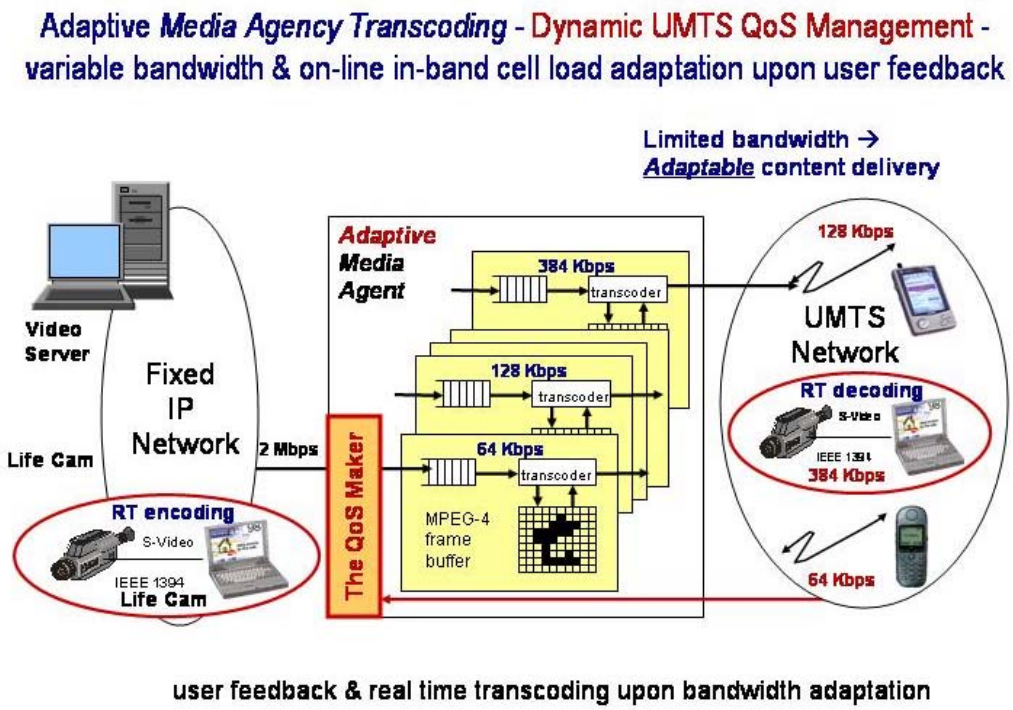


Figure 45: A WLI based adaptive media transcoder

Next, we regard the case of “well-known” forwarding routines or bootstrapping network services (such as the ANTS code distribution scheme) available at all active nodes as trivial and therefore do not include them in the general WLI framework. Instead, we focus on the application specific aspects of the wandering logic where programming code neither resides on, nor is transferred to a “node-by-node” distribution scheme previously to capsule/shuttle processing at the active nodes (which is the ANTS case). Being analogous to ANTS capsules, shuttles in WLI are transferred between active nodes along generic link-layer channels. To support IP-only routing in a heterogeneous network of active and non-active nodes, shuttles can be embedded within ANTS capsules, e.g. using the ACTIVE IP Option field in standard IP packets (Figure 46).

The overall structure of a WLI shuttle with the above implementation as compared to the ones of an RTP packet and a Mobile IP packet is illustrated on Figure 47. Encapsulating information is one of the main principles in network engineering. We are looking to identify some basic structural patterns in protocols that can be used for the formal specification of a follow-up model.

A special feature of WLI is that because of the dynamically reconfigurable nature of the active nodes discussed in the next section, not only an update of the type-dependent header of a shuttle during network traversal is possible, but also all the information of the usually “static” common header (source and the destination addresses, resource limits to be enforced by the nodes, etc.) is changeable, [WGT98]!

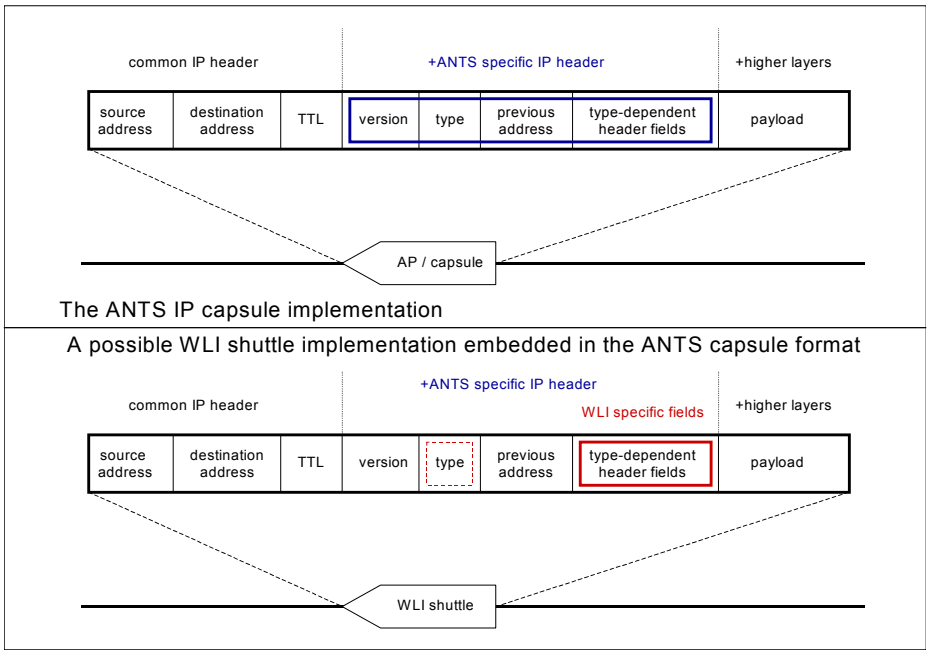


Figure 46: Embedding a WLI shuttle within the ANTS capsule

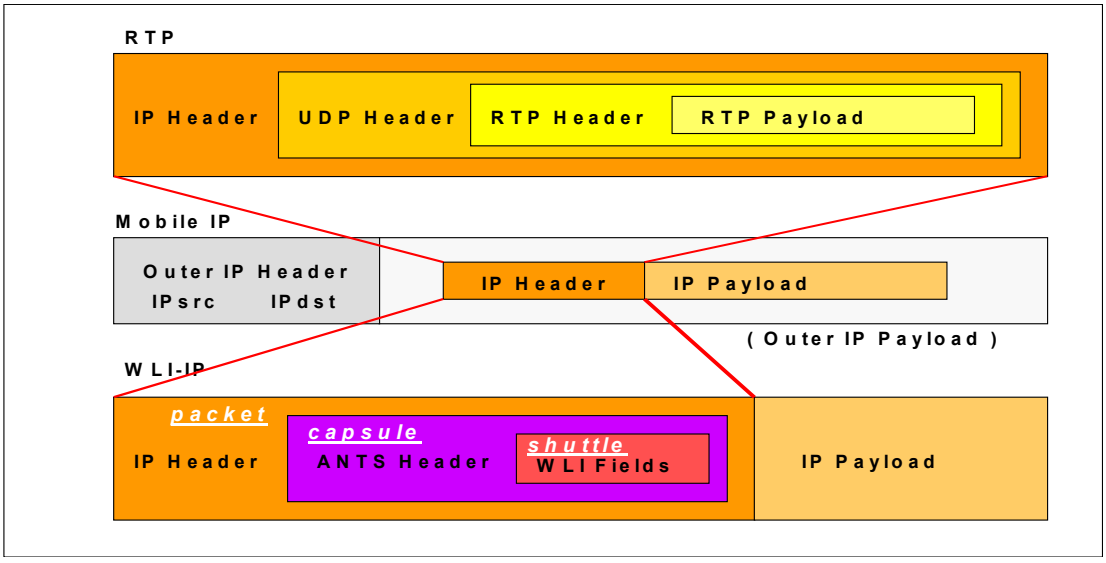


Figure 47: Embeddings of and within the IP header field in comparison.

A *code group* in ANTS is a collection of related capsule types whose forwarding routines are transferred as a unit by the code distribution system (Figure 48). WLI regards code distribution as an external autonomous process that can be generated, e.g. as a WAVE procedure ([DSB96], [Sap96], [Sap99]), or observed in the network. Its definition can be included a priori or a posteriori to the WLI model. We are not going to discuss such details in this work.

In WLI, a *flow* represents an "active stream" of *shuttles*, being the generalized model of capsules, which integrate different services in a single module (Figure 48). The data/code tuples with the different number of red dots represent different types of protocols/services in terms of the ANTS model. Some of these services are designated to configure the receiver netbot/node for processing the next incoming tuples and shuttles in the sequence. This flow is produced from data/code tuples being multiplexed and de-multiplexed into shuttles at each end of the tunnel correspondingly. The shuttle delivery and the resource requirements should be verified with the resource availability and priority policies (incl. conflict management at the receiver side).

A shuttle flow can comprise [passive] *packets, capsules* and *shuttles*; in terms of WLI, they all are different *classes* of shuttles: P, C, and S, respectively. To recapitulate: a packet carries only data, a capsule --- (pointers to) programs and data, and a shuttle – also programs and data, whereas we distinguish two kinds of them: for shuttle processing and for netbot re-configuration.

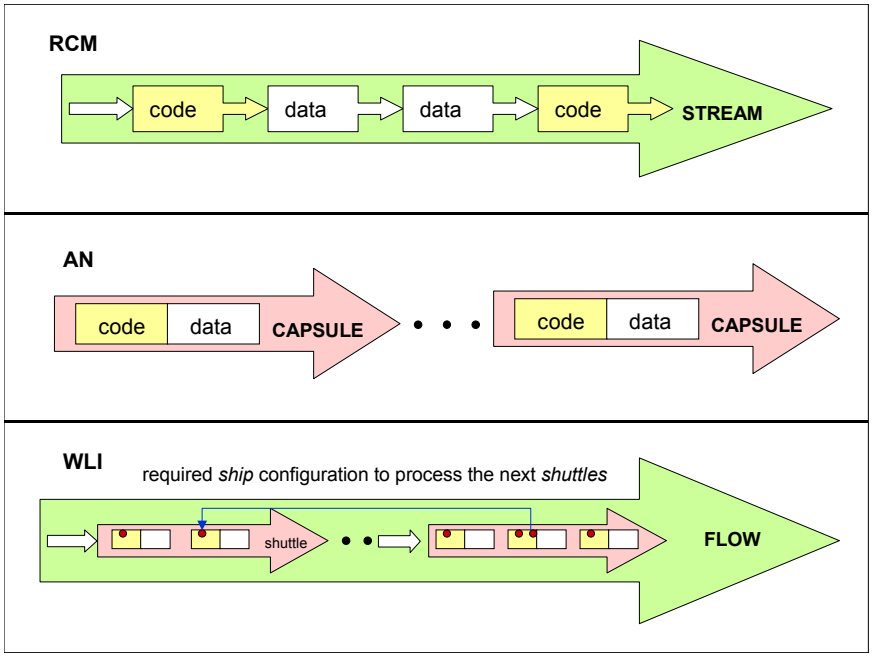


Figure 48: The WLI flow model as integration of the RCM and AN

5.2.4.4 Active Nodes and Netbots

The task of providing sufficient network protection simultaneously to the most desired component flexibility/adaptability along with a consistent view of the network itself and its resource allocation in a dynamically changeable environment is not trivial at all. Yet, the key difficulty in designing programmable networks is comprised in how to allow nodes to execute user-defined programs while preventing unwanted interactions.

The ANTS approach considers the execution of protocols within a *restricted environment* that limits their access to shared resources. Active nodes realize this by exporting an API for use by application-defined processing routines, which combine these primitives using the constructs of a general-purpose programming language (e.g. Java) rather than a more restricted model, such as layering. They also supply the resources shared between protocols and enforce constraints on how these resources may be used during protocol execution. Here we review the ANTS node design and its enhancements in WLI.

5.2.5 EXPLORING THE WLI ARCHITECTURE

The general architecture of an active node was described in chapter 3 (Active *Networks*). Then, an overview of the WLI's *netbot* architecture was given earlier in this section (The *WLI Model*). Now, we are going to describe the first level details of an active node/netbot from our viewpoint to a reconfigurable active network.

As we formulated earlier in this chapter, the basic WLI principle (Figure 44) states that the *netbot architecture reflects the shuttle structure at some previous step*[86]. Figure 49 illustrates the re-configuration of a netbot upon arrival and processing of a dedicated shuttle. This action implies a new arrangement[87] of resources within the netbot, if there are no conflicts with running processes and reservations. *Determining* which resources and parameters have to be changed to meet the shuttle/agent requirement is a different task from the one of distributing this knowledge throughout the network, say by means of the *Resource ReSerVation Protocol* (*RSVP*, IETF RFC 2205-9), to guarantee the resource availability when the transmission takes place. It is the role of the Cockpit (NodeOS) to negotiate with the shuttle configuration program and to find out the right solution, or eventually to discard the shuttle requirement. As we deal with temporal logic, we are not interested in the implementation details, but on the *goal* or the *expected* result.

Figure 50 represents a simple temporal specification of a *netlike schematic* as referred in [Sim88], [Sim89] and [Sim90]. The layout of the graphic is unimportant for the moment. What are we interested in, is the essence of the underlying mesh as a concept in time (t). We have an interconnect of four nodes (c_1-c_4) and four links (l_1-l_4) as initial state of the system at the moment T.

[86] For the moment, we exclude recursion in the WLI model.

[87] Depending on the implementation policy, the re-configuration can be limited, localized. or negotiated.

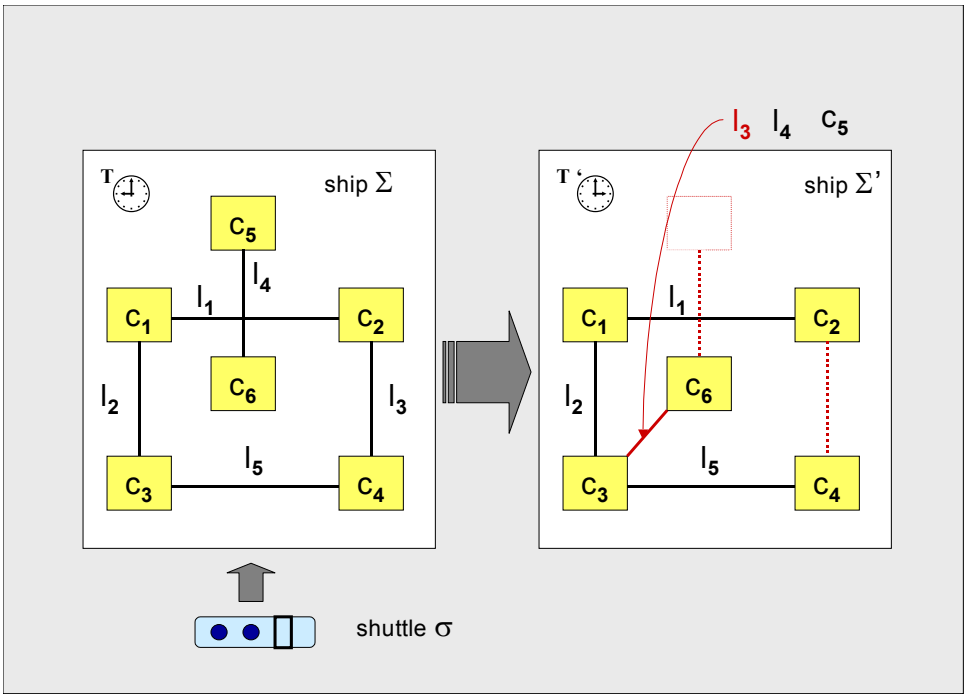


Figure 49: Changing a netbot's arrangement after arrival of a configuration shuttle.

The specification shown on the right-hand side of the image is composed of two parts: one for the nodes with the links given as parameters, and one for the links with the nodes given of parameters. The two specifications are not complementary, but *ambiguous*, since they describe the same architecture from different viewpoints. This is what we call *perceptual dualism*. The information is provided in such a form, because of *the need to be verified* (!) before being stored or acting upon it.

This kind of object specification/*recognition*[88] and verification has been thoroughly investigated in cognitive studies on 3D vision, e.g. [Hoff98]).

The change of the architecture on Figure 50 at the moment T' (e.g. in result of a shuttle arrival and processing as the one shown on Figure 49), i.e. adding the link l_5 between nodes c_3 and c_4, can be captured in a second *dual* node-link specification on the left-hand side describing only that change. This is all we need at the moment T' to recognize the new architecture, provided that we have stored and know the old one at the moment T.

[88] Vision is highly vivid, structured, rich on information, and yet easy to control in human beings ([Hoff98]). This is most probably the reason why the study of vision have been investigated with such a great interest for many years in cognitive science. Therefore, we use the imaging analogy in perception to explain our approach to describing functional *network awareness* in terms of the WLI concepts.

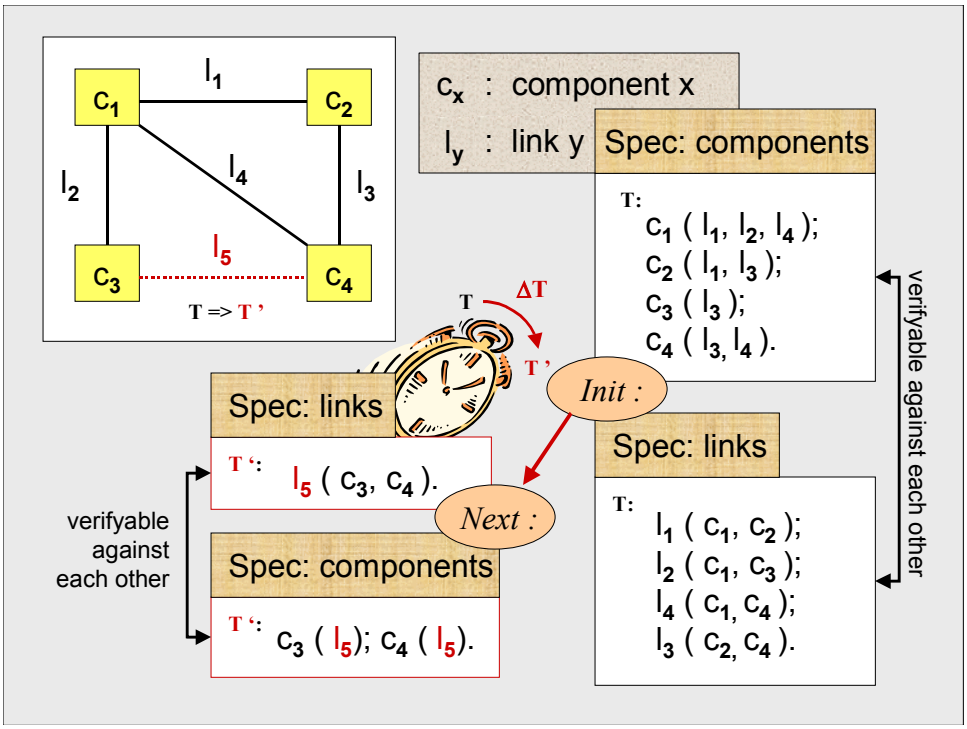


Figure 50: A simple temporal network specification

This scheme of providing □-as for differential changes in time has been used for years to describe motion images in MPEG format. The benefit of the WLI model is that e.g. visual content information, such as image encoding and decoding is only one particular service for conveying information by means of packets/capsules. What we can do more with capsules/shuttles in a configurable active network is that we not only record a “still image” of the desired change/re-configuration in an active node/netbot, but also an entire sequence or even causal branches of sequences of changes to be invoked in that node/netbot. This is what we regard as a *programmable network*.

Finally, the WLI model allows the generation of a “*virtual*” active node/netbot, a logical network element that emerges by reserving resources from neighboring network elements as shown on Figure 51. This may be the result of a node/netbot re-configuration upon arrival and processing of shuttles or of other *out-of-band* signaling mechanisms (protocols) between the neighboring structures. The virtual node/netbot could be also a temporal characteristic of the active network, which is desirable upon request.

In the next chapter, we will discuss the application of the WLI technique in the area of wireless networking in more detail.

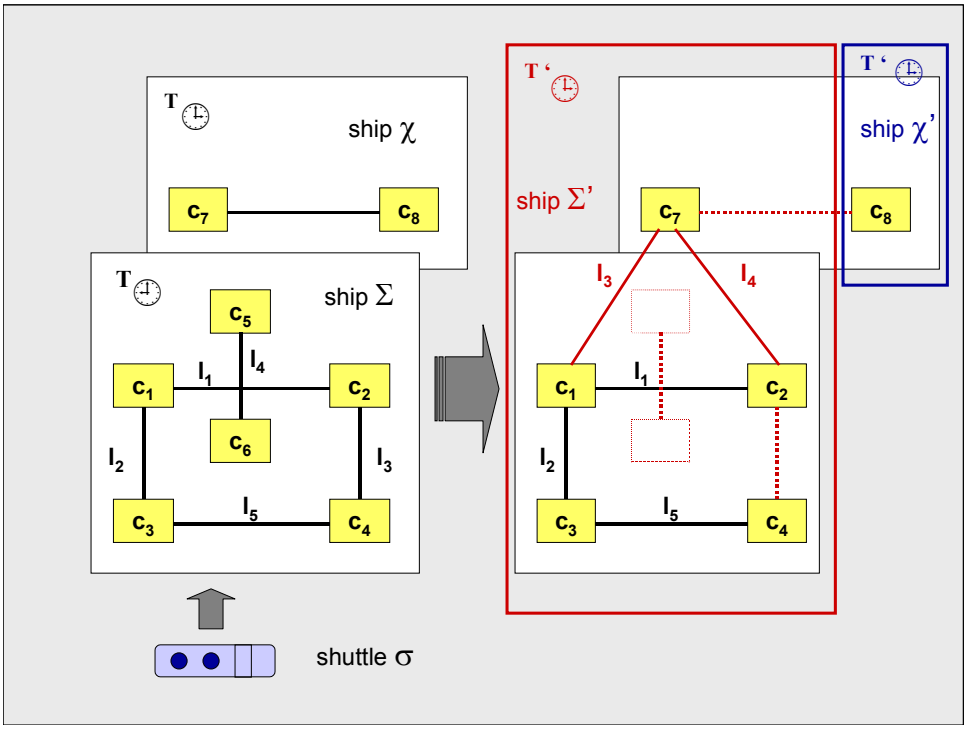


Figure 51: Configuring a virtual active node/netbot upon shuttle request.

For the time being, it is important to note that in the case of *in-band* signaling for re-configuring active nodes/netbots via active packets/shuttles, it is indifferent (from viewpoint of temporal logic and WLI) whether the netbot/node *activates* a link to an existing resource on its own corpus, or it borrows that resource from a neighboring node/netbot and establishes a link to it. In terms of WLI, the netbot Σ on Figure 51 appears to be turned into the netbot Σ' at the moment T' upon arrival and processing of the shuttle σ.

Herewith, the netbot Σ' is a *virtual* one generated from the netbot Σ (at the moment T) by "removing" two of its components (c_5 and c_6) and two of its links (l_3 and l_4) and by borrowing another component (c_7) from the netbot χ and establishing two links to it from the components c_1 and c_2; the previously removed link identifiers (l_3 and l_4) are reused.

There are two mechanisms to produce a new module in WLI: a) *by inclusion* from a neighboring netbot, and b) *by installation* of software in an existing resource of the netbot or through switching hardware to another configuration of the netbot. In WLI/TLA we formulate this result as producing the virtual netbot Σ' from the netbot Σ upon processing of the shuttle σ. This is what is necessary to be specified in order to be able to prove whether certain properties of the system satisfy some desired requirements.

5.3 THE WANDERING NETWORK PRINCIPLES

The scientific contribution of this work is to propose and demonstrate a simple and flexible mechanism for network evolution based on the emergence, the change and the movement of functional units within a given physical infrastructure, which recognizes its own boundaries. Such a network is known as an *autopoietic system*. The following definition is closely related to the one given by Maturana and Varela in [MaVa80] and revised in [Mat00].

> **WLI Definition 5:** A Wandering Network (WN) is a dynamic composite entity realized as a unity of a closed set of *productions* of mobile nodes, called *netbots,* such that through their interactions in composition and decomposition (programming, adaptation, configuration, reflection, etc.) at all functional levels they define the network as *self-creating,* i.e. as an *autopoietic system* by:
>
> - *recursive constitution* of the same system of productions that produced the netbots and their communication patterns, and
> - specification of the network extension in terms of its commuting components defined and its boundaries determined by the end-users as a dynamic entity apart from the surrounding environment, invoking the desired changes in the information infrastructure.

The WLI model is based on four general principles, [Sim02a]:

1. Dualistic Congruence
2. Self-Reference
3. Multidimensional Feedback
4. Pulsating Metamorphosis

In the following four sections, we introduce these principles as fundamental frame of the Wandering Network. The Dualistic Congruence describes the kernel property of a WLI architecture. The next two principles are broadly used in modern software/hardware system design and network engineering. They are adopted and generalized for the purpose of this work from published research in the areas of active networking, configurable computing and adaptive systems. The fourth principle is closely related to advances in natural sciences and in particular to concepts and ideas in neurobiology and biophysics.

5.3.1 THE DUALISTIC CONGRUENCE PRINCIPLE (DCP)

This principle was already addressed earlier in this chapter in the WLI definition. The Wandering Logic model is based on: a) the dual nature of the *ploions*, the active [mobile] network component abstractions in their two manifestations, *netbot* (active mobile nodes) and *shuttles* (active gene-coded packets), and b) on their congruence.

The Dualistic Congruence Principle states that *a netbot's architecture reflects the shuttle's structure at some previous step and vice versa.* Figure 44 illustrates these **mutual implications** of the netbot-shuttle behaviors expressed in temporal logic.

Thus, *netbots* are both reconfigurable computing machines and active mobile nodes in terms of hardware and software. Shuttles transport software which can activate / replace netbots and their components/aggregates. A netbot processing shuttles can change its state and re-configure its resources and connections *a posteriori* for further actions. In addition, it can adapt (itself) *a priori* to communications in such a way that it *best-match the structure of the active packets (shuttles) at the time of delivery.* Finally, a netbot can also change the state of a shuttle.

*Shuttles***,** in turn, can be e.g. interpreted by a reconfigurable computing element inside a netbot to build and/or invoke new functions. A shuttle approaching a netbot can *re-configure itself* becoming a *morphing* packet to provide the desired interface and match a netbot's requirements. This operation can be e.g. based on the destination address and on the class of the netbot included in this address.

5.3.2 THE SELF-REFERENCE PRINCIPLE (SRP)

> **WLI Definition 6:** The following characteristics identify a wandering network as *self-referring*:
>
> 1. Mobile nodes, *netbots*, are living entities: they can be born, live and die. Netbots can also organize themselves into clusters based on one or more *feedback* mechanisms. Communication between the netbots is realized through exchanging programs and data by means of *shuttles,* active packets, which may also contain encoded structural information about the netbots or parts of the network itself. The structural information can be used to maintain the operation of the network as a whole, as well as to invoke desired or necessary changes in the infrastructure through service utilization and components' feedback.
>
> 2. Each netbot knows best its own architecture and function, as well as *how* and *when* to display it to the external world. Netbots are required to be *fair* and *cooperative* w. r .t. the information they display to the external world; otherwise they is excluded from the network community.
>
> 3. Each netbot can acquire or *learn* some other function and extend its architecture by some additional functional components in software or hardware, as well as to become a (temporary) aggregation (a cluster) of other nodes with a joint architecture and functionality.

The Self-Reference Principle addresses the *autopoiesis* and autonomy properties of the AN elements.

5.3.3 THE MULTIDIMENSIONAL FEEDBACK PRINCIPLE (MFP)

The *feedback principle* in network engineering is well known in protocol design for applications such as traffic control. However, not all degrees of freedom have been exploited until now. Active networking introduces a new paradigm for this mechanism, which can be spread out to *any* service, device and application in a communicating environment.

Active networking introduces a new paradigm for this regulation mechanism, which can be spread out to *any* service, device and application in a communicating environment. The reason is that the network offers much better opportunities to address traffic issues on a *per-service* basis than the terminal devices alone.

For instance, an application for facility management such as gas pipeline monitoring allows each user to see composite images constructed by fusing information obtained from a large number of sensors via autonomous mobile "web" cameras over a wireless network. Each sensor in the network can be observed by a number of users, who will have different requirements concerning the encoding and presentation of the information they access. What should be provided in this case first is a set of core-differentiated services on a *per-user* and *per-flow* base for feedback-enabled monitoring and traffic adjustment for QoS provisioning in real-time, Figure 52.

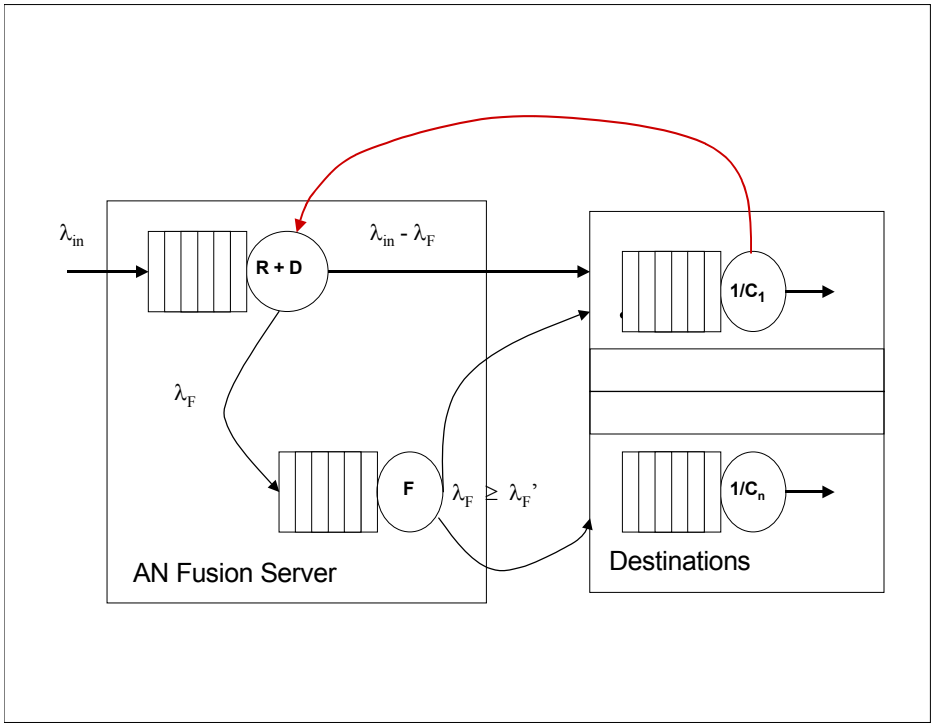


Figure 52: The feedback principle: using an active network fusion server for traffic control

This actually corresponds to a dynamic change (re-configuration), - in fact, a programmability and *adaptability* (as means) to ensure *dependability*[89] (the reason) -, of the network topology and resources in **multiple dimensions**.

An active network provides a couple of means for such a solution. Here is where the multiple dimensions come from.

[89] This is a generalized concept and thus differs from definitions given in the area of fault tollerance.

For instance, the AN Fusion Server on Figure 52 can be *enabled* anywhere within the network. Since, each active node controls its own resources, this implies a manipulation of the traffic on a *per-(active)-node* and a *per-configuration* basis. Then, an active packet may contain some network, user or application related data starting from look-up tables and personal configurations, and ending with programs such as encoders, compilers and even compiler-compilers to be mounted on the destination node: the *per-(active)-packet* and *per-method* dimensions.

Furthermore, merging data within the network reduces the bandwidth requirements of the users' who are located at its (low-bandwidth) periphery. Also, user-specific multicast services within the network reduce the load on the sensors and the network backbone. Therefore, a traffic adaptation on a *per-multicast* branch base is also possible.

In addition, the routers and switches of an active network perform customized computations on the messages flowing through them: the *per-message* dimension. For example, the operator of an active network could send a trace program to each router and arrange for the program to be executed under certain conditions when their packets are processed.

Besides, active routers could also interoperate with legacy routers, which transparently forward datagrams in the traditional manner. Addressing subsets of legacy routers for interactions defines another dimension, the *per-interoperability-task* one.

Finally, the traffic processing can be customized via a set of differentiated auxiliary services on a *per-application, per-session* and even on a *per-data-link* basis in terms of OSI.

5.3.4 THE PULSATING METAMORPHOSIS PRINCIPLE (PMP)

We call the generic process of network self-creation and self-organization the *Pulsating Metamorphosis Principle* (PMP), Figure 53.

> **WLI Definition 7**: The evolution of the Wandering Network is determined by the *Pulsating Metamorphosis Principle* (PMP), stating that:
>
> 1. There are two types of moving network functionality from the center to the periphery and vice versa inside a Wandering Network, which are referred to as *pulsating metamorphosis*: horizontal, or inter-node, and vertical, or intra-node, transition[90].
> 2. A net function can be based on one or more facts (events, experiences). The combination of net function and facts is called a *knowledge quantum* (*kq)* in the WLI model. Knowledge quanta are a new type of capsules, which are distributed via shuttles in the Wandering Network. Net functions and facts can be recorded by, stored in and transmitted between the netbots. They can be selectively processed inside the netbots and distributed throughout the Wandering Network (WN) in an arbitrary manner.

[90] Figure 56 and Figure 57 respectively.

3. A net function can emerge on its own (the *autopoiesis* principle) by getting in touch with other net functions (i.e. states and net constellations), facts, user interactions or other transmitted information. *The function defines the network and vice versa.* This new property of the network is called *network resonance.*

4. Network elements are *living* entities. They can encode and decode their state in *knowledge quanta*. This mechanism is called *genetic transcoding.*

5. Facts have a certain *lifetime* in the Wandering Network. This lifetime depends from the clustering of facts inside the netbots (knowledge base), as well as from their transmission intensity, or bandwidth (known as "weight"). As soon as a fact does not reach its frequency threshold, it is deleted to leave space for new facts. Since net functions are based on facts, their lifetime (and hence, the life time of the corresponding network constellations) depends on the facts. Which facts determine the presence of a particular function inside the Wandering Network is defined individually for each function. Through the exchange and generation of new facts, it is possible to modify functions in order to prolong their lifetime. The lifetime of a knowledge quantum is defined by the lifetime of its network function. A modification of a net function is determined by a new set of knowledge quanta.

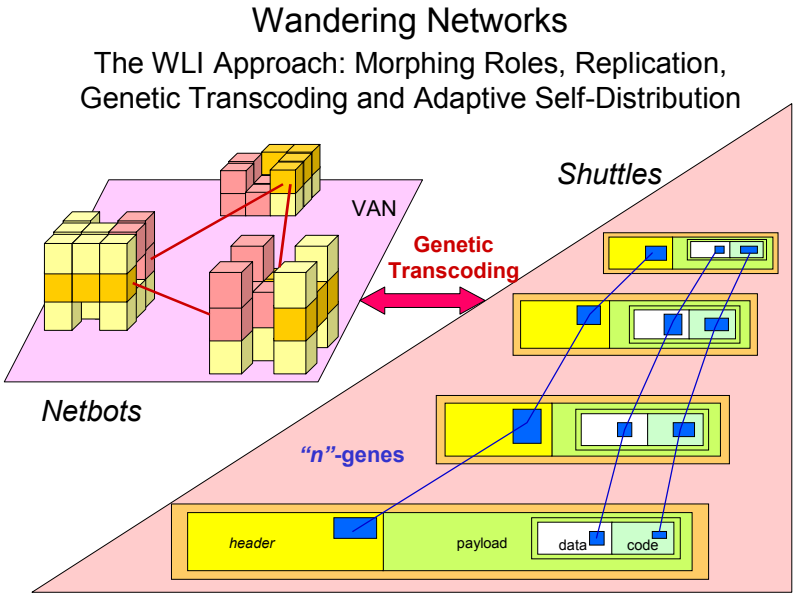


Figure 53: The Wandering Network as an "n"-geneered evolution

The network *resonance*[91] is the leading WLI characteristic. It can be regarded as a kind of adaptive meta-policy for network development.

[91] The analogy of this special property of the Wandering Network is known in the biology as the Sheldrake's theory of the morphing resonance, [Shel81].

With its help, clusters and constellations of network elements or their functions can be (self-) correlated and/or (self-) organized in groups, classes and patterns and stored in the cache of the single nodes/netbots or in the (centralized) long term memory of the network, in order to be used later as a decision base or as a development programs for particular processes in the network (e.g. service location, customer care, billing, etc.).

The WLI Definitions 4-7 describe the concept of the Wandering Network.

The WLI idea differs from the current research approaches for network evolution known as Open Signaling and Active/Programmable Networks.

The above four principles define the overall concept framework of the Wandering Network, [Sim01], [Sim02a]. In the following, we will discuss the impact of this approach on future network architectures.

5.4 YET ANOTHER NETWORK-NETWORK

Now, how shall we induce more activity in a Wandering Network ? For instance, we could allocate different netbot or active node classes depending on their role (function) within the network. For instance, Wetherall and Tennenhouse [WeTe96] define 4 basic classes of capsule mechanisms or network functions :

1. ***Fusion*** : the active node is delivering *less* data than it receives e.g. filtering of an MPEG-4 video stream content.

2. ***Fission*** : the active node is delivering more data than it receives, e.g. generating additional packets for multicasting

3. ***Caching*** : the active node *stores* incoming data for later use upon request, e.g. storage of web pages for local processing and reducing the data flow

4. ***Delegation***[92]: the active node is performing tasks on behalf of another active node which are delegated by means of capsules, e.g. becoming a unified messaging node which migrates closer to a nomadic user while she moves

The WLI approach extends this role framework with the concept of (re-)configurable and programmable *functional specialization* of the node both in terms of hardware and software (Figure 54). To retain the simplicity of the WLI model[93], we postulate that each active node (or *netbot*) can be assigned exactly one single function at a time. Thus, an active node could behave e.g. as a fusion server during a session, and then to obtain the assignment of becoming a network cache proxy for another session.

[92] **Note**: the active node configiration itself can be delegated, Figure 54.

[93] for a general view on the node structure, please refer to section.

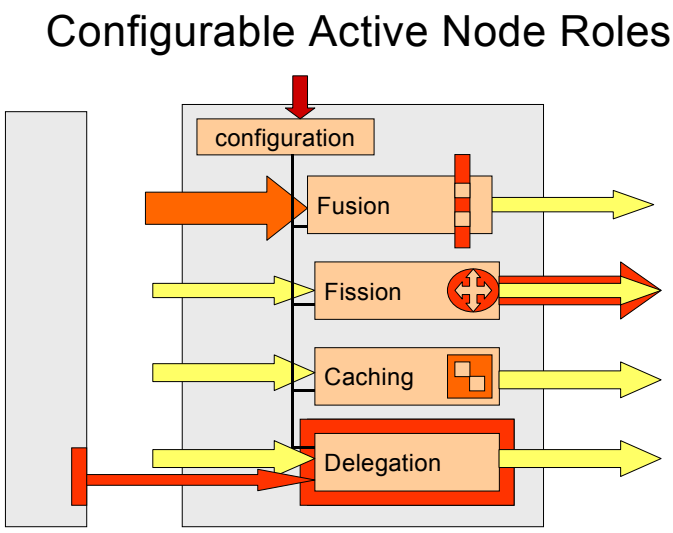


Figure 54: Multiple AN functions

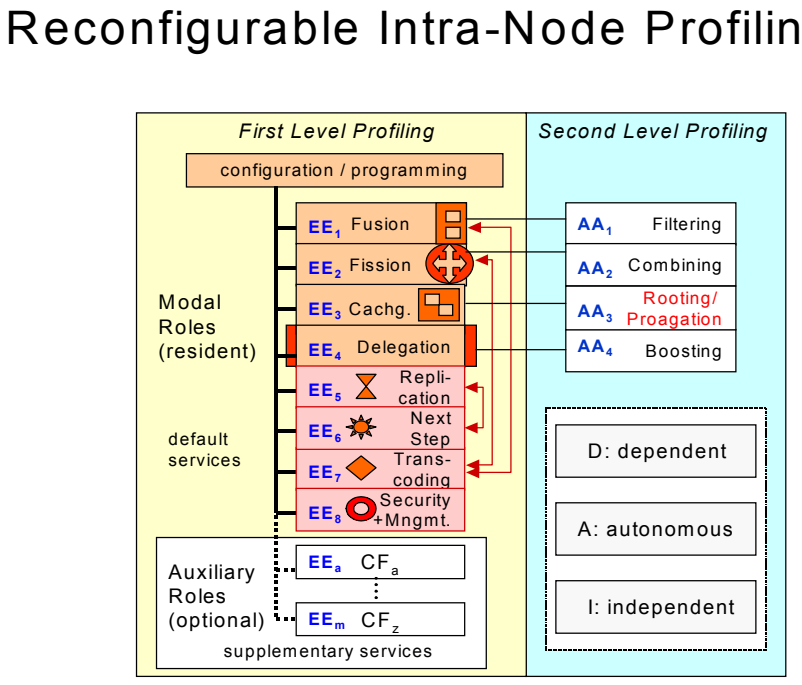


Figure 55: A netbot's internal functional organization

We distinguish between *modal* (basic) functions resident at each node/netbot and *auxiliary* (optional) ones that can be transported, installed and enabled via capsules/shuttles to be later customized by the user. By default, we consider that each function is assigned a single "registry" execution environment (EE) with the modal functions being priorized for access.

Kulkarni and Minden, [KuMi99], propose seven *classes of protocols* in active networks, some of which could be regarded as specific instantiations of the ANTS capsule mechanism classes:

1. ***Filtering*** (fusion): packet dropping or some other kind of bandwidth reduction technique;
2. ***Combining*** : (fission): joining packets from the same stream or from different streams;
3. ***Transcoding*** : transforming user data / content into another form;
4. ***Security Management*** : capsule authorization and resource access control;
5. ***Network Management*** *:* self-configuration, self-diagnosis, self-healing via event reporting, accounting, configuration management and workload monitoring;
6. ***Routing Control*** *:* overlaying and managing several virtual topologies on top of the same physical network infrastructure as an application-layer service;
7. ***Supplementary Services*** *: adding new feature to the packets* without altering, but depending on their contents, e.g. content-based buffering.

WLI regards the above two classification schemes, with exception of the routing control, as a *horizontal inter-node functional wandering (self-organization)* of the active nodes (netbots), Figure 56. We call the capsule mechanisms (functions) identified by Wetherall and Tennenhouse "First Level Profiling", and the protocol classes (functions) of Kulkarni and Minden – "Second Level Profiling".

In WLI routing control is considered as a special class of *virtual vertical intra-node overlay functional wandering (self-organization)* which is interdependent from all of the other functional classes (node roles), Figure 57. For instance, we can generate a QoS oriented network topology on demand.

The two schemes of functional autopoiesis, horizontal inter-node and vertical intra-node wandering are operating in parallel to realize an adaptive virtual topology by utilizing the pulsating metamorphosis principle (PMP) which opens a new dimension of hyperactivity in networking.

In order to address the performance enhancements, we included the protocol boosters as an additional class to the categorization of Kultarni and Minden along with an instantiation of the delegation mechanism of Wetherall and Tennenhouse. Furthermore, we combined the security and network management classes into one single class. Finally, we assigned two additional roles to the First Level Profiling: *Replication* and *Next-Step* for packet / function replication and netbot state description respectively.

To complete the model, routing and propagation of functionality were included in the Second Level Profiling as dependants of the caching class which refers in turn as a bootstrapping mechanism to the node state (Next Step) and all other instances of the functional classes in the First Level Profiling. Figure 55 illustrates the netbot organization according to this scheme.

The first two roles of the First Level Profiling correspond partially to the functions “Forward and Copy” (FaC) and “Oracle” suggested by Raz and Shavitt, [[RaSh00] to enhance the AN architecture framework. A capsule/shuttle replication could be quite useful for deploying knowledge-based services such as selective “activation” of the network topology and thus adding an additional level of flexibility to the AN model (e.g. to change a node’s routing algorithm and/or table).

The Next-Step function operates as an internal programmable switch which stores the next node role to come. It is a standard module for each node/netbot. Since most of the network traffic carries large amounts of rich multimedia content, a *transcoding* function for congestion control and local, feedback-enabled content-, user- and resource-dependent QoS management appears to be also useful.

The new model provides a unified and structural approach for a flexible intelligent network of the new generation. This solution is not only applicable for out-band signaling „intelligent networks“, but also for the new generation of the so-called ”programmable“, active networks where extensions, new services and new versions can be easily installed and configured in a *usability* driven manner.

5.5 RELATED WORK

The following three examples should illustrate the author’s contribution and the new qualities of Wandering Logic approach [SiRe02a], [SiRe02b].

Example 1: Servents. The AS1 approach (section 4.3.5.2) to programmable Active Services [AMK99] enables operators (but not end customers) to download and run service agents (*servents*) at strategic locations inside the network. All routing and forwarding semantics of the present day Internet are preserved by restricting the computation model to the application layer. AS1 supports a range of application domains such as active media gateway services where servents perform application-level rate control and transcoding techniques.

Servents can be regarded as a mobility extension of the SIBB (Service Independent Building Blocks) concept in Intelligent Networks towards next generation customizable architectures. However, restricting computation to the application layer only, creates a substantional management overhead even if only a few custom services are required.

> *The AS1 architecture is complex and under control of the network operator only. It represents a complete service creation and distribution factory and contains such utilities as a service definition environment, a service-location facility, a service resource management system, a service client dynamic control system, and a service composition mechanism. Servents are not aware of the underlying hardware topology. They are not autonomous. AS1 is not capable to dynamically track or reflect end user behavior, “on-demand” or threshold-related, and cannot guarantee optimal utilization of network resources. The active services cannot be automatically switched off or removed to release node resources.*

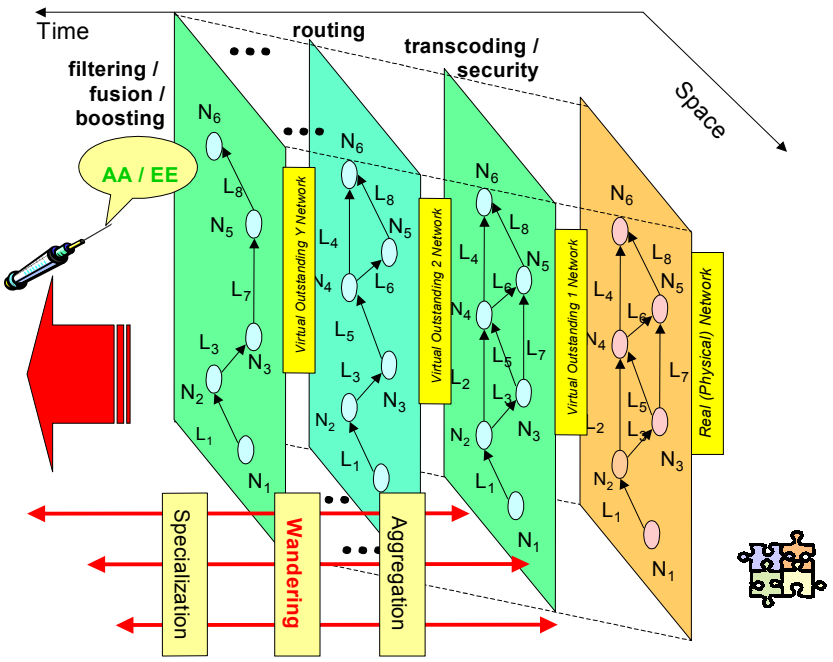


Figure 56: Horizontal network wandering (ex-pulsing) - *inter*-node functional autopoiesis generated virtual *outstanding* networks of the same physical infrastructure

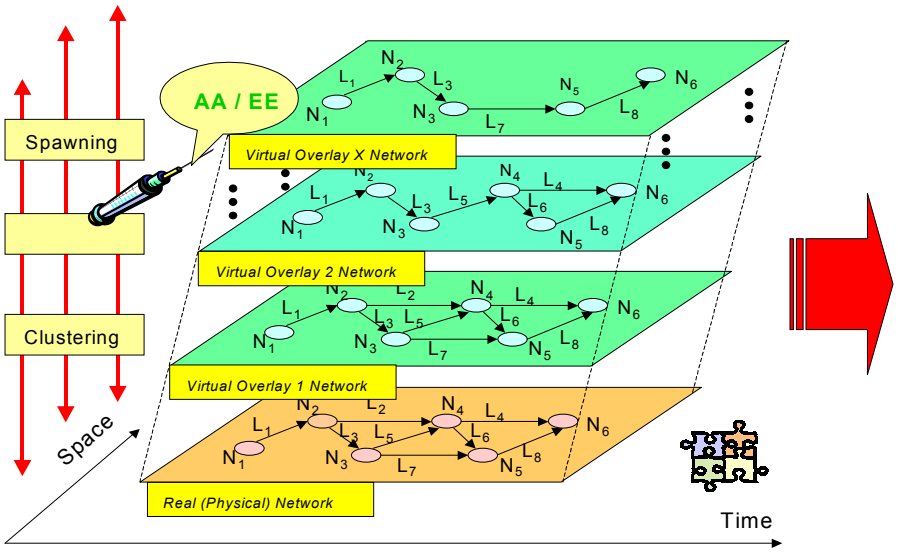


Figure 57: Vertical network wandering (in-pulsing) – *intra*-node functional autopoiesis and generated virtual *overlay* networks over the same physical infrastructure

From our viewpoint, servents belong to the second level profiling class of the application layer horizontal inter-node functional wandering under operator control. They represent a part of the WLI architecture model which is not self-regulating yet.

Example 2: Switchlets. Switchlets represent the middle layer of the SwitchWare architecture (3.3.3.1, [Alex98a], [GNS98]). They provide base functionality or dynamic extensions to active IP routers to guarantee security on an as needed basis. Switchlets can be dynamically loaded across the network, but execute entirely on a particular router. In combination with active packets, they can implement arbitrary protocols or functionality.

The Tempest approach (3.3.4.2, [Merw97b]) regards switchlets as logical network elements resulting from the partition of ATM switch resources to allow the introduction of alternative control architectures into an operational network. Each switchlet has a well-defined open programmable interface for dynamic updates as lightweight services.

> *The above definitions of switchlets address flexible and trade-off capable architecture models for the underlying hardware and network infrastructure. However, these models are still under the "manual" control of the operator. Thus, their programmability reflects the way of how this operator perceives the network and its development, but not necessarily the ones of the different user classes. Although being capable to flexibly utilizing network resources, they cannot predict, follow and dynamically adapt to user behavior patterns.*
>
> *Switchlets belong to the network layer horizontal inter-node functional wandering. However, if a particular switchlet function is a) either replaced for some reason (performance, fault, etc.) by a lower layer reconfigurable hardware component (e.g. a DSP transcoder chip, which is not the case in current architectures), or b) replicated and transported to an upper application layer (executing environment) in order to be then transmitted to another node as a servent, we speak of an upwards vertical functional wandering. Of course, the "downloading" of this network function in the destination node through an application layer gateway is called downwards functional wandering. This is a typical capability and unique property of the Wandering Network (the Pulsating Metamorphosis Principle) compared to other active and programmable architectures.*

.

Example 3: Routelets. The Genesis approach ([Camp99b], [Camp01]) introduced the routelet, the open programmable virtual router node, a basic abstraction of a spawning network architecture. Routelets represent the transport environment, the lowest-level operating system support dedicated to a virtual network. A virtual network is defined by a set of routelet interconnected by a set of virtual links. While routelets process packets along a programmable data path at the internetworking layer, control algorithms (e.g., routing and resource reservation) are considered programmable using the virtual network kernel. Thus, the transport environment represents a programmable data path at a router. Genesis routers are capable of supporting multiple routelets, which are components of distinct virtual networks that share computational and communication resources.

Although spawning networks created the notion of "nested (children) virtual networks" within a virtual network which are capable a) to inherit properties of the parent network(s) and b) to implement a set of overlay protocols on demand[94], they are still not "lively pulsating" like a Wandering Network. The Genesis kernel lets virtual networks spawn and grow in one direction only, from parents to children, but not across the network.

For instance, the spawning mechanism can expand a wired customer network to support an extra set of wireless protocols for these customers, but it cannot automatically "zip-and-pack" itself (and the routelets, of course) to another geographical site even for a single nomadic customer (who may ask for the same virtual home network environment) where the appropriate hardware infrastructure already exists. Routelets can be generated, clustered and programmed to provide the functionality of a virtual router, but they cannot be "unwrapped", redistributed or selectively destroyed following a simple rule, e.g. a service usability threshold. Despite representing a further step towards network virtualization and growth, they do not provide any evolutionary mechanism for self-configuration and self-distribution.

Finally, even being programmable entities, servents, switchlets and routelets do not necessarily implement interdependent feedback signaling mechanisms to adapt per se to some changing condition of the communicating environment.

Although they provide some degree of function replacement and mobility within the network, these "classic" AN abstractions are not based on a particular self-regulation principle, such as the Dualistic Congruence, Self-Reference or Pulsating Metamorphosis, which belong to the foundations of the Wandering Network.

The Wandering Logic Intelligence is a unifying, and though, a new, evolutionary network design concept based on the *four principles of self-organization* defined in this thesis (Section 5.3).

Almost any network application from the extensive list discussed in section 3.4 (firewalls, web proxies, nomadic routers, transport gateways, application servers, etc., Table 1) can be realized in a more vivid, flexible and sophisticated fashion following the WLI paradigm, e.g. by mobilizing and programming the modal and/or auxiliary roles of the reconfigurable intra-node profiling architecture on Figure 55. Presenting a realization scenario for each one of the above applications is beyond the scope of this work. However**, detailed discussions of the WARAAN algorithm, a special case study within our target domain (wireless networks) for applying the WLI model to autopoietic routing in active ad-hoc mobile networks is presented in chapter 6.**

[94] e.g. at the boundary between different network infrastructures which is the case of an internationally roaming mobile professional user

5.6 CONCLUSIONS

Computing and networking are two models of the human brain, and thus of intelligence per se, related to *processing* and *distribution* of information. In fact, they are two perspectives (micro and macro) of the same concept.

The WLI hypothesis states that we can understand (and possibly optimize and improve) computing and communication processes by applying the analytical approach known in natural sciences of including further details of the "black box", the network elements, into the information exchange model (the protocol, the capsules, the shuttles). We claim that the deepest nature of Information (processing) is Communication (development) and this dualism is manifested in the *wandering* essence of intelligence per se, the *WLI*.

Now, let us consider the "*active mess"* as the extreme case of applying all known active network approaches to the existing legacy infrastructure and standard communication protocols simultaneously in a single network (the Internet!). Then, let us also apply the 'surgery' requirement that network resources can be configured, used and managed to the deepest possible layer, under consideration of security and safety constraints (!), and *on customer request* by means of active networking and configurable computing.

One of the basic principles of active networking was that it should facilitate the rapid introduction of new services and applications without the interference of standardization. How shall we recognize the different shuttle flows, protocols along with the occupied and free resources? Where are the limits of this integration of software and hardware?

We hope that the WLI hypothesis will not explode into defining a whole network cosmology. On the contrary, we believe that understanding networking is quite simple (by its nature) and that it is sufficient to apply only a few of the available numerous options for implementing a characteristic in order to obtain the desired result. For instance, a shuttle differentiation policy can be based on such a simple principle[95] as the code division mechanism used in CDMA. To us, this appears to represent a much more general *low* as the ones in natural sciences (of course, within a certain application area).

The same might be the case with such network technologies as ATM, OFDM, etc. They could be organized as principles of, say, a *General Networking Theory*[96]. However, in order to explore this in a scientific manner, we need the apparatus, the math, the logic, the calculus or at least the hypothesis of "What could be *out there?*"*.* All we need to come closer to the desired result are the means of: a) writing down "how-all-this-may-work-together", and b) examining the above hypothesis with the selected means in a stepwise, scientific manner, plus c) changing the means, if they are not found of appropriate. Upon this, we can design a prototype from the model and test is it works well.

---

[95] We are not going to discuss further details of the WLI model in this dissertation, except the ones described in the implementation part. We regard the development of WLI policies as a free research field.

[96] The shift of networking from an engineering discipline into a real science is probably not so far away.

We believe that we can start here with temporal logic, and then extend this formalism whenever found for appropriate. If we miss the proper tools, they can be developed in parallel to the WLI methodology …

A configurable network is characterized by both node and packet *programmability* and *adaptability*. In addition, active packets transport *mobile code*.

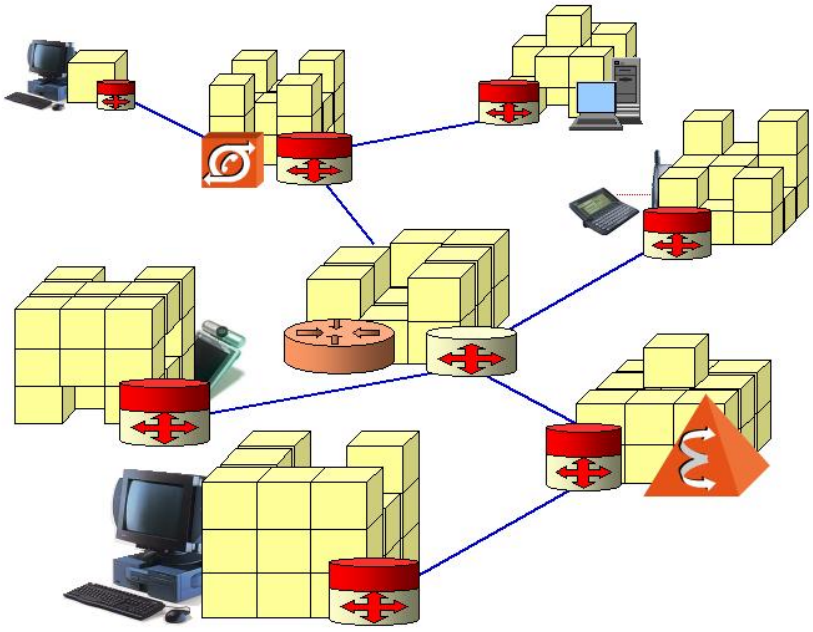


Figure 58: The Wandering Network as an ad-hoc network evolution

In this work, we postulate that *all nodes can be also mobile*, e.g. as mobile platforms such as vehicles or wearable user equipment[97]. In other words, the WLI domain comprises active mobile ad-hoc networking. Chapter 6 is devoted to a case study in this area. Figure 58 illustrates a snapshot of such an "always being under construction" Wandering Network, where the different shapes of the nodes represent different functionalities at a given moment. The formal approach requires that all these properties of the active system be described in a formal way, such as a calculus or logic. In addition, the formal language should support generic design features such as openness, distribution and object-orientation.

* * *

[97] User equipment (CPE) does not necessarily refer to network terminals; it may be configured to *route* or *serve in behalf* of some other user in the neighborhood.

CHAPTER 6: CASE STUDY – WLI ACTIVE AD-HOC MOBILE ROUTING

"Felsen sollten nicht Felsen und Wüsten Wüsten nicht bleiben ..."

J. WOLFGANG VON GOETHE
SEPT. 6TH, 1776, HERMANNSTEIN / ILMENAU

6.1 SCOPE AND MOTIVATION

The goal of this thesis is to provide a generic design methodology, referred to as the *Wandering Logic Intelligence (WLI),* for reasoning about autonomous networked systems. Ultimately, the methodology is aiming to deliver a *formal*[98] recursive design model of the discourse domain, further referred to as the *Wandering Network* [Sim01], which has been defined as a superset of the worlds of evolving active networking and configurable computing viewed from the perspective of biological *autopoietic* systems ([Mat75], [Var79], [MaVa80]).

However, it is not the task of this work to deliver the complete proof the Wandering Network model. This thesis' main goal is a) to identify the major design principles in active networking and configurable computing and b) to demonstrate their application in a methodologically sound way to a practical problem in the area of network engineering.

Chapter 2 reviewed the various characteristics of Active Networks and noticed that the scope of their application is very broad, starting from network management through dynamic caching and multimedia filtering to spawning networks.

The two basic claims of deploying Active Networks in network design are:

- *code mobility*: free choice of *where* and *how* to perform *packet processing;*
- *state mobility*: free choice of *where* and *how* to place *states* within the network.

In short, AN is about remote programming and re-configuration of network nodes to deploy non-standard protocols and coding techniques. However, Active Networking as a design concept should not be only limited to programming the single network nodes. It carries a much greater potential of being able to create an overall life-cycle system design methodology, which can have an enormous impact on future network generations.

This thesis claims that the AN approach can be further extended with the WLI methodological framework[99] by monitoring spontaneous changes in network topology and node behaviour (functionality) in a way which can enable the self-deployment, [Tschu99c], and *self-organization* and *re-production* [Min94] of the entire network architecture inside and outside the nodes.

[98] in the sense that we want to prove mathematical theorems about the behaviour of artificial self-maintaining systems.

[99] a step-by-step life cycle of *(re-)design-verification-deployment-feedback*.

As we already noticed in Chapter 2, WLI is treated in this work as a system oriented design methodology, whereas Active Networking and Reconfigurable Computing -- as enabling technologies. We hope that understanding the WLI concept in terms of an evolving formal design methodology will help moving research efforts towards a self-maintaining and reproductive active network architecture.

The subject of this chapter is the description of a suitable protocol and architecture for mobile ad-hoc networks based on virtual active topologies, which allow adaptable behavior of the network due to node mobility. It is assumed that there exists a channel access protocol withholding disruption and interference problems in the network. In the method that we propose, network nodes are allocated addresses depending on their physical connectivity and address availability. In general, each node can be assigned a single address. However, in some cases nodes (netbots) may have more than one address.

6.2 CONSTRUCTIVE BACKGROUND: ROUTING IN MOBILE NETWORKS

The next section provides some helpful definitions used later in this chapter.

6.2.1 MOBILE DEFINITIONS

6.2.1.1. Mobile Network

A mobile network is a network in which some of its nodes (endpoints and/or routers/switches) change location relative to each other. The ends or leaves of the network, called mobile terminals (MT) or mobile hosts (MH) may move among stationary (fixed) routers/switches (R/S). On the other hand, router/switches may move over stationary endpoints (e.g. in satellite networks), or both MTs and R/Ss may move independently (e.g. in packet radio networks).

Depending on the mobility of the ends and intermediate nodes of the network, we distinguish between four general classes of networks, three of which are said to be mobile:

Interim. / End	Fixed Terminals	Mobile Terminals
Fixed R/Ss	Wire-line	Cellular
Mobile R/Ss	Satellite	Packet Radio / Ad-hoc Mobile

The different requirements of these types of networks are reflected in the different organizations of their topologies and functions. In particular, we are interested in packet radio or ad-hoc mobile networks where both R/Ss and MTs are mobile.

A network of mobile wireless switches that employ radio communications is generally called *packet radio network*[100] (e.g. DARPA PRNET [KGBK78], [JuTo87] SURAN [ShWe87]). Tactical military communications require survivable, adaptive networking where R/S mobility is an important advantage. Recently, there has been a growing interest on civil networks of mobile switches[101] [PeBh94]. Such networks are referred to as *ad-hoc*[102] mobile networks and are conceptually identical with packet radio networks.

6.2.1.2 Ad-Hoc[103] Mobile Network

Currently, there are two kinds of mobile wireless networks. The first one is known as *infrastructured networks*. These are those networks with fixed and wired gateways. The bridges for these networks are known as base stations. A mobile unit within these networks is connected to and communicates with the nearest base station that is within its communication radius. As the mobile travels out of range of one base station and into the range of another, a handoff occurs from the old base station to the new one and the mobile is able to continue communication seamlessly throughout the entire network. Typical applications of this type of networks include office wireless local area networks, GSM networks and recently, GPRS and UMTS networks. The second type of mobile wireless network is the *infrastructureless mobile network*, commonly known as an *ad-hoc network*.

Infrastructureless networks have no fixed routers. All nodes are capable of movement and can be connected dynamically in an arbitrary manner. Nodes and of these networks function as routers which discover and maintain routes to other nodes of the network. Example applications of ad-hoc networks are emergency services and rescue operations, meetings or conventions in which persons wish to share information and interposition operations in inhospitable environments quickly. In the rest of this work, we are going to use the following definition:

> **Mobile Definition 1:** An ad-hoc network is a dynamically reconfigurable wireless mobile network with no fixed architecture or central point of administration. It represents a radically distributed architecture with the following characteristics:
>
> - all nodes/hosts can be deployed as clients and as servers.
> - each node/host is mobile and acting as a router/switch.
> - nodes/hosts and relationships/routes between them are of temporary nature; they can emerge and disappear spontaneously in the network.

Ad-hoc mobile networks are deployed in applications such as disaster recovery and distributed collaborative computing.

Mobility of routers/switches (R/S) raises quite different and much more challenging organizational issues than those in cellular networks.

---

[100] An overview of the subject is given in [LGT98].
[101] or mobile hosts with the ability to perform switching functions
[102] ad-hoc [latin: ad-hūc, "even more"], a term adopted by the IEEE 802.11 subcommittee.
[103] ad-hoc, ad-huc : 1) until now, (and) yet. 2) still, and more, moreover.

Routing and multicasting in ad-hoc mobile networks face the challenge of delivering data to destinations through multi-hop routes in the presence of node movements and topology changes. In particular, rapid response to R/S movement requires autonomous organization mechanisms. The primary design problem in packet radio networks is the one of clustering the mobile switches into groups. This problem is motivated by two considerations: (i) spatial reuse of the control channel, in terms of frequency (FDMA), time (TDMA) or spreading code (CDMA), and (ii) reducing the information overhead.

The AN approach to ad-hoc mobile networking is a good entry point towards the goal of realizing dynamically adaptable autonomous systems by enabling the user to customize the behavior of the network elements by abstraction (virtualization) and re-programming, ([Tschu99b], [Chin00]). However, a user is still the (single) Observer of the system who provides a close snapshot view at the network at a single moment, or a sequence of (programmable) moments. In fact, he or she may have different views of the network at the same time or at different times. In addition, there may be not only one but also many users, each one of them with their different views at the network. Consequently, the user, the Observer along with his/hers views at the network is a part of the network itself by being involved in a recursive feedback mechanism, a fact that has been realized earlier in natural sciences such as physics and biology.

The above definition of an ad-hoc mobile network is closely related to the WLI model described in this work. The enhancements of the Wandering Network are summarized as follows:

1. Each host/node can *simultaneously* maintain *one or more* of the following roles (deployment modes) w. r. t. a communication path: *(a) dependent (client), (b) independent (server), or (c) interdependent (agent, proxy, both client and server).*
2. The functionalities of nodes/hosts are freely *reconfigurable* and *re-programmable* both in hardware and in software and can move/migrate from node to node.
3. There is set of principles for *self-organisation* and *self-maintenance* w. r. t. points 1. and 2. which keep the network going within a changing environment until at least two nodes remain present along with the communication path between them.

6.2.1.3 Routing System

A routing system in mobile networks should be able to manage node mobility in such a way that the communication endpoints are unaware of their relative movement or of the movement of the network itself. We define a routing system as a set of component functions for:

- monitoring the network topology and services (incl. mobility of nodes and services);
- monitoring the usage of network services to predict the required changes in topology and service provisioning;
- distributing this information for use in route (re-)construction, *node function re-distribution + topology (re-)design;*
- locating session endpoints;
- constructing and selecting routes;
- forwarding traffic according to selected routes;
- *adjusting network topology to the service usage life cycle.*

The routing system is responsible for deriving routes that meet the service requirements imposed by the end users. Changes in the network or session state may invoke changes to existing routes in order to maintain their feasibility. Such changes have a direct impact on the provided QoS and occur more frequently than in fixed wire-line networks. The routing system is required to detect and quickly respond to such state changes in order to minimize the degradation of services in existing sessions by utilizing minimal network capacity for maximum throughput and customer satisfaction.

6.2.1.4 Ad-Hoc Mobile Routing

There are two classes of routing protocols in ad-hoc mobile networks [RoToh99]: a) table-driven (TD), and b) source-initiated on-demand driven (SIOD). A number of algorithms have been proposed for both routing schemes. The table-driven protocols attempt to maintain consistent, up-to date routing information in the ad-hoc mobile nodes while responding to changes in the network topology by propagating updates throughout the network in order to maintain a consistent network view. The source-initiated on-demand driven protocols do not require periodic route updates; they create routes by initialising a route discovery and maintenance process only when desired by the source node. When a node requires a route to a destination, it initiates a route discovery process within the network. This process is completed when the route is found or all possible route permutations have been examined. Once a route has been established, it is maintained by some form of route maintenance procedure until either the destination becomes inaccessible along every path from the source, or until the route is no longer desired.

The TD ad-hoc mobile routing scheme is similar to the connectionless approach of forwarding packets, without regard to *when* and *how* frequent such routes are desired. A route to every other node of the network is always available, yet it requires frequent updates of the routing tables which lead to a significant signalling overhead as the network grows and the node mobility increases. Table driven routing protocols include among others Destination Sequenced Distance Vector Routing (DSDVR), Clusterhead Gateway Switch Routing (CGSR), and the Wireless Routing Protocol (WRP).

In the SIOD routing scheme, routing is not performed instantly; when a node requires a route to a new destination, it has to wait until this route is discovered. While TD protocols mainly support shortest path as QoS metric, only a few SOID protocols address QoS. While both schemes, TD and SIOD, support flat[104] routing philosophy and only some of the SIOD algorithms feature multicast capability. Source initiated on demand routing protocols include such protocols as Ad-hoc On-demand Distance Vector Routing (AODVR), Dynamic Source Routing (DSR), Temporary Ordered Routing Algorithm (TORA), Associatively Based Routing (ABR) and Signal Stability Routing (SSR).

For a detailed review and comparison of the current ad-hoc mobile wireless routing protocols please refer to the study of Royer and Toh [RoToh99].

[104] except for the Cluster Gateway Switch Routing (CSGR) table-driven protocol which uses hierarchical cluster head-to-gateway routing scheme.

MULTICAST

Multicast is regarded as the key ad-hoc mobile network service to support multi-party wireless communications. Since the multicast tree is no longer static, the multicast routing protocol must be able to cope with node mobility and cluster management including multicast membership dynamics (e.g. join and leave) by maintaining QoS for each single node. This means that it is inadequate to consider QoS merely at the network level without regard to the underlying media access control (MAC) layer [LiGe97].

MOBILITY PREDICTION

To accommodate highly mobile nodes (such as aeroplanes) while consuming a minimal amount of network resources, the routing system must be capable of predicting future node locations in addition to responding to current movements of the node. By tracking the non-random *mobility patterns* of the mobile nodes' behaviours, one can predict the future state of network topology and perform route reconstruction proactively in a timely manner. In addition, by using the predicted information for changes in the network topology, it is possible to eliminate transmissions of control packets otherwise needed to reconstruct the route and thus reduce overhead. In [SLG00], the authors propose and evaluate the effectiveness of various schemes to improve routing protocol performances by using mobility prediction.

Currently, it is not clear which particular algorithm or class of algorithms is the "best" for all ad-hoc mobile scenarios[105]. However, ad-hoc mobile routing approaches have introduced a number of new paradigms, such as exploiting user's demand and the use of location, power and association parameters. *Adaptability* and *self-configuration* are the key features of these approaches. Today, the major challenges for ad-hoc mobile wireless networks include: (i) multicast, (ii) QoS support, (iii) location-aided routing, and (iv) power aware routing.

The main concern in routing is, however, to find an efficient logical topology on top of the physical infrastructure and to design suitable routing procedures enabling the high performance of the network. Active networks allow the superposition of virtual network architectures on top of physical architectures in two dimensions: 1) vertically, throughout all network layers and 2) horizontally, throughout the different elements of the network.

6.2.2 INVESTIGATION FRAMEWORK

Active Networks emerged with the major design goal to accelerate the rapid introduction and deployment of new network protocols and services. The preliminary goal of the research programs in Active Networking during the 90ies were:

- to *identify* hidden assumptions in the "classical" type of networking (incl. both in-band and out-band signaling), and
- to *invert* (explore) the design space.

[105] A comprehensive review of ad-hoc routing protocols is given in [RoToh99].

Previous work in this area summarized and reviewed in research (e.g. [WGT98], [WLG98], [Weth99b] and [Chin00]) delivered satisfactory results. The long-term goal of Active Networks is to simplify the network through programming in order to design more complex (e.g. adapting, self-configuring, self-deploying, autopoietic, etc.) architectures.

An essential characteristic of the WLI approach is the inheriting ability to spread out information about architectural changes among the nodes/netbots of the Wandering Network instantly by encoding *executable re-construction (genetic) instructions* within the transported shuttles – the so-called *"network" genes, N-genes,* (Figure 59).

The Propagation of Architectural Changes
in a Wandering Network

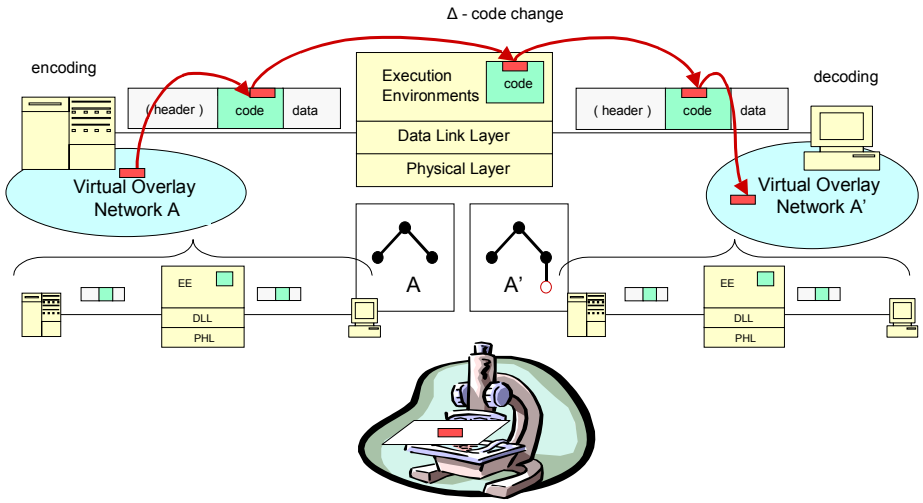


Figure 59: Encoding, transport, change and decoding of architectural information inside the Wandering Network

This is another unique feature, which differentiates the Wandering Network from all previous approaches in active and programmable networks. For this reason, we selected the subject of routing in ad-hoc mobile networks for a case study to demonstrate the feasibility of the WLI approach to network evolution.

The goal towards we are striving in of this thesis is the proof of the assumption that AN technology as an integral part of the WLI approach delivers an appropriate methodology for automating the process of route adaptation, and hence of propagating topology changes within a dynamically changeable network infrastructure. [The mechanism proposed in this work is a combination of user-initiated feedback mechanism of tracking user mobility with network-centric adaptation and maintenance of connection hand-off and service requirements within each traffic session. For this reason, network monitoring is distributed between the nodes of the network.]

Routing issues in ad-hoc mobile networking are a difficult challenge for protocol designers, since rapid reconstruction of routes is crucial in the presence of topology changes. The primary concerns in ad-hoc mobile networks are bandwidth limitations and unpredictable topology changes. In such an environment, it is important to minimize disruptions caused by the changing topology for critical application such as voice and video. Furthermore, agreeing on which algorithm is the"best" may even be more challenging. By using an active network approach we can (a) delay this decision until run-time, and (b) hopefully dissolve it by letting different routing algorithms run in parallel, [Tschu99a].

In the next section we are going to discuss the application of the WLI approach to adaptive multi-protocol routing and QoS maintenance in ad-hoc mobile networks.

6.2.2.1 General Model

Figure 60 and Figure 61 illustrate the paradigm shift in routing semantics by using the AN approach in communications.

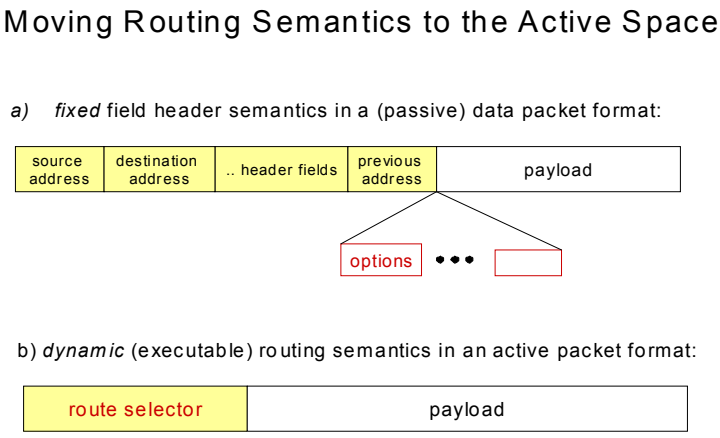


Figure 60: Changing the semantics of routing by means of active packets

In [Tschu99a], the author proposes a hybrid active networking approach for ad-hoc networking which maintaining two modes: a) a passive mode for plain data forwarding, and b) an active mode for active node re-configuration.

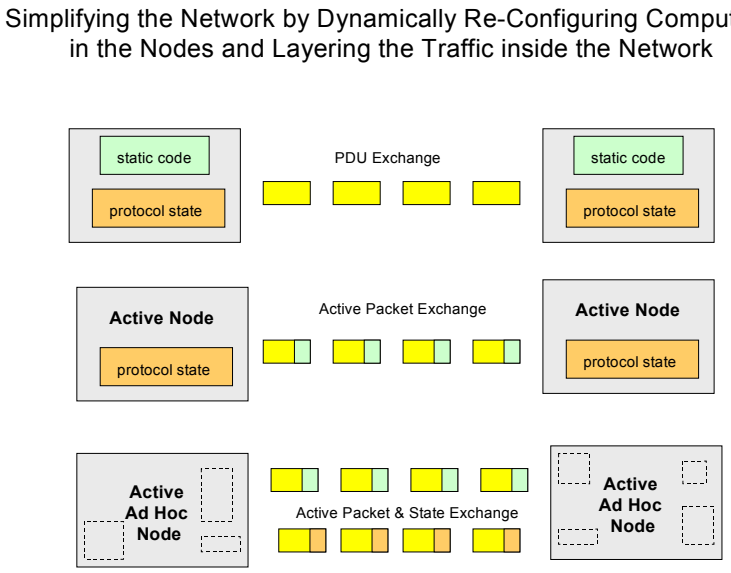


Figure 61: Evolving network activation

Thus, an active (hybrid) router/switch forwards data in the passive mode and uses active packets for signalling in the active mode, Figure 62.

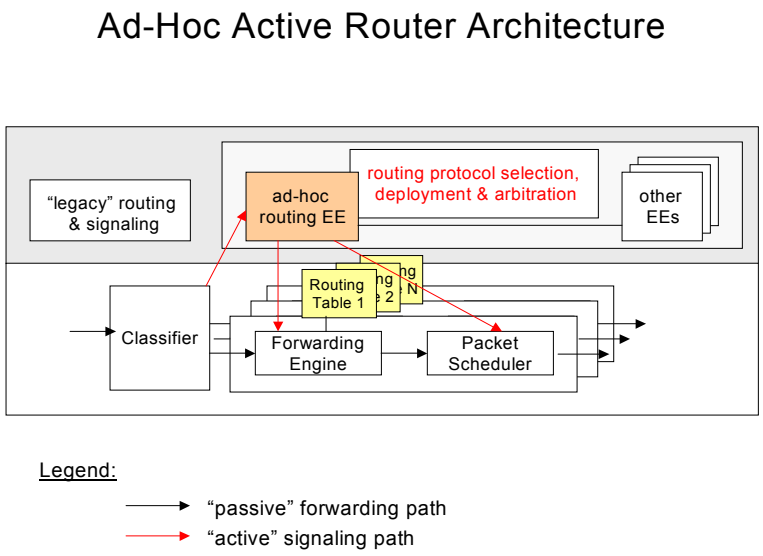


Figure 62: A multi-protocol active router architecture for ad-hoc networking

A similar approach is applied in the WLI case study for routing in ad-hoc mobile networks as a kind of *meta-routing deployment* scheme. Here, special active routing packets are used for realizing the signalling and the selection and deployment of the appropriate routing protocol within a multi-protocol executing environment.

The essential point in adopting WLI approach as a leading concept in the ad-hoc mobile routing case study is that we can not only invoke "on demand" a specific protocol to route a distinct packet based on the feedback messages from the neighbouring nodes which are carried by the active packets (shuttles), but also significantly simplify the entire messaging and evaluation scheme of the routing protocol by including executable architectural information.

6.2.2.2 The Target Domain: Ad-Hoc Mobile Routing

In their study about active services in wired and wireless networks, Kulkarni and Minden defined routing as a distinct class of AN protocols. The following benefits of active networks have been identified, [KuMi99]:

- *Overlaying several virtual topologies* on top of the same physical infrastructure in AN helps applications create different *classes of routes;*
- Applications can use their *own routing protocol* which is derived from the virtual topology defined by the application;
- Several *routing strategies* can be implemented and deployed *in parallel* (e.g. GPS/GIS based routing along with distance-vector routing, etc.);
- The primitives of a protocol of a routing class AN service require:
    - information about the *ports* of the neighboring nodes
    - information about the *queue* sizes at the ports

While keeping these frame in mind, we decided to demonstrate the practical application of the WLI model on routing in ad-hoc mobile networks. We selected this research domain not only because it is a very promising research area which attracted the recent interest of the active networks community ([Tschu99b], [Gold01]), but mainly because it best matches the Wandering Network philosophy ([Sim99f], [Sim01], [Sim02a]).

Traditional routing relies on distributed routing databases, maintained by the operators either in the network nodes or in specialized management nodes. There is *no guarantee* for routing information in ad hoc mobile networks.

In mobile ad-hoc networks, we face two challenges in routing management:

- all nodes are potentially mobile, and
- each node can spontaneously join or leave a communications session.

Therefore, new techniques are required to address routing in ad-hoc mobile networks. Active Networking is probably the closest starting point to tackle effectively the problem of tracking and propagating information about dynamic changes in the network topology in a predictable, distributed manner.

However, applying network-centric intelligent techniques to solve routing and other problems in mobile ad-hoc networks rise is going beyond the original paradigm of programmable active networks leads to more autonomy in service provisioning. Hubaux et al. [Hub01] argue that self-organization in mobile ad hoc networks is introducing some new and attractive perspectives in telecommunications:

1. *Self-organization* can bring a paradigm shift in the way networks are organized that can even lead to fundamental change in the relationship between Information technology and society;
2. Infrastructure-less, self-organizing network means more freedom; the network is deployed by the end-users and not by the operators, or some other party
3. Ad-hoc mobile self-organization is characterized by:
    - *quick propagation* of changes in topology or reachability
    - *quick adaptation* of the network w. r. t. these changes

The major assumption in applying the WLI model for this application domain is: *the more a node knows* about its neighbors and environment, *the better it can serve* the ad hoc mobile community. For instance, the delivery rate of the two-hop variant of the GEDIR[106] algorithm [LiSt98] can be improved significantly if each mobile ad hoc node is aware of its 2nd-hop neighbors, i.e. the neighbors of its neighbors.

It is evident that answering the question of how much of that "neighborhood" knowledge is required to effectively solve routing problems in mobile ad-hoc networks is not trivial, as the data overhead grows with the link-state tree expansion and some information might be redundant, dubious and even irrelevant. The solution can depend on the specific topology, on the node mobility and capacities, on the traffic characteristics, etc.

However, it is at least clear that many unexpected situations such as the *count-to-infinity problem* or *the dubious split-horizon problem* [Tan96], which may occur due to spontaneous changes in the network topology, could be avoided or prevented[107], if the mobile nodes were not only evaluating instant messages sent from their neighbors about changes in their rooting tables, but also capable to superposition and interpolate the different perspectives of the entire network as seen by all nodes in parallel at a given moment.

In other words, **if mobile nodes, *netbots*, were capable to recognize the different *network topology patterns* reported by the surrounding environment, they could be able to participate actively in using and changing these patterns to fit some optimization criteria such as shortest path routing or best QoS.**

[106] GEographic DIstance Routing

[107] It should be noted here that to answer the question about the *net transparency* in the views of the single netbots is not a trivial task even if all nodes tell the truth about themselves and their environments; in general this depends on some additional model details and parameters such as power, channel assignment, etc.

6.2.3 RELATED WORK AND PERSPECTIVES

In his thesis, Chin [Chin00] investigated the viability of applying active networking technologies to routing in mobile communications. In this work, various benefits of ANs were demonstrated through extensive simulation studies in unicast and multicast routing for IP and ATM networks. The author concluded that AN-based protocols manifest the following benefits in mobile networks: (i) efficient adaptation to mobility, (ii) reduced signaling overheads, (iii) high reuse of allocated network states, (iv) extensibility, (v) topology independence, and (vi) scalability.

The following two sections summarize the major results of this work.

6.2.3.1 General Advantages

STATES

ANs promote iterative reuse of allocated states for both connection-oriented and multicast connections. For instance, a connection in mobile ATM networks can be iteratively updated to maximize reuse. Similarly, multicast states at routers/switches can be preserved in such a way, that only parts of the multicast tree can be affected.

Since in ANs allow computation at each (active) router/switch, routers/switches can make their own decisions based on the available information residing at their site or at their neighboring routers/switches and determine locally whether updates are necessary or not. In this way, the core network elements make their own decisions rather than the end-host(s).

SIGNALING

The main source of signaling overheads in a conventional network is the examination of routers/switches for data to be processed at the end-hosts. Because signaling messages are local in ANs, this procedure is not required anymore.

The message overheads are minimal compared to solutions deployed from the end-hosts, since computation in active networking is distributed among the (active) routers/switches. For instance, there is no need to obtain a snapshot of the multicast tree before any adaptation can be performed. Local computations can be performed *when events happen* and not *after*. One can even use some heuristic approach to *predict* even the occurrence of events at active router/switches, e.g. node movement detection by means of GPS/GIS technology. This is especially important in the case of congestion control where it is important to isolate the congestion and to notify the end-hosts of the congestion state.

FILTERING

Active routers/switches can play an active role in filtering out redundant signaling messages. In a multicast service, unnecessary path optimization and join the core nodes to conserve bandwidth can remove messages. The ability to filter out redundant messages is crucial in ensuring the scalability of the protocol as the number of subscribers grows.

Active routers/switches can be also extended with a number of useful network services. For instance, they may host some programs that can be used as protocol boosters or for providing different QoS to different parts of the multicast tree.

6.2.3.2 Advantages for Mobile Networks

ROUTING

In mobile networks, routes need to be updated frequently, even more so in multicast communications, particularly when the host migrates. Therefore, it is crucial that these updates are performed as soon as possible. In [Chin00], the author has shown that TCP's performance improves significantly with the faster arrival of binding updates. The case of local computation at the router/switch becomes even more essential in connection with QoS provisioning. Solutions that reallocate QoS after each host's migration are not useful when there are a large number of subscribers in a multicast session.

By using AN-based protocols, updates can be only performed on desired parts of the network, in the connection path or multicast tree.

6.2.3.3 Open Issues

ACTIVE PROGRAM LOCATION

The deployment model, presented in [Chin00] for wireless multicasting, considers only routers/switches that are on the communication path. As a result, strategic positioning of programs is not required.

The way in which active programs should be positioned within the network has not been studied sufficiently until now in the AN community. For instance, in [SRBW01] the authors investigate a remote socket architecture implementing a protocol booster concept [Feld98] in a front-end proxy node for wireless links. The performance results of the system are quite satisfactory when used as a single node. However, this and other works in the area still cannot answer the question *how many* such nodes are required and *where* the active programs should be positioned in order to have an optimal solution.

In fact, strategic program positioning actually defeats the purpose of having AN. One benefit of active networks is their *topology independence*. In mobile networks, strategic program positioning is infeasible because of the frequent host migrations. However, deploying active programs within a session alongside signaling messages, on the communication path facilitates the identification of *dynamic* strategic points, which may be important in some areas such as routing in ad-hoc mobile networks (AMN).

6.3 APPLICATION SCENARIO: WLI AD-HOC MOBILE ROUTING CASE STUDY

In terms of a physical realization, the *Wandering Logic Model* can be probably best suited by the concept of an *autonomous Active Ad-hoc Mobile Network (AAMN).* Therefore, we selected ad-hoc mobile networking as application domain to demonstrate the benefits of the WLI approach. Unlike cellular wireless networks, an ad-hoc mobile wireless network does not have any fixed communication infrastructure. In other words, for an active connection, *the end host as well as the intermediate nodes can be mobile.* Furthermore, each host acts as a router and moves following an arbitrary scheme. Therefore, routes are subject to frequent disconnections.

The proposed WLI model use shuttles (i.e. active packets / mobile code) to control the routing state in mobile nodes. The following assumptions are guiding the WLI routing model:

1. **The Self-Reference Principle:** Each netbot knows best its own configuration and routing state, as well as *how* and *when* to display it. This information can be encoded in shuttles and propagated throughout the network. The netbot also maintains its own *reachability tree*[108] w. r. t. a particular transmission session, a netbot's role, application ID, or each data flow transfer ending at it.

   In addition, each netbot always provides *true*[109] (fair) information about its connectivity to other nodes. Of course, some selection/filtering mechanisms might be applied to different subsets of nodes depending on some self-maintenance and performance optimisation criteria; it is not required that a netbot always tells the *entire* information about its state.

   Finally, netbots are supposed to be communicative and *cooperative*[110]. There might be different mechanisms to stimulate their cooperativeness in providing their resources to other netbots such as the *nuglets* referred in [BuHu01].

2. **The Multidimensional Feedback Principle:** A reachability tree can be dynamically verified and updated with the corresponding reachability trees (or parts of them) of other netbots. This "directed" routing information is periodically verified against and updated by the network topology patterns contained in a special kind of routing shuttles, *r-shuttles*, which periodically traverse the netbot.

---

[108] In general, mobile nodes may run several different applications (e.g. multicast sessions) allocated to different subsets of the network. They are also serving as routers for other nodes. We used the term "reachability tree" instead of the well-known routing table for two reasons: (i) the ultimate goal of a routing algorithm is to determine the shortest path to each destination which may be the node itself; therefore, a routing tree rooted at the sink represents a loop-free set of paths which can be best maintained at the sink itself; (ii) routing trees contain *interconnection patterns* that can be easily verified against and matched with other patterns carried by shuttles.

[109] The treatment of un-trusted systems such as e.g. the Byzantine Generals problem is not part of this work. However, even when following this assumption, the restrictions of footnote 107 do still apply.

[110] *Fairness rules* for node cooperation is an interesting research issue because of the limited performance of the mobile nodes which are required to serve as routers for their neighbours. In general, a trade-off between "routing for others" and the own tasks' maintenance should be considered. It is clear, that each node can serve only a limited number of neighbours depending on its own configuration and power consumption. In general, the more power the netbot has, the more neighbours it can access and the more hops it can send its packets to.

6.3.1 METHODOLOGY

The WLI methodology addresses the following subjects in ad hoc mobile communications:

1. An on-demand exchange of communication environment (CE) functions can take place as soon as the netbots approach each other the reach of their active CEs.

2. There is a set of fixed, network nodes ("docked" netbots) for power recharging and complete update/renewal of CEs by the "agile" netbots.

3. The newly acquired CEs such as (parts of) routing algorithms are loaded in the configuring EE which assembles the functional pieces into a program in the netbot.

4. Re-routing shuttles: re-routing the next available: state-of-the link, state-of-the-node, entire RT.

5. Predictability: including prediction information in the shuttle, e.g. expected congestion.

6. Select different strategies for routing based on a "benefits" plan and feedback about power consumption, traffic threshold values, QoS requirements, creditability, event-based.

7. Information about the positioning of netbots is generally managed via GPS/GIS access.

8. Intelligently managing data about "neighbours of the neighbours" to ensure predictability.

9. "Living Routing Tree", tracking how agile the netbots are: classes of netbots.

10. Updating RTs depending on the traffic patterns. / Recognizing routing patterns: topology.

11. A netbot can promptly select a routing strategy upon evaluating the current RT state.

12. If the netbot is not able to find a solution, it can deploy 2-3 algorithms in parallel (incl. measurements) and select the best solution.

13. There are 3 parts of the methodology:
    - General conditions
    - Target oriented strategy decision
    - Maintenance

Let us consider the following mobile ad-hoc routing scenario for routing within a wandering network. Initially, there is a single *netbot* traversing the 2D space as illustrated on Figure 63. The legend explains the main components of the architecture. It is essential, that the netbot can only communicate within a certain transmission range r_t (an allocated radio channel) which is a function of its power consumption. Figure 64 explains the internal netbot architecture.

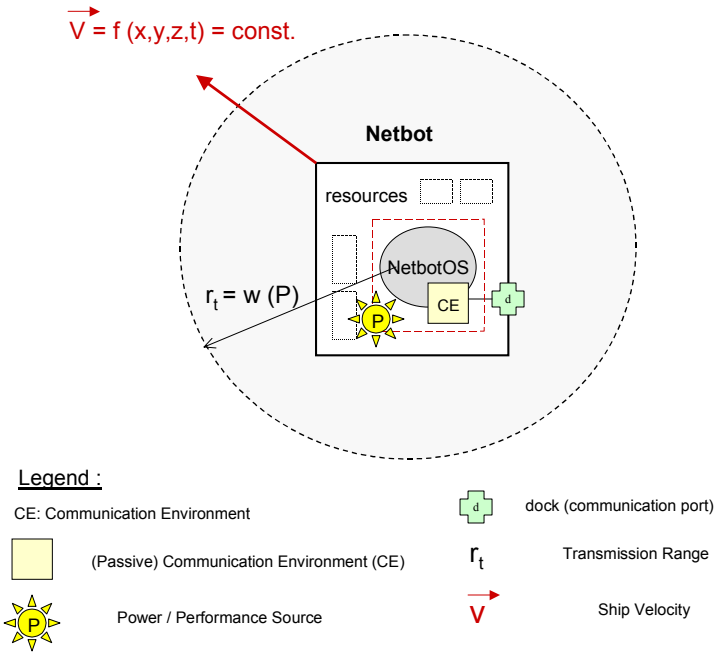


Figure 63: A netbot, traversing the 2D space with a constant velocity.

The netbot is able to exchange shuttles with other netbots via docking ports, virtual interfaces (APIs) which open communication channels from activated communication environments (CEs).

Let us now have a second netbot, B approaching the first one, A, as shown on Figure 65. Note, that netbot B has two transmission ranges (radio channels), the one of which is identical with this of netbot A.

As soon as both netbots are close, enough to each other, their common communication environments CE are activated (automatically) and they start exchanging messages. Then, node B learns from node A that the last one has some information to “upload” for node C that is currently out of the scope of the two nodes. Netbot B looks back in its communication history (“netbot log”) and informs A that it has met C in the fleet W, a cluster group of netbots which can be accessed through the cluster head or “van” F according B’s last record. B cannot take A’s load even if it is ready to pay 2.000 nuglets for it, since B has a very tight schedule to deliver its own load to the destination haven D. Besides, B’s capacity is almost completely used to take someone else’s data on board.

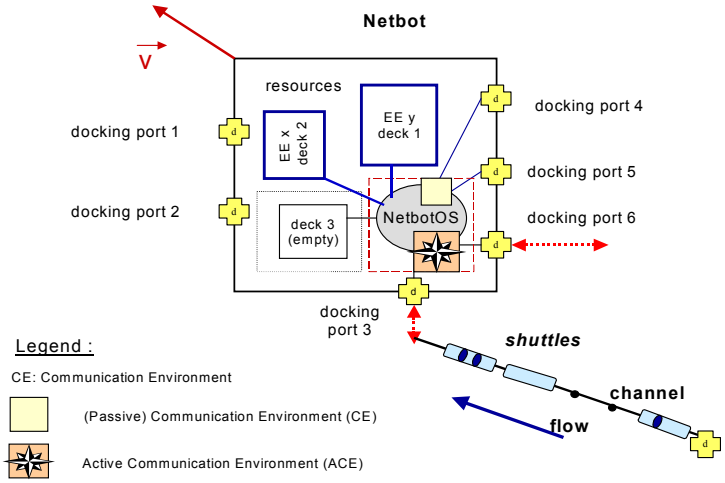


Figure 64: Schematic representation of the internal netbot's architecture

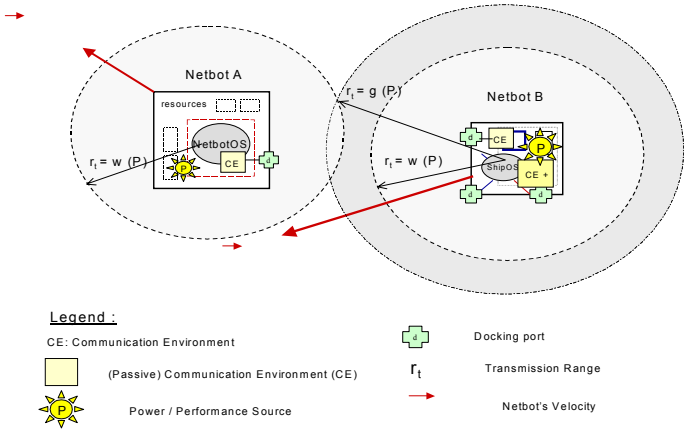


Figure 65: Netbot B detecting netbot A within its transmission range r_t = g (P)

However, B suggests A to try to route A's data to the fleet W, since it has some free processing capacity that can be used for as less as 10 nuglets per terabit (npt) at the moment. Precondition is that netbot A has the newest communication environment CE+ that extends the operating radio spectrum, and thus the transmission range, which is also used inside the fleet W. B, suggests A to upload CE+ free. A agrees. CE+ is then uploaded from B to A by means of shuttles as shown on Figure 66.

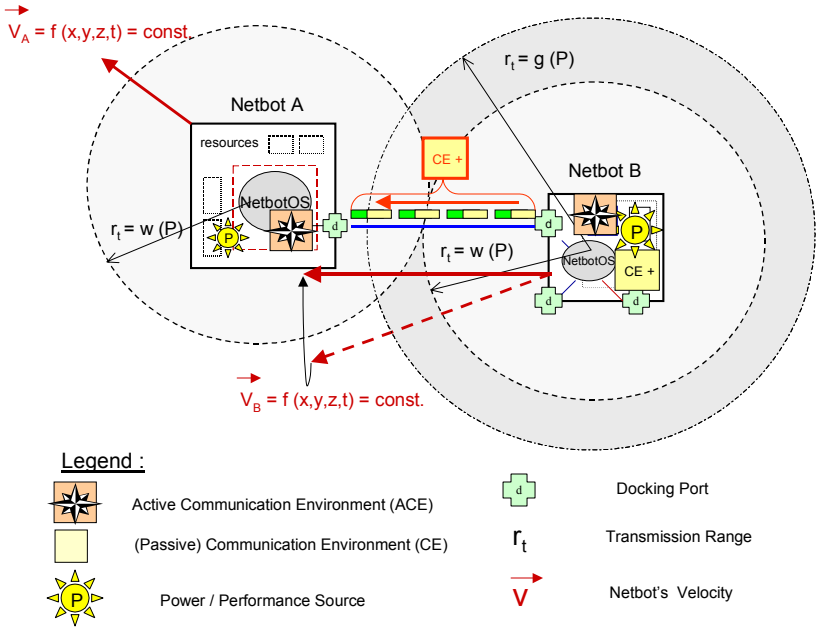


Figure 66: Netbot B transports its second communicating environment CE+ to netbot A

There is nothing special in this action, which simply performs **a function's transfer from one mobile node to another. This procedure does not necessarily need to be realized by means of WLI. It only shows that a netbot can acquire a new capability to perform its mission, in our case - a new radio channel encoding which improves its transmission range to access its destination[111] remotely.**

Until now, we illustrated only how a single netbot can continuously develop and exchange new functionalities in WLI, such as e.g. a new communication protocol or an update of an old one. This is a typical feature of Active Networks.

One can regard communication environments as dedicated EEs, which can be not only node resident and activated upon request, but also uploadable and exchangeable like active applications (AA). Figure 67 shows an example of such a developed communication architecture where only one CE was initially available. Note that all three CEs can be active in parallel w. r .t. different communication domains, a typical WLI characteristic.

[111] It should be noted, that in mobile ad hoc networking, the delivery trade-off really goes between mobility (distance tracing) and remote access (data tracing). If a netbot knows (i) where its destination is (e.g. by using GPS/GIS technology), (ii) it is fast enough to get close to the target ahead of its schedule and (iii) it can upload the data through a fast link docking channel, this netbot does not really need to make complex computations on how and where to route in a complex multi-hop ad hoc environment (which is certainly required for fixed nodes). It could simply "move there" and deliver the data.

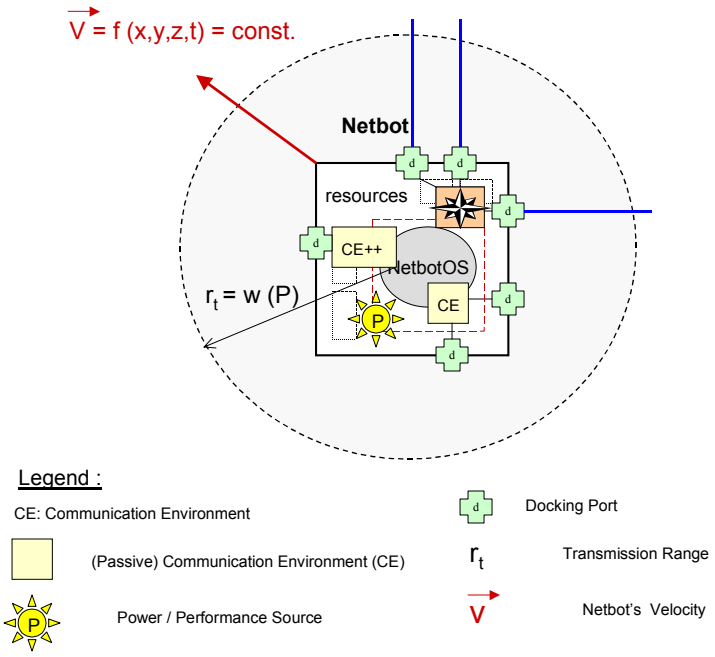


Figure 67: A netbot with three communication environments, one of which is active

Now, both netbots have the same communication environments and can participate in larger ad hoc mobile networks supporting these two environments as shown on Figure 68. Both netbots can decide (a) to split and follow their initial destinations, or (b) to build a temporary cluster (by changing their directions and velocities) and maintain a constant distance *d* to facilitate further communications.

While (a) does not deliver the desired result directly, the interesting part begins with (b). Thus, we consider that A and B decided to stay together for a while by keeping a distance d between them with the original speed and destination of B. To make the case even more interesting, we assume that there is no navigational help such as GPS of where the node C might be. Let us assume that A does not have any routing table. Let us assume the same for netbot B, yet with the condition that B has a *reachability tree*, r-tree[112], including netbot C in its history.

[112] R-trees may be hierarchically indexed because of the different kinds of transfer, domain and content they serve.

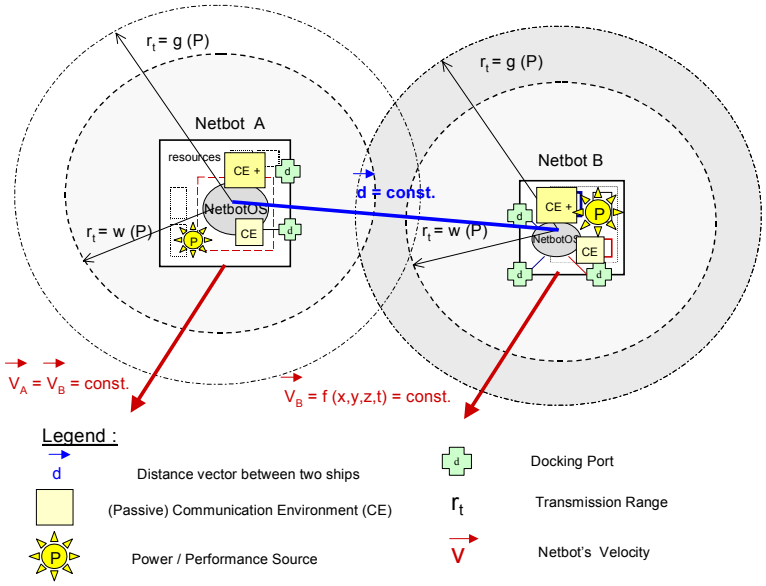


Figure 68: Two netbots building a temporary cluster

We postulate also that B has information about some other netbots' "netbot routing logs" with their reachability trees that may contain a useful[113] route to C cached in B's history file while being traversed from their *routing shuttles*, r-shuttles, previously to meeting A. Thus, netbot B can transfer this information or parts of it to A in the same way as it did with the communication environment.

Then, both netbots, B and A, can try to verify the different pieces of C's reachability information (distributed processing) and derive a *hypothesis* of how to optimally route from A to C in case that such a route or an entry point to the fleet W does exist. This hypothesis should be checked and updated again and again with newly arriving r-shuttles carrying r-trees from their sources until an upstream connection to C or a gateway to W, where C is supposed to be, can be established. We call the encoded r-trees inside the r-shuttles, *r-genes*[114]*,* and the process of encoding and decoding them - *r-geneering*.

[113] The good news is that such reachability trees of netbots or "vanguards" (vans) may can be encoded in the shuttles along with some useful information such as time, location, destination and constant speed to predict their emergence and prepare the transfer or routing in some access area in advance.

[114] There are different kinds of node genes, *n-genes*, in WLI. Some of them, such as the r-genes, are responsible for encoding the routing state of the network from the netbot's perspective, whereas others are used to describe some other node information, such as e.g. the EE history.

Let us now consider that one of the two netbots, e.g. A[115] (Figure 69), can establish a direct connection to C, which is supposed to be a single node (in case that it does not belong to a fleet), a van itself, or a gateway (border) netbot of some fleet W. In this case, netbot B can communicate to netbot C via netbot A using a different access spectrum range, common to A and C. Netbot A performs as a router agent for B and C in delivering information between them. Therefore, it has two active communication environments for each connection.

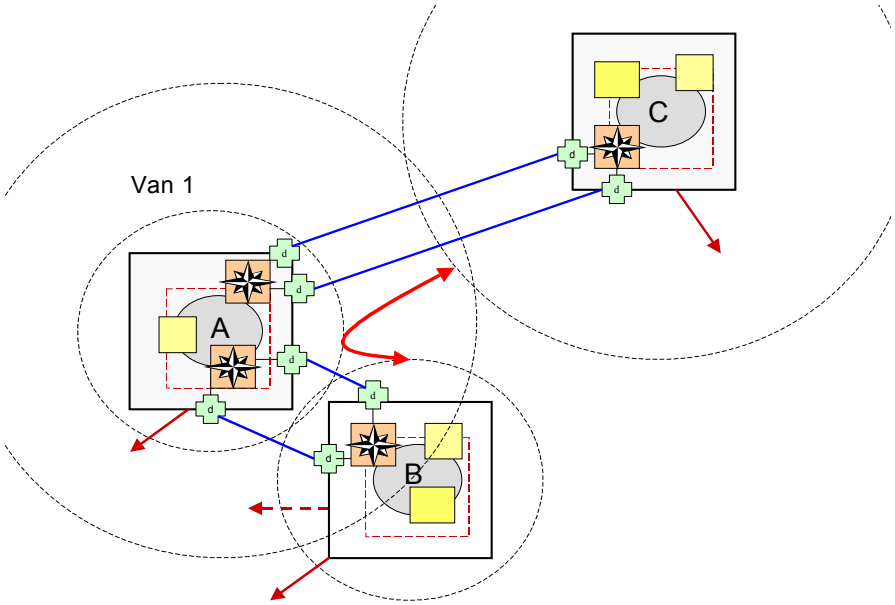


Figure 69: A router agent with two active communication environments

The WLI Principle of triggering simultaneous multiple roles inside a node allows the automatic activation and self-adjustment of different communication environments, i.e. different algorithms or parts of a distributed algorithm based on a feedback for the different distance ranges and the access spectrum, in parallel. This kind of network activity is similar to the sentient computing approach, [Add01], in adapting a communication environment to a particular user profile. However, in the WLI case this adaptation is: a) performed on a mutual base in a multiple nodes environment, and b) including dynamic changes in the behavior, i.e. of the profile, of the particular network elements. Therefore, each communication environment is dynamically adaptable to the characteristics of the particular connection for which it is responsible.

There can be multiple CEs active on a single netbot. For instance, Figure 70 shows the communication of information between the netbots B and C via a second van D that happens to maintain a third connection inside or outside its own fleet, etc. The only limitations in deploying multiple communication environments are power and resource consumptions, as well as the performance degradation of the acting CEs.

[115] We call A, a *fleet head* of the netbot set (A,B) in WLI, which should be considered as a synonym of a “cluster head” as referred in the ad-hoc networking literature .

An instant change of a netbot's role, e.g. a van assignment is also possible; it depends on the environment conditions and the movement and activity of the single nodes within the cluster (fleet). Therefore, a wandering network represents a *second*[116] *order* autopoietic system [MaVa80].

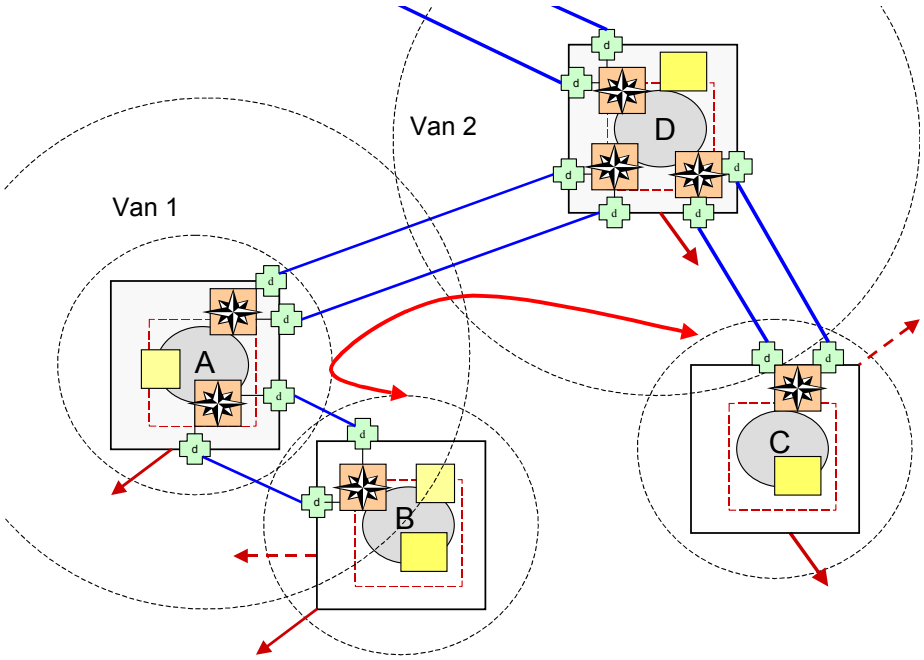


Figure 70: Multiple active communication environments on single van netbot

6.3.2 The WLI Routing Algorithm

Let us now consider a scenario of the emergence of a wandering network. We point out that this example solely illustrates the capability of the WLI approach to handle routing in active ad-hoc mobile networks in a comprehensive, straightforward manner.

The algorithm is called *WARAAN*, WLI Adaptive Routing Algorithm for Active Ad-hoc Networks, [Sim02b]. It is a Gnutella-like protocol [GNUT] which is not limited only to large scale peer-to-peer networks and evolves its full potential in ad-hoc mobile networks where the algorithm can be layered with a few other routing algorithms and policies depending on the application context.

[116] A first order autopoietic system maintains the stable, i.e. adaptable recursive, operation of its interactions inside the system itself, such as e.g. the internal metabolism of an organic cell. In our case, this corresponds to the self-maintenance of the active nodes themselves.

Our example is based on the notion of an *attributed non-terminal* which was defined by Vogt, Swierstra, and Kuiper as a part of their milestone work on Higher-order Attributed Grammars (HAG), [VSK89], which is turn based on the historical contributions of Kastens [Kast80] and Knuth [Knuth68] on the semantic of context-free languages. In the past 30 years, attributed grammars have proved to be appropriate means for structured modelling in many language-based application areas such as pattern recognition, [TsFu80], graphics systems design, [BaZa89], electronics and logic circuit programming, [Holm97], as well as neural networking [HuBr98]. Because the goal of all routing algorithms is "to discover the *sink*[117] *trees* for all routers", [Tan96], and because of the simple linear encoding of parsing trees generated by the production rules of a context-free grammar, we preferred to use the HAG model in our WLI routing scenario with some modifications, instead of generating a more complex network model based on a general formal approach such as the graph grammars, [Goet88]. Moreover, the universality of higher-order tree transducers was recently motivated again by Noll and Vogler [NoVo01] for a series of applications.

We assume that the routing state of the wandering network is completely described by the set of reachability trees (r-trees) T_R of the individual netbots in the network. A reachability tree is the replacement for a routing table in WLI.

> **Mobile Definition 2:** A *reachability tree* (r-tree), T_R, is a dynamic directed tree structure allocated in the operating communication environment of a WLI netbot and responsible for the base routing in a Wandering Network. The root of the r-tree is always the host netbot. The leaves and the intermediate nodes of the tree are the corresponding netbots from which the host netbot can be reached. Each netbot is responsible for:
>
> 1. maintaining its r-tree by collecting information from the shuttles traversing that netbot;
> 2. forwarding shuttles to other destinations; and
> 3. reporting changes in the own r-tree structure, such as establishing new connections or cancelling old ones, to its neighbours.

A modified higher-order attributed grammar[118] (HAG) represents our model for the netbot's reachability tree.

In our model, each node except the root represents an attributable virtual non-terminal. This means that at every single moment the r-tree can be expanded or collapsed at such a virtual non-terminal. They are synthesized attributes of this non-terminal represent the computed potential links to other netbots of the network.

6.3.2.1 Informal Description

Let us now go back to the construction[119] of reachability trees in WLI and assume two single netbots, A and B, freely traversing the two dimensional space.

---

[117] To emphasise the netbot-related nature of the routing path generation in WLI, we decided to use the term "reachability tree" in this work.

[118] The formal definitions can be found in the appendix.

[119] The complete scenario for constituting a wandering network of up to six netbots of the same fleet and than back to three and four netbots is given in the appendix.

There is no connection established between them[120]. Thus, each netbot contains only one single element in its reachability tree: itself, the root. Then, at some point of time both netbots approach each other within their access range. One of them, say A, initiates a connection protocol with the other netbot. The opposite side replies positively and the connection is established. Next, each netbot constructs a new branch of its reachability tree ending at the new neighbour.

At the next moment, a new netbot C approaches A and requests a connection. Upon a positive reply the connection is established and the local r-trees at each node are extended by the new branch. However, this time both B and C are unaware of the fact that they may contact each other by letting A to route the shuttles between them. Therefore, netbot A is required to inform each one of its neighbours about the existence of other members[121] in the network. This is achieved by encoding and encapsulating the correspondingly "missing branch" information as executable *r-genes* (reachability genes) into the *r-shuttles*[122] (routing shuttles) which A transmits to its neighbours. Instead of a destination address and a TTL-counter (time-to-live), each r-shuttle is carrying an encoded tree branch called a *q-tree* (quest tree) it is required to traverse until being discarded at the end nodes[123]. The communication environment can manipulate[124] both the q-genes and the r-genes of an r-shuttle in order to update their information, Figure 71. In case that netbot C also has some neighbors it can route to, it is required to send this information via r-shuttles towards A, which in turn takes care to distribute it along the remaining branches of its reachability tree.

Generating a new branch[125] of the r-tree on a netbot and dispatching r-shuttles to inform the neighbours about the change can be performed simultaneously. Besides, the same procedure is performed simultaneously on both sides of the newly established connection.

---

[120] For the sake of simplifying the example, we postulate that all netbots have only one access range (frequency), so that the network is constructed within the same fleet. This implies the existence of only one fleet head which we define as the netbot which establishes the first connection (the originator). Of course, this role can be later transferred to other netbots depending on the changing environment of the network.

[121] We postulated in WLI that *fairness* and *cooperation* are a must. All kinds of hiding and manipulating information for any reason are not subject of this work.

[122] Reachablity shuttles in this scenario are much more likely to be regarded as dedicated mobile agents, ff.

[123] Of course, an r-tree update method using an IP-like final destination and a TTL-counter could be also applied to guide the r-shuttles and limit their rotation in the network. However, since r-shuttles have a particular role to inform the netbots on their path about changes in the r-tree of their originating netbot, it does not make much sense to generate multiple instances of them at the originator for each single node in the branch (multiple destinations). Moreover, r-tree updates are supposed to occur much seldom than the actual packet transmission between the netbots. Furthermore, while traversing the branch, a part of it may simply disappear for some reason, so that the r-shuttle can be discarded at the intermediate node. In addition, we use this case study to demonstrate that the WLI shuttles may carry multiple instances of encoded structural information: here – one for the r-tree change of the originating node and one for the q-tree to be traversed by the r-shuttle. We claim that this executable information does not overload the r-shuttle because of the limited number of ad-hoc netbots communicating on the path (up to 5 hops between two fleet clusters of a second degree hierarchy).

[124] Please note again that fairness has the highest priority in autopoietic WLI networks.

[125] There are two essential differences between Higher-order Attributed Grammars (HAGs) and reachability trees. Firstly, whereas a higher order attributed grammar is constituted of terminals, non-terminals, and inherited and synthesized attributes, a reachability tree is constituted only of non-terminals which can be attributed later by synthesized attributes. Secondly, whereas a HAG can be only expanded by a branch in a parsing tree, a reachability tree can be expanded and collapsed, at any single node of the tree.

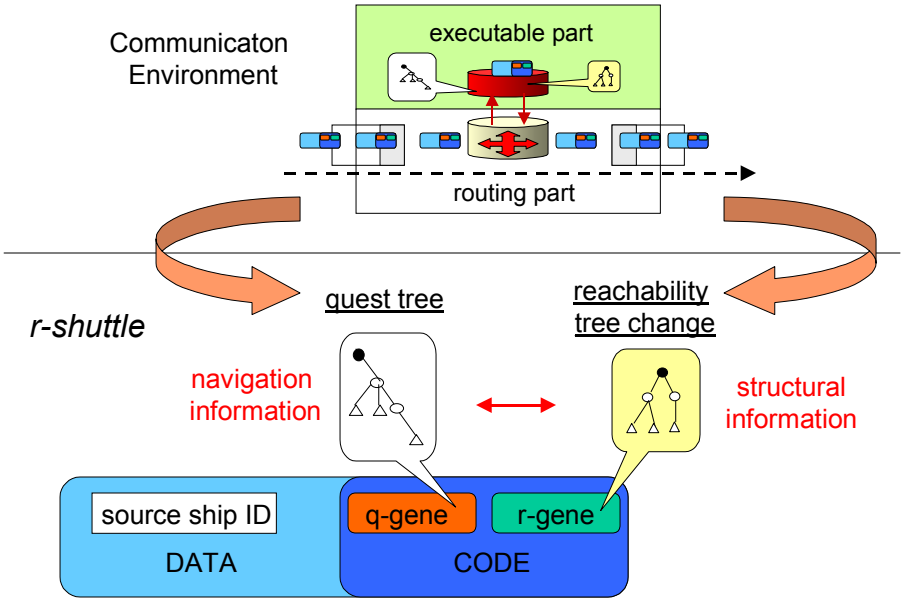


Figure 71: A communication environment manipulating the genetic structure of r-shuttles

These are two important advantages of the distributed WLI routing algorithm. We claim that by using the r-tree and q-tree types of encoding in the shuttles, the updated routing information is sent effectively to all affected nodes of the network. The computing overhead for encoding and decoding the r-trees should be considered as minimal because of the event related character of the reachability tree updates.

Figure 72 illustrates[126] the first two steps of the WLI routing algorithm[127], the *projection* phase:

i) Connect (X, Y) & Build (T'_X,T'_Y), and
ii) Inform (X, Y, T_X ,T_Y).

We call the second phase of the WLI algorithm the *capturing* one. It starts with the evaluation of the incoming shuttles and the expansion of the r-trees at the referred non-terminals by the "missed branches" encoded in the r-genes.

---

[126] The coloured circles denote *acting* nodes with the red one being the new netbot joining the network. The oval legends display the (parts of the) r-tree contents represented in the particular elements with the colour ones being active in the particular step of the algorithm. The shuttles on the figures are assumed to contain q-genes.

[127] The notion is taken for the general case of two netbots X and Y and their reachability trees T_X and T_Y . The prime sign upon T means the next state or the change of the reachability tree.

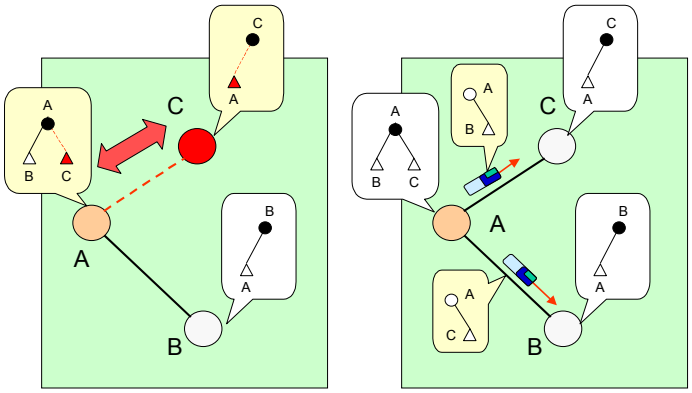


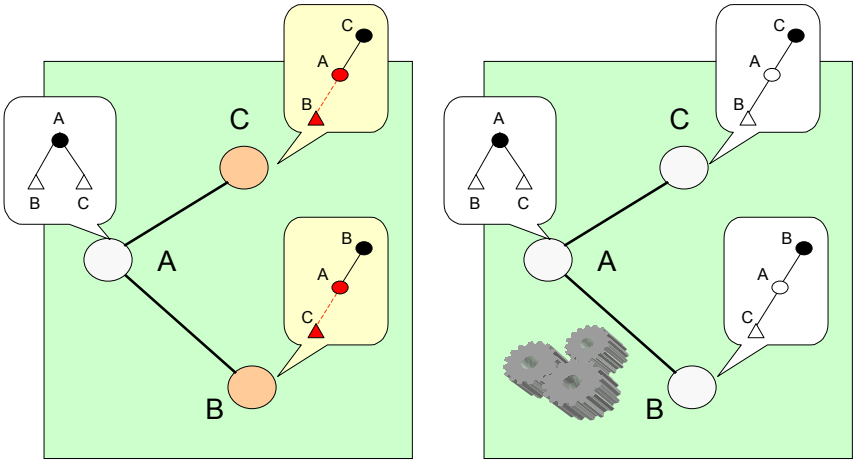
Figure 72: *Projection*: building and transporting r-trees

As soon as the r-shuttles arrive at their destinations, they are guided to the corresponding communication environment responsible for the link they come from. The CE then unpacks the "missing branch" information encoded in the *r-genes*, which are part of the executable code carried by the r-shuttle, and verifies it with the structure of its reachability tree. If the delivered information is redundant, it is discarded. For instance, this happens when the r-tree has been already constituted that way that the r-gene information represents a sub-branch of the netbot's r-tree (perhaps by a previously delivered shuttle from some other source[128]). In case that the "new branch" information is a *really* new one, the CE takes care for expanding the r-tree at that virtual non-terminal which is assigned to be a root in the sub-tree encoded in the corresponding *r-gene*, Figure 73, step iii.

Finally, the reachability trees of all netbots are verified against each other (Figure 73, step iv) by broadcasting periodically r-shuttles containing the entire r-tree to the neighbors which analyze the incoming information on their side with the local tree structure and send back their feedback to the originating node. If no feedback is registered on, a connection after some period has elapsed, the associated link is considered for failed and the change is reflected in the local r-tree and reported to the neighbors. If the requested netbot is only an intermediate station on the path of the shuttle, the responsible CE updates the netbot's reachability tree by the r-shuttle's information and forwards it to the next hop on the shuttle's path. If there are any new structural changes on the shuttle's route ahead known by the CE, the shuttle's q-tree is updated[129].

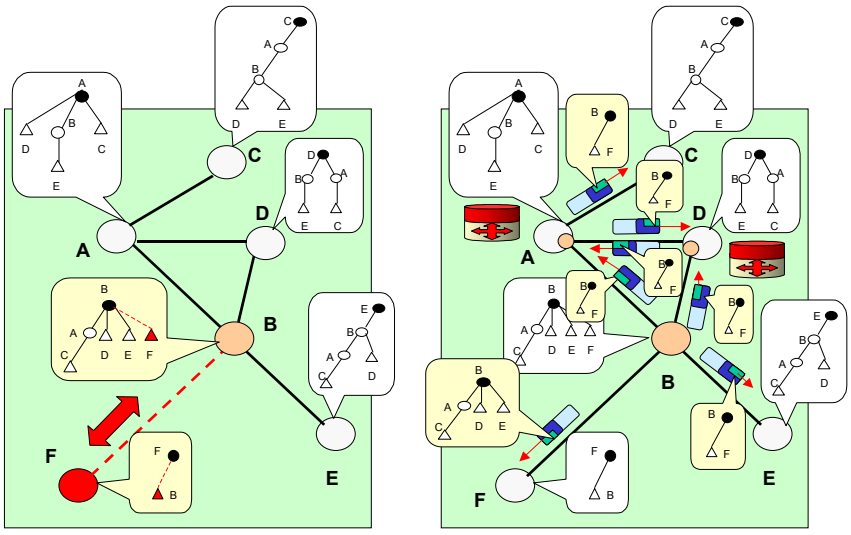
[128] Please refer to the complete 6-netbot example in the appendix.

[129] The r-tree can be also updated if there has been some recent changes in the source netbot connectivity known for some reason by the intermediate netbot, provided that the intermediate netbot is allowed to make such changes. For the moment, we assume the following: once being encoded in an r-shuttle, the r-tree remains unchanged (read-only).



iii) Expand (T_C , T_B) iv) Verify (T_A , T_B , T_C)

Figure 73: *Capturing*: expanding and verifying r-trees



i) Connect (B,F) & Build (T'_B, T'_F) ii) Inform (A, C, D, E)

Figure 74: Projecting the inclusion of the sixth node of a wandering network

Figure 74 and Figure 75 illustrate the more complex example of updating the r-trees in "three-nodes-ahead" evolving ad-hoc architecture[130]. Note, that the same four steps – `Connect`, `Inform`, `Expand` and `Verify` -, are taking place every time a new netbot joins the fleet. This is because of the distributed and parallel nature of the WLI algorithm which complexity[131] is estimated to be **O (4 + m)**, where "m" is the maximum number of hops throughout all r-trees in all nodes participating the network.

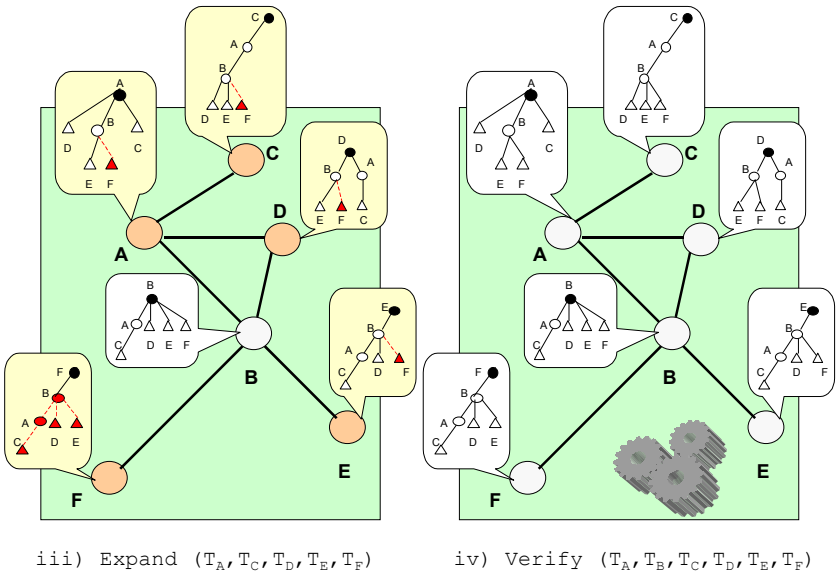


Figure 75: Capturing the inclusion of the 6th node of a wandering network

Note that the small coloured circles inside the netbots A and D in step `ii` on Figure 74 denote the dual nature of the node in two consecutive steps: a) it acts as a *virtual non-terminal*[132] ([VSK89]) which expands its r-tree at some node upon evaluating the first incoming r-shuttle which delivers new information about the connectivity of that node, and b) it acts as a router/ filter/replicator[133] for all other incoming shuttles with the same r-gene content from that node.

---

[130] Please refer to the appendix for tracking the consecutive steps of the network expansion.

[131] **Note**: the above assumption holds for flat hierarchies, i.e. the ones with only one level of interaction: the fleet or the local cluster as an ad-hoc distributed network (i.e. without a "head"). In case that new hierarchies are introduced, we have to adjust the algorithm to match the emerging levels and architectures of interaction. However, even then, the number of steps of the algorithm is progressive, yet limited which allows us to easily combine our approach with some well-known multi-level clustering strategies in ad-hoc mobile networking such as [GeTs95], [Shar96], [Iwa99]. *In each O(x) formula we imply 1 step for verification at the end of the cycle.*

[132] The new link computed and assigned to the r-tree by the local CE after decoding and evaluating the r-gene of the incoming r-shuttle is *a new synthesized attribute* of the virtual non-terminal *"netbot"*.

[133] in case that meanwhile there has been established a *new link* ending at that node which is not considered in the q-tree of the incoming shuttle at the time of its encoding in the source netbot.

In case that a new, direct link is established between two netbots which already communicate[134] through other nodes, the "shortcut" is passed through as a new branch in the r-trees of the both netbots as shown on Figure 76. Then, in the same step of the algorithm, each r-tree is depth-searched again to eliminate the dummy links and relocate the remaining branches on a shortcut path. For instance, in our case the link (B,D) is cancelled in the r-tree of netbot A, since there exists a shorter path from D to A. Analogously, the link (B, A) is cut through in the reachability tree of netbot D because D and A are now communicating directly, and not via B.

However, since A leads to C on that path and there is no other way for C to reach D, except through A, so the (A, C) branch is relocated, i.e. expanded, at the newly generated A. The complexity of this part of the WLI algorithm is **O (4).**

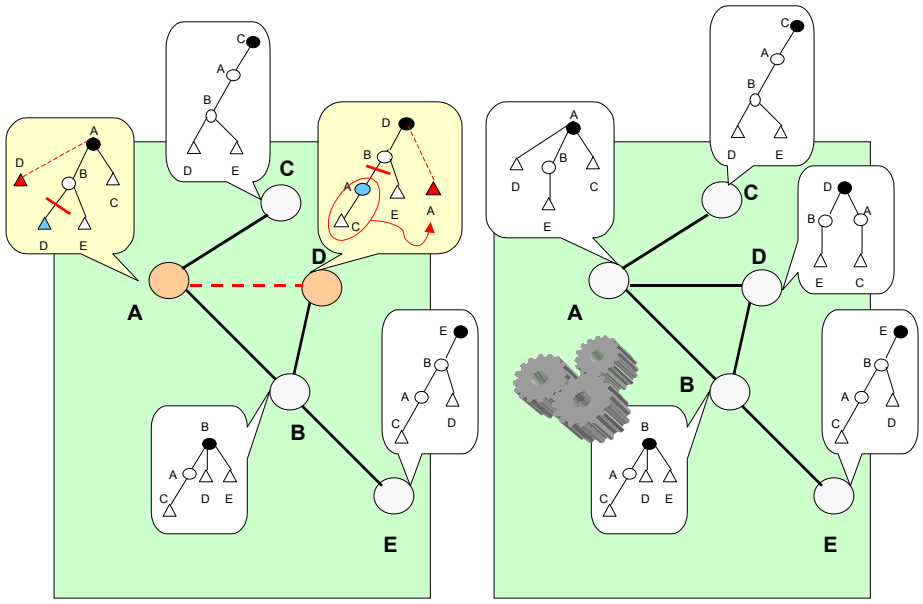


Figure 76: Introducing a Short-Cut

Finally, let us consider the propagation of the r-tree changes when a netbot leaves the fleet for some reason (failure, movement, etc.) as shown on Figure 77. Firstly, the netbot can leave the network gracefully by informing its neighbours for his intention. Secondly, even if the netbot is going to leave the network spontaneously, this case can be reduced to the graceful one.

[134] i.e. both netbots are already present as virtual non-terminals in each others' r-trees

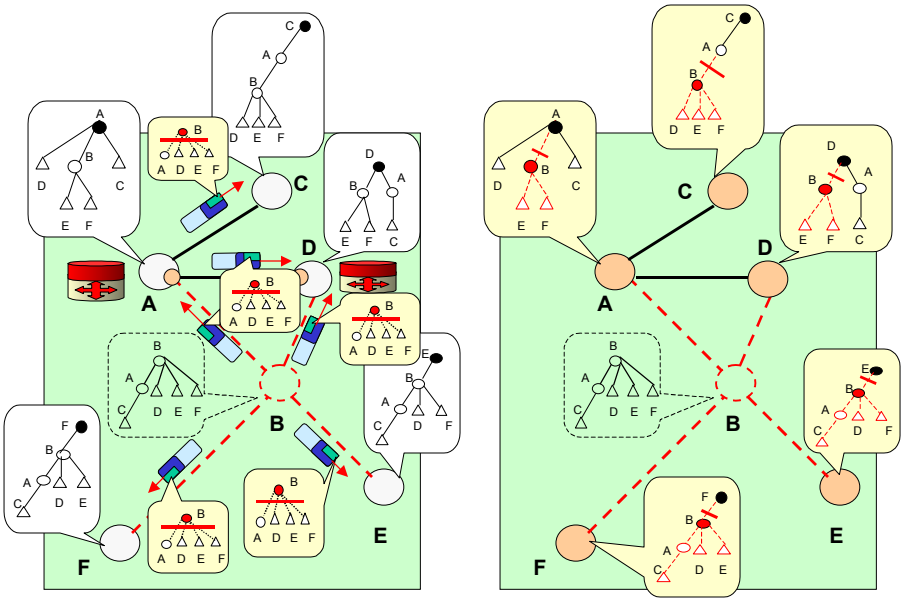


Figure 77: Projection of and capturing the exclusion of an intermediate node

Each netbot can maintain an *alarm shuttle* (a-shuttle) containing an *a*-gene with the first level of the netbot's reachability tree which includes the direct neighbours as leaves. The a-shuttle has a unique identifier that can be recognized by any netbot in the network. When the netbot intends to leave the network, it fires replicas of the alarm shuttle in all directions as a last action before going to inform its neighbours about this event. The a-shuttle is updated as soon as the first level netbot connectivity changes. This function is maintained in parallel with the rest of the netbot's activities and does not require a specific schedule[135].

As soon as the a-shuttle reaches a netbot, its a-gene is unpacked and the local r-tree is updated with the new information. If the same shuttle comes later, e.g. from another line, its content is simply discarded and the shuttle is forwarded to the outgoing lines. Alarm shuttles may implement a TTL data field such as in the common data packets to spare the q-gene and though to limit their circulation in the network.

The complexity of this part of the algorithm is **O (3+m)**, where "m" is the maximum number of hops throughout all r-trees in all nodes participating the network. This value includes the firing of the shuttles (1), the collapse of the r-trees in each netbot (2), and the verification of the r-trees (3), which may take several steps depending on the newly emerged topology of the network, but though regarded by us as a single linear step.

[135] For instance, it can be performed each time a new link is established or an old one is cancelled.

6.3.2.2 The TLA Specification Technique

In the following, for understanding the WLI ad-hoc algorithm specification, we shortly introduce the major concepts of Temporal Logic (TL) and its Lamport's extension, the *Temporal Logic of Actions (TLA)*. Further details about these formalisms can be found in [MaPn92], [Lamp94] and [Lamp01]. Appendix B provides a compact reference guide to TLA.

Temporal Logic (TL) was first defined by J.A.W. Kamp [Kamp68] in 1968 and introduced by Amir Pnueli [Pnu77] in 1977 as a specification language for reactive and concurrent systems. Temporal Logic is an extension to conventional propositional logic, the type of logic that exists in almost any software program or hardware model, such as a C statement:

```
if ((x > 0) && ((y == 2) || (x < z + y))) {...}
```

Propositional logic considers atomic propositions (like `x > 0` or `y == 2`) and operators such as `and`, `or`, and `not`. Temporal Logic introduces temporal operators which allow us to describe the temporal properties of a system. For instance, a requirement like: "`now x>0 and y>0 sometime` (in the future)" can be specified as "`{x>0} && <>{y>0}`". Similarly, the expression "`next x >0 and y > 0 until x == y + z`", can be written as follows: "`{() x> 0} && {y > 0} U {x == y + z}`".

Temporal Logic introduces the following future operators:

- **Next**, written[136] as `()` or as `(+)`, which asserts about a property holding the during *next time* unit (during the next cycle).
- **Always (in the future)**, written as `[]` or as `[+]`, which asserts about a property being true *always* in the future.
- **Sometime (in the future)**, written as <> or as <+>, which asserts about a property being true *sometime* in the future.
- **Until,** written as `U` (e.g. ρ `U` ψ), which asserts about a property or assertion ρ being true until sometime in the future when property ψ becomes true.

These future operators have corresponding past operators:

- **Previous**, written as `(-)`, which asserts about a property holding the during *previous time* unit (during the previous cycle).
- **Always in the past**, written as `[-]`, which asserts about a property being true always in the past.
- **Sometime in the past**, written as <->, which asserts about a property being true sometime in the past.
- **Since,** written as `S` (e.g. ρ `S` ψ), which asserts about a property or assertion ρ being true since sometime in the past when property ψ became true.

[136] Some unary temporal operators such as *Next* may use a prefix and/or a postfix notation as e.g. the prime operator ("`'`") in "`x'`".

Some temporal logics divide temporal assertions about a system's behaviour into *safety* vs. *liveness* properties.

> **Temporal Definition 1:** *Safety* **-** no matter what inputs are given and no matter how choices are resolved inside the system design, the system will not get into a specific undesirable situation, such as emission of undesirable outputs, or undesirable modes of operation being reached.
>
> **Temporal Definition 2:** *Liveness* - some desired configuration will be visited eventually, or some output will be generated eventually.

Safety properties can be used to verify that certain necessary relationships between signals always hold, such as a request signal must always be low for at least one cycle prior to an acknowledge signal going high.

Safety properties are typically described using the Temporal *always* operator (the `[]` box). For example, we can write using the "`()`" next cycle operator:

```
[]({readySignal == 1} → (){ackSignal == 0})
```

and read

always `readySignal == 1` implies next[137] `ackSignal == 0`

The example:

```
[]({readySignal == 1} → (-){ackSignal == 0})
```

is read:

always `readySignal == 1` implies before[138] `ackSignal == 0`

Liveness properties are typically described with the temporal *eventually* operator (the <> box). For example:

```
<>{out1==1} && ()()[]{out2 < 2} && (-){out3==0}
```

is read: "eventually, sometime in the future, `out1==1`, and, two cycles later and thereafter `out2<2`, and a cycle earlier `out3==0`".

Hardware setup and hold properties in Temporal Logic are safety properties. For example, the `dmaRdy` signal requires 1 cycles address setup time:

```
[]{dmaRdy==1} → (-)Stable1(Addr)
```

where `Stable1(Addr)` means that `Addr` is stable for 1 cycle. The above expression illustrates how Temporal Logic caters for assertions about the past as well as the future.

---

[137] i.e. one cycle later

[138] i.e. one cycle earlier

6.3.2.2.1. Temporal Logic and Reactive Systems

There are two categories of system behavior (Figure 78):

(a) *Transformational (Sub-)Systems* are those that have all inputs ready when invoked and the outputs are produced after a certain processing period. Examples of Transformational Systems are data acquisition and multimedia compression systems (in software and hardware) or even a simple computation procedure.

(b) *Reactive (Sub-)Systems* never have all its inputs ready -- the inputs arrive in endless and perhaps unexpected sequences. It is virtually impossible to write a transformational program that implements a controller such as this. In fact, most controllers are by definition reactive, not transformational, with application domains ranging from process control, military, aerospace, and automotive applications to DSP, ASIC design, medical electronics, and similar embedded systems (e.g. a traffic-light controller, the Pathfinder mission).

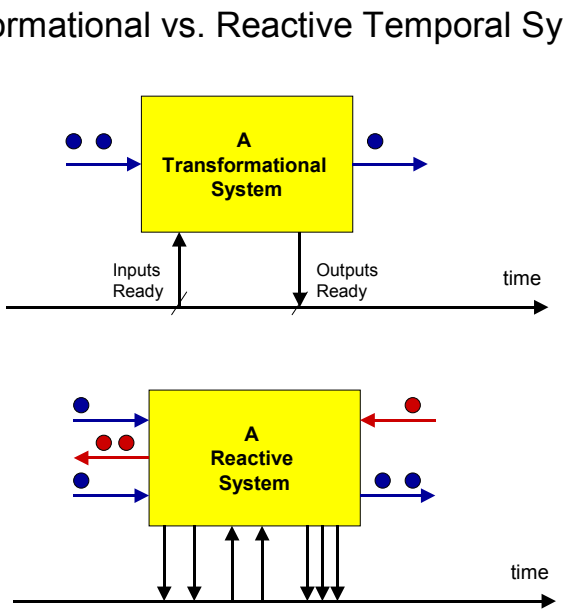


Figure 78: Comparing transformational and reactive systems

Whereas a transformational (sub-)system performs an internal, dedicated, and often time-independent function, almost every system has a reactive component, because it is seldom isolated from the surrounding environment. On the contrary, the reason that the system exists is typically to collaborate or interact with some entity or entities in its environment. Thus, sending, receiving, recognizing and ordering sequences of data such as the behavior of a communication protocol is an event-driven, undeterministic, activity within a reactive system.

Temporal Logic defines a formal way for asserting and specifying program behaviour as a function of time. It also provides the adequate formal means for reasoning about time-critical characteristics of reactive systems. For instance, a traffic-light controller needs to assert:

```
[]((RedOn → <> GreenOn) && (GreenOn → <> RedOn))
```

which reads: “always (i.e., now and any time in the future), either the lights are red, in which case there exists a time further in the future where lights will be green, or lights are green, in which case there exists a time in the future where lights will be red”.

***Statecharts* are often considered as an adequate specification language for automata and hence for communication protocols. In our opinion (which is the commonly accepted opinion in the research community), statecharts are *not* a specification formalism, even though they are a very good high-level design tool.** The difference can be shown in the following example which can be easily described using a TL assertion.

Every time in the future where `isReady==1`, one of the following should be `true`:

a) `x==1` in the preceding cycle, `x==0` in the following cycle, and starting at the following cycle `y==1` until some cycle further in the future where `z==0`.

b) `z==0` since some cycle in the past when `y==1` occurred followed by `y==0` a cycle later.

In temporal logic we can write:

```
[]({isReady ==1}  -> (
   ((-){x==1) && {x==0} && (){y==1} U {z==0})) ||   //   (a)
        ( {z==0} S ({y==1} && (){y==0}) )           //   (b)
  ))
```

One can try to construct the corresponding statechart or state diagram for these assertions. She will realize that she is starting to *implement* the specification using a state-based tool, an implementation which is far more complex (typically exponentially or even larger in size), unintuitive, and error-prone. In short, an implementation is *not* a good specification vehicle.

6.3.2.2.2. Advocating TLA

In the beginning of his seminal paper on TLA [Lamp94], Leslie Lamport argues:

> "*Reasoning about 5000 lines of C would be a Herculean task, but we can reason about a one-page abstract algorithm. By starting from a correct algorithm, we can avoid the timing-dependent synchronization errors that are the bane of concurrent programming. If the algorithms we reason about are not real, compilable programs, then they do not have to be written in a programming language*.”

This introduction clearly identifies the main reason for which we also decided to use TLA to specify formally the WLI routing model.

The referred example in the above citation was the distributed spanning-tree algorithm used in the DEC's Autonet local area network [Schr90]. According to Lamport, the algorithm could be described in about one page of pseudo-code, but its implementation required about 5000 lines of C code and 500 lines of assembly code[139].

The temporal logic of actions (TLA) is both very flexible and adequately expressive. It is a combination of two logics: a logic of actions and the standard temporal logic (TL). TLA is mostly applied to reasoning about concurrent systems (algorithms) which is supported by ordinary mathematics formulas.

A concurrent algorithm is usually specified with a program. Concurrent programs are more complex than simple procedural programming languages. Correctness of the algorithm means that the program satisfies a desired property. The main goal of TLA specifications is to detect algorithmic errors.

In TLA, both systems (algorithms) and their properties are specified by formulas in the same logic. The assertion that a system meets its specification and the assertion that one system implements another are both expressed by logical implication. Thus, the essence of a TLA specification is a theory stating that the correctness of the algorithm implies the property of the system, where *implies* is the ordinary logical implication.

TLA is very simple; its syntax and complete formal semantics are summarized in about a page (Appendix B). Yet, TLA is extremely powerful, both in theory and in practice. It uses ordinary mathematics, plus some temporal logic to express safety and liveness properties, and has the usual modularity, binding and hiding features one needs from a specification language. The ability to express ordinary mathematics without change is very important for real-time behavioural properties of the system such as dynamic QoS parameters. There are no research needs to be done on how to incorporate these in the specification. Lamport has already shown how to treat real-time, and statistical parameters are just defined as mathematical formulas in a TLA module. In this way, all system properties may be expressed as a conjunction of a safety property with a liveness property.

Other formal methods like Petri Nets do not express liveness properties of any specification. Liveness is essential for quality of service (not just liveness but also timeliness!). Process-algebraic approaches do not explicitly express arbitrary liveness properties either. Extensions to both Petri Nets and process algebras to incorporate certain timing features and statistical features have been proposed. However, the advantage of TLA is that we do not need to figure out whether some proposed extension is a good or a bad idea. It can be just put in the model in ordinary mathematics formulae, as it is.

We consider the following **main reasons for using TLA$^+$** as a description technique for the WLI model:

[139] Assembly code was needed because C has no primitives for sending messages across wires.

1. **Formalization**: TLA$^+$ is *a complete formal language*, with a precise syntax and formal semantics. It has two specific characteristics:
    a. TLA$^+$ does not allow the specification of Boolean valued variables to avoid the problem of formalizing Boolean arrays `x[e]` which leads to complicated syntactic and semantic rules and undefined formulas when allowing expressions of the form `[](x[e])`.
    b. TLA$^+$ also does not allow types because they do not lead to completely formal definitions. Instead, Lamport uses *operators* to generate the required data structures.

2. **Realm of Applicability:** Formal languages often have limited realms of applicability. Not all system properties of interest can be expressed in one single language[140]. The TLA$^+$ technique does not have built-in primitives for real-time systems or procedures, but it can easily specify a quite broad class of communicating systems.

3. **Simplicity**: TLA$^+$ is simple enough for practical applications[141]. In addition, TLA$^+$ specifications can be written in ASCII[142]. Hence, we join the Lamport's claim [Lamp93] that a language which can define the Riemann's integral in 15 lines is powerful enough to express any mathematical concepts likely to arise in real specifications.

4. **Practical Orientation:** There are quite many formalisms for system specification, but only a few of them are also practical methods supported by tools. It is hard to define a language that is powerful enough to handle practical problems and yet has a precise formal semantics.

5. **Verifiability:** An important reason for selecting a formalism is *how good it is for formal verification*. A method based on logic has an advantage over one based on another formal language (such as LOTOS and SDL) because one does not have to translate from the specification language to a logic for reasoning. TLA works well in practice because most of the reasoning is in the domain of actions, the realm of "ordinary" mathematical reasoning and because the use of temporal logic is minimal to simplify the reasoning[143]. One reason is that the deduction principle, from which one deduces `P => Q` by assuming `P` and proving `Q`, is invalid for most modal logics[144].

---

[140] For example, the Duration Calculus [RRH93] can define real-time properties, but it cannot express simple liveness.

[141] The specification of a gas burner introduced in [RRH93] and discussed in [Lamp93] requires continuous mathematics such as the definition of the Riemann integral $_a\square^b$ (f) over the closed interval `{a,b}` of real numbers.

[142] Lamport's typesetter program $\mathrm{TLA}_{\mathrm{E}}\mathrm{X}$ [Lamp01] produces pretty-printed TLA$^+$ specifications directly from ASCII.

[143] Modal Logics, such as Temporal Logic and the Duration Calculus, are more difficult to use than Ordinary Logic.

[144] Engberg et al. [Eng92] have found that formalizing temporal logic reasoning to be much more difficult than formalizing ordinary mathematical reasoning. Temporal logic proofs that look simple when done by hand can be tedious to check mechanically. However, they claim that mechanical verification of TLA proofs is feasible largely because it involves very little temporal logic.

By using the TLA technique, it is possible to verify a service component against itself for the properties of *safety* and *liveness*. This is a very important requirement for communicating systems, since it provides a clear statement about the functional correctness of the system before any code is generated. In this way, we spare the phase of software verification and development time to redesign a probably wrong component is saved.

6.3.2.3 Formal Description

Our specifications are based on a few module examples recently published by Lamport in his book draft on TLA[145], [Lamp01]. TLA$^+$ specifications provide a coherent and concise way of communicating a design. They are formal descriptions to which tools can be applied for finding errors in the design and for testing the system. For instance, Figure 79 illustrates the construction of a reusable asynchronous network interface in TLA out of a stand-alone module. The interested reader is further kindly advised to consult the above reference, as well as Appendices B and C of this thesis work for detailed descriptions.

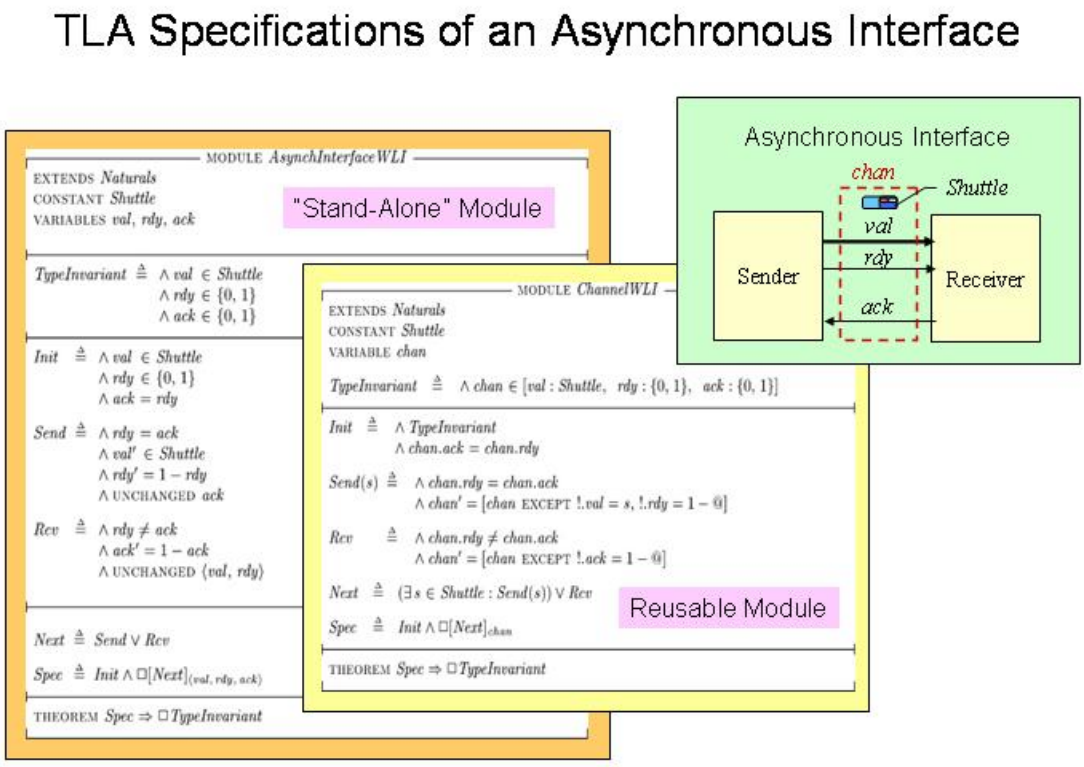


Figure 79: A TLA building block for a network interface

[145] For further details about the Temporal Logic of Actions (TLA), the reader is kindly asked to refer to the TLA web page: http://www.research.compaq.com/SRC/personal/lamport/tla/.

We have generated and verified the WLI routing algorithm with the WinEdt[146] text editor and the TLA tools provided in the above source[145]: the $\mathrm{TLAT_EX}$ typesetter, the SANY Syntax Analyser, and the TLC Model Checker running on Sun's JRE-SE-1.3.1 under Windows 2000. Figure 80 demonstrates the run down of the three phases of model checking for the example in Figure 79 with the TLA toolset: i) syntax analysis (parsing and semantic processing of external modules), ii) TLA type setting, and iii) model checking. In the following, we provide a step-by-step description of the core part of our adaptive routing specification for an active ad-hoc mobile wireless network.

Lamport's TLA Toolset : SANY Syntax Analyser,
$\mathrm{TLAT_EX}$ Typesetter and TLC Model Checker

Figure 80: A trace of the TLA toolset on the WLI-based ad-hoc routing algorithm

Figure 81 illustrates the propagation of the reachability tree information between two netbots in a wandering network. The top part of this figure represents the horizontal network architecture, whereas the bottom's part – a vertical view of it. In each netbot, we clearly identify routing related communication environments (CE) operating upon input and output channels (inChan, outChan) with their FIFO queues (in, out) of length q.

To simplify our model to a level suitable for formalization, we allocated the above entities inside the NodeOS. Nevertheless, a network object such as the CE happens to maintain its image/instantiation at the application level in order to operate with other executing environments. This functionality which may be related to other characteristics of the network such as QoS control and maintenance can be additionally provided to the core description we are currently interested in this case study – the *WLI-based active ad-hoc routing*.

[146] http://www.winedt.com

The essential characteristic of our WLI model is the dynamically adaptable routing algorithm and the propagation of the reachability tree information by means of the so-called *r-shuttles* which contain executable descriptions of the changes inside a reachability tree (structural code) in *r-genes,* as well as the always actual routing descriptions (navigation code) in *q-genes*.

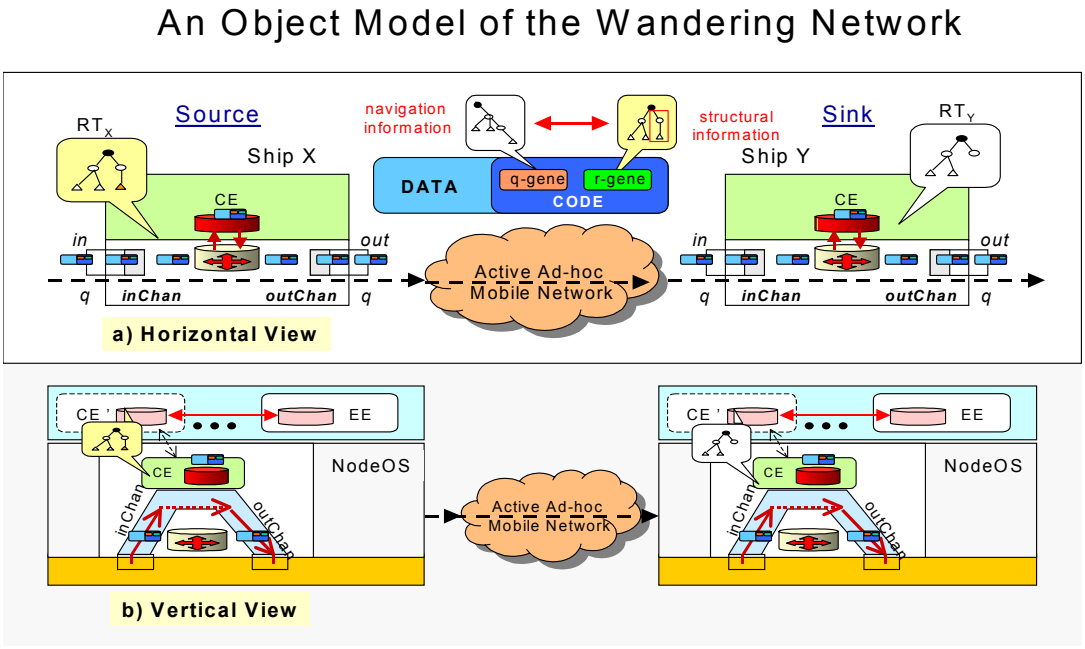


Figure 81: Propagating the reachability tree information in a wandering network

The most important aspect when starting the formal design of a system is the level of abstraction or the "grain of atomicity" and the choice of *what* system changes and *where* they have to be represented as a single step or a sequence of steps in behaviour[147]. TLA$^+$ is particularly effective at revealing concurrency errors. Since the purpose of our specification is to catch errors caused by the synchronous interaction of concurrently executing components, we avoid describing unnecessary details of the data structures.

Figure 82 represents the main netbot abstractions of our routing-related model of the wandering network that are formally described and discussed below in TLA$^+$ [Lamp94].

There are three types of entities we use in this model: a communication channel, an active communication environment (CE), and a reachability tree. Two of them[148], the communication channel and the CE of a netbot can be represented as a set of FIFO queues operating on shuttles as shown on Figure 83.

[147] A coarser- grained specification is simpler than the finer-grained one. However, finer-grained specifications are more accurate.

[148] The reachability tree (RT) and the implementation of the routing algorithm itself with its shuttle processing functionality in the communication environment can be modelled as processor and memory specifications.

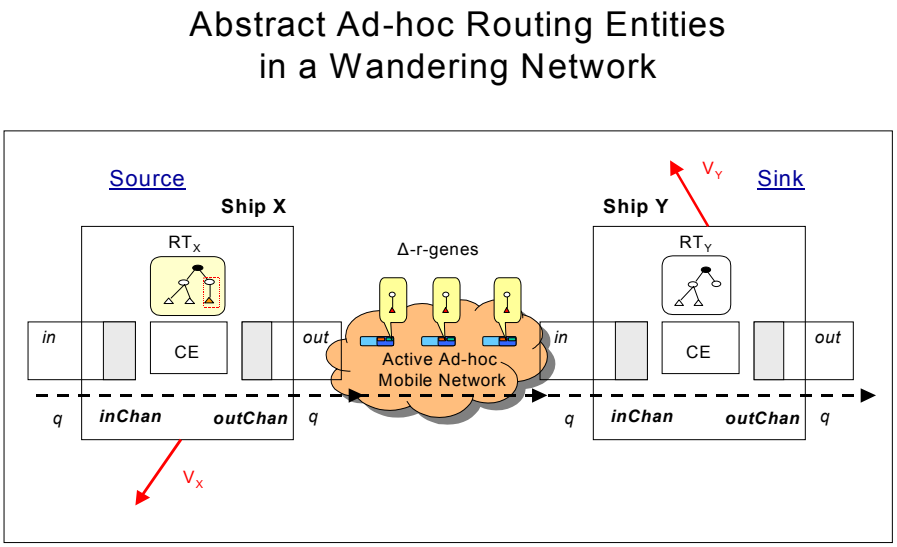


Figure 82: The abstract ad-hoc routing model of a wandering network

According the TLA system model, it consists of three parts:

- a *header*, containing declarations of variables, constants and external interface (extensions[149]) of the specification, as well as a predicate about the initial state of the system optionally followed a type invariance[150] predicate;
- a *body*, consisting of predicates and actions describing state changes of the system which are directly addressed by the next step predicate at the end[151], and
- the *tail*, addressing the entire specification of the module as a single conjunction of the initial state definition and an *always* repeating sequence of next steps introduced by the temporal operator □.

The tail part may also contain additional temporal logic formulas addressing theorems and proof rules for the *safety* and *liveness* conditions of the system specification.

[149] Appendix D provides the complete description of the external TLA$^+$ modules used in this work.

[150] TLA is an untyped logic.

[151] predicates, actions and functions in TLA are defined in a precedence order of appearance.

Modeling A Ship's I/O as a Sequence of FIFO Queues

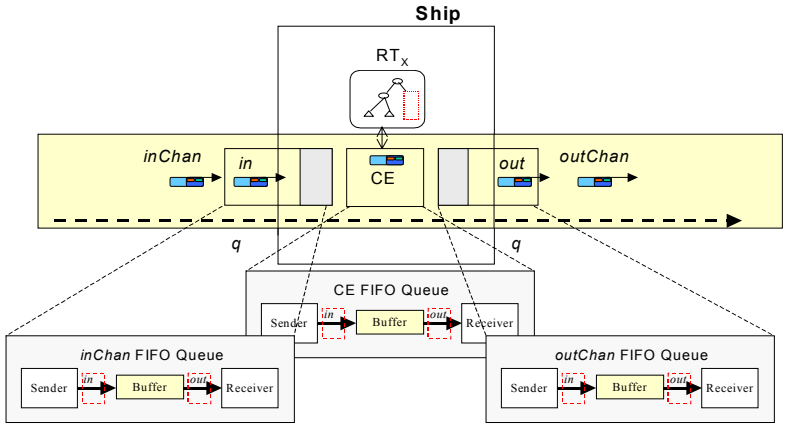


Figure 83: A component based model of a netbot's I/O

The TLA$^+$ specification of a single FIFO queue is given on Figure 84.

```
MODULE InnerFIFOWLI
EXTENDS Naturals, Sequences
CONSTANT Message
VARIABLES in, out, q
InChan  ≜ INSTANCE ChannelWLI WITH Shuttle ← Message, chan ← in
OutChan ≜ INSTANCE ChannelWLI WITH Shuttle ← Message, chan ← out

Init ≜ ∧ InChan!Init
       ∧ OutChan!Init
       ∧ q = ⟨⟩

TypeInvariant ≜ ∧ InChan!TypeInvariant
                ∧ OutChan!TypeInvariant
                ∧ q ∈ Seq(Message)

SSend(msg) ≜ ∧ InChan!Send(msg)   Send msg on channel in .
             ∧ UNCHANGED ⟨out, q⟩                                  SenderAction

BufRcv ≜ ∧ InChan!Rcv             Receive message from channel in .
         ∧ q' = Append(q, in.val)   and append it to tail of q.
         ∧ UNCHANGED out
                                                                  Buffer Actions
BufSend ≜ ∧ q ≠ ⟨⟩                Enabled only if q is nonempty.
          ∧ OutChan!Send(Head(q))  Send Head(q) on channel out
          ∧ q' = Tail(q)             and remove it from q.
          ∧ UNCHANGED in

RRcv ≜ ∧ OutChan!Rcv              Receive message from channel out .
       ∧ UNCHANGED ⟨in, q⟩                                        Receiver Action

Next ≜ ∨ ∃ msg ∈ Message : SSend(msg)
       ∨ BufRcv
       ∨ BufSend
       ∨ RRcv

Spec ≜ Init ∧ □[Next]⟨in, out, q⟩

THEOREM Spec ⇒ □TypeInvariant
```

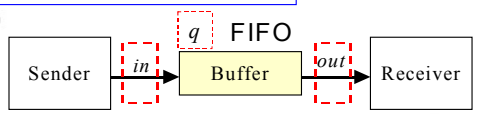


Figure 84: The TLA$^+$ specification of a FIFO queue

The core part of our WLI-based ad-hoc mobile routing algorithm is the specification of the dynamical development and maintenance of a netbot's reachability tree as a function of its own and its' neighbours connectivity to the environment based on the *trusted*[152], instant information exchange among the netbots.

What we are interested in is an effective method to encode tree structures in a linear form in order to implant them as executable, or parsing, information into shuttles which in turn transport this information to other netbots. We selected a LISP like encoding of the tree structure which can be easily parsed and updated on the receiver side. An example of such an encoding and its stepwise development is illustrated on Figure 85.

From step 1 through step 10 the connectivity of netbot/node A is stretched out to other ad-hoc nodes of the WLI network. Every new level of descendants in the tree hierarchy is enclosed by interlocking braces where the children of a parent node are separated by commas. Steps 11 and 12 represent the disconnection of two nodes, C and F respectively, where in the first case the whole C branch is taken out of the A's reachability tree (r-tree).

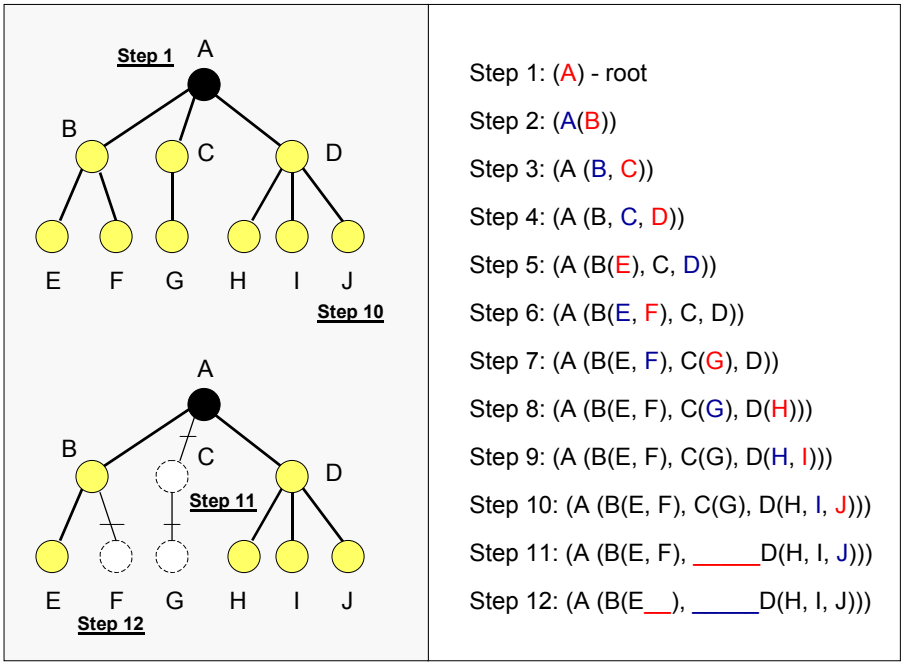


Figure 85: A stepwise linear encoding of a netbot's reachability tree

Next, Figure 86 describes the complete maintenance of a netbot's reachability tree in TLA$^+$.

[152] A Wandering Network is an autopoietic system, and hence by definition, - not a playground for the Byzantine generals problem. Of course, a netbots' communication environment is assumed to provide at least a minimal level of a "sandbox" security prevention.

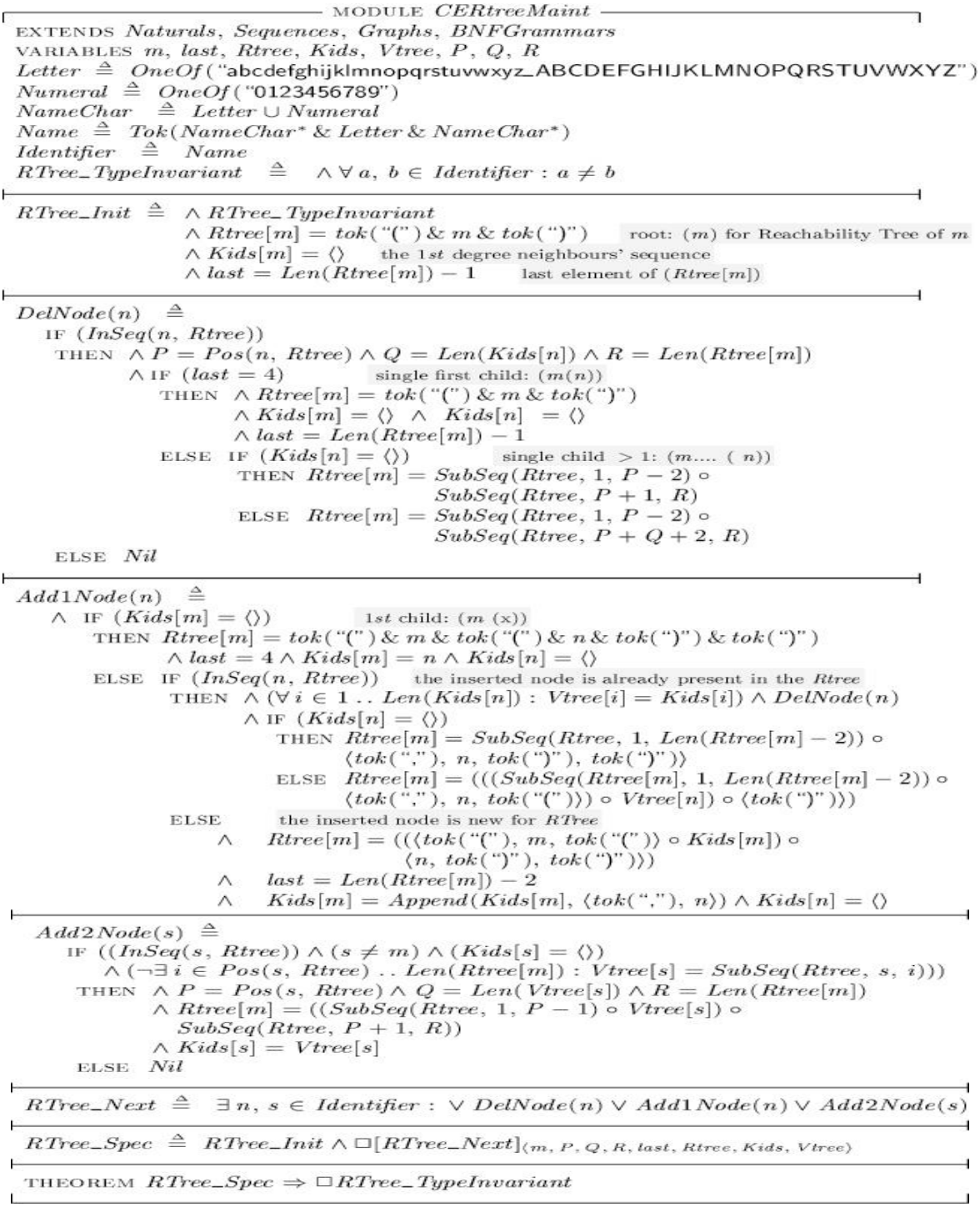


Figure 86: Maintaining a netbot's reachability tree in TLA$^+$

Essentially, the module *CERtreeMaint* (Communication Environment Reachability tree Maintenance) is constituted of the three basic parts of a TLA specification (header, body and tail) where significant descriptions are separated in segments by horizontal lines and comments are given in a grey shadowed background.

The header part extends the standard TLA modules [Lamp01] for natural numbers, sequences, graphs and BNF grammars. It also declares the variables which lead to specifying a netbot's identifier, as well as the type invariance definition stating that no two netbot's identifiers (e.g. IP addresses, or simply names) are the same. Furthermore, the header entails a description of the initial state of the reachability tree, *RTree_Init*, characterized by the conjunction of the type invariance statement, the r-tree "flat-embracing" encoding (token string, Figure 85) *Rtree* containing only the root of the tree, the empty token string of the root's children, and the zero length variable addressing the last child element of the netbot's reachability tree.

The body part of the module is constituted of three action descriptions of the changes to take place in a netbot's reachability tree when a new netbot/node joins the network or an old one leaves it:

- *DelNode (n)* takes an old node's identifier as an argument and removes it along with its descendants (the entire tree branch) from a netbot's r-tree without leaving the empty spaces in the encoded token string shown on Figure 85.
- *Add1Node (n)* takes a new node's identifier as an argument and includes it as a direct descendant (1^{st} degree parenthood) along with its own children (if any) into a netbot's reachability tree. This action is performed after establishing a connection between a new netbot and its "entry point" into the network and exchanging the first r-shuttles between them.
- *Add2Node(s)* takes a new node's identifier as an argument and includes it as an indirect descendant (2^{d} degree parenthood) in some branch of a netbot's reachability tree. This action is performed after obtaining a forwarded r-shuttle with the information of some netbot's branch extension resulting from a new node's joining the network.

The `IF-THEN-ELSE` statements address details of the preconditions required to perform the required changes in an r-tree's structure. The definitions of the unfamiliar operators can be found in the standard basic modules about sequences, graphs and natural numbers in the appendix C of this work.

The module's body is terminated by the *RTree_Next* definition stating that the next state of a reachability tree can be one of the above three actions.

The tail of the specification contains a temporal formula about the entire r-tree specification, *RTree_Spec*, which is represented as a logical conjunction of the initial state RTree*_Init* and an always occurring next state *RTree_Next* where the state variables are given as indexed arguments. Finally, the tail part is concluded by a theorem stating that a correct *RTree_Spec* specification always implies type invariance of the reachability tree.

* * *

A part of the behaviour of a netbot's communication environment w. r. t. realising the WLI routing algorithm, the module *InnerCEInstanced*, is illustrated on Figure 87 through Figure 89. The grey shadowed comments are either explaining some detail of the specification or referring to placeholders for further module enhancements. In the above description, we addressed in detail and for demonstration purposes only the segments of the algorithm responsible for the maintenance of a node's reachability tree.

The introducing section of the specification (A) on Figure 87 begins with the header part as an extension of the previously described module *CERtreeMaint* and the declaration of the variables and constants. In addition, two assumptions have been included as comments, as well as a "ZERO" operator (Z) as placeholder for further system refinements. In the remaining part of the figure, two equivalent descriptions are given: one for the input channel of a netbot's communication environment, and one for its output channel. Each one of these descriptions describes the corresponding instantiation of an asynchronous channel following the standard frame of a FIFO sequence provided in [Lamp01].

The specification continues with Figure 88 and the core operations upon a recognized r-shuttle inside the "buffer" of a netbot's communication environment. The presentation begins with a type invariance statement for the CE's buffer, followed by the standard *RBuf_Init* operator. Characteristic for the evaluation of this part of the description are the transported shuttle tree content, *Stree*, of some netbot's "flatly" encoded r-tree structure or part of it as incremental/decreasing network connectivity, and the shuttle identifier, *Sid*, defining the shuttle as a *leaving* one (fired when a netbot leaves the network), belonging to a netbot's own neighbours, or to some of its neighbours' neighbours of degree ≥ 2. The CE buffer section of the module's body part includes two main actions: 1) fetching the r-shuttle from the input channel, *RShuttle_Fetch (p)*, and 2) update of the netbot's r-tree based on the r-shuttle information from the input buffer channel, *RTree_in_Up*. The result of these actions is summarized as a logical conjunction in the operator *RBuf_Proc (p)* addressing the CE's buffer processing. The Z placeholders refer to actions which are not part of the specification yet.

A few details of the algorithm that should be considered in an extended version of the routing algorithm have been included as comments at the end of the *RBuf* specification on Figure 88.

The final part of the WLI-based CE's routing specification shown on Figure 89 represents a version of the Lamport's FIFO queue description (Figure 84, [Lamp01]) which is enhanced by the reachability tree initiation predicate *RTree_Init* introduced in the *CERtreeMaint* module, and the *RBufExec* action addressing *RBuf_Proc* and the characteristic active network processing inside a netbot's communication environment.

The next step in applying our WLI approach to dynamic routing in ad-hoc wireless mobile networks is the verification of the TLA specification against conditions for safety and liveness (to be defined), and finally - the code generation (Java, C++).

MODULE *InnerCEInstanced*

This module is an extended version of the *Lamport*'s module *InnerFIFO* for use with *TLC*. The current version 1.0 of *TLC* does not handle INSTANCE statements. Therefore, they have been removed and replaced by explicit definitions of the operators originally defined through instantiation.

EXTENDS *CERtreeMaint*

VARIABLES *in*, *out*, *q*, *val*, *type*, *Stree*, *Sid*

CONSTANTS *Message*

$Z \triangleq 0$ The "ZERO" operator: a placeholder for refining the specification

ASSUME 1.: NO CONFLICTING, MULTIPLE OR CONCURRENT SIGNALS AT THE NODE !!!

ASSUME 2.: Event-driven establishment of new contacts to other ad-hoc nodes

---

At this point, module *InnerFIFO* contains the two statements:

InChan $\triangleq$ INSTANCE Channel WITH Data $\leftarrow$ *Message*, *chan* $\leftarrow$ *in*
OutChan $\triangleq$ INSTANCE Channel WITH Data $\leftarrow$ *Message*, *chan* $\leftarrow$ *out*

Therefore, the instantiations below is performed "by hand". However, since "!" can't appear in an identifier name, we use "_" in its place. For example, instead of adding the definition of *InChan*!*Init* to the current module, as does the first INSTANCE statement, we instead add the definition of *InChan_Init* .

Below are all the definitions from Channel, except with *in* and 'out' substituted for *chan* , with *Message* substituted for *Data* , and with "*InChan_*" or "*OutChan*" prepended to the names of all defined symbols.

$$InChan\_TypeInvariant \triangleq in \in [val : Message,\ rdy : \{0, 1\},\ ack : \{0, 1\}]$$

$$InChan\_Init \triangleq \land InChan\_TypeInvariant \land in.ack = in.rdy$$

$$InChan\_Send(d) \triangleq \land in.rdy = in.ack \\ \land in' = [in \text{ EXCEPT } !.val = d,\ !.rdy = 1 - @]$$

$$InChan\_Rcv \triangleq \land in.rdy \neq in.ack \land in' = [in \text{ EXCEPT } !.ack = 1 - @]$$

$$InChan\_Next \triangleq (\exists\, d \in Message : InChan\_Send(d)) \lor InChan\_Rcv$$

$$InChan\_Spec \triangleq InChan\_Init \land \Box[InChan\_Next]_{in}$$

$$OutChan\_TypeInvariant \triangleq out \in [val : Message,\ rdy : \{0, 1\},\ ack : \{0, 1\}]$$

$$OutChan\_Init \triangleq \land OutChan\_TypeInvariant \land out.ack = out.rdy$$

$$OutChan\_Send(d) \triangleq \land out.rdy = out.ack \\ \land out' = [out \text{ EXCEPT } !.val = d,\ !.rdy = 1 - @]$$

$$OutChan\_Rcv \triangleq \land out.rdy \neq out.ack \land out' = [out \text{ EXCEPT } !.ack = 1 - @]$$

$$OutChan\_Next \triangleq (\exists\, d \in Message : OutChan\_Send(d)) \lor OutChan\_Rcv$$

$$OutChan\_Spec \triangleq OutChan\_Init \land \Box[OutChan\_Next]_{out}$$

Figure 87: The routing behaviour of a netbot's communication environment, section A

Below are all the new definitions on operating upon the buffered r-data, with *Message* substituted for *Data* , and with "*RBuf_*" prepended to the names of all defined symbols.

$RBuf\_TypeInvariant \triangleq q \in [val : Message,\ type : \{0, 1, 2\}]$

0: leave shuttle; 1, 2: an own neighbour's & some neighbour's shuttles

$RBuf\_Init \triangleq \wedge RBuf\_TypeInvariant \wedge q = \langle\rangle \wedge Stree = \langle\rangle \wedge Sid = \langle\rangle$

$RShuttle\_Fetch(p) \triangleq \wedge Stree = (Head(q)).val$ the shuttle tree
$\wedge Sid = (Head(q)).type$ the shuttle identifier

$RTree\_in\_Up \triangleq \wedge$ CASE $(Sid = 0)$ the input r-shuttle is a "leave" shuttle
$\rightarrow \wedge DelNode(Stree[2])$ delete the orig. node
$\wedge Z$! forward the "leave" shuttle to the own tree !
$\Box(Sid = 1)$ an input r-shuttle from own neighbour
$\rightarrow \wedge Add1Node(Stree[2])$ add $1st$ deg. node in *Rtree*
$\wedge Z$ send the own *Rtree* (if any) to the new node!
$\wedge Z$ send the new node link along the own *Rtree*
$\Box(Sid = 2)$ input r-shuttle from a $2d$ neighbour
$\rightarrow \wedge Add2Node(Stree[2])$ add $2d$ deg. node in *Rtree*
$\wedge Z$ forward the new node link along the own *Rtree*

Timed *RTree* update: if there is no alert shuttle from each direct neighbour within a given time window consider a Disconnect procedure: $Sid = 0$

$RBuf\_Proc(p) \triangleq \wedge RShuttle\_Fetch(p)$
$\wedge RTree\_in\_Up$ passive *CE* state changes via processing input shuttles
$\wedge Z$ $RTree\_out\_Up \rightarrow$ active (internal) Rtree updates
$\wedge Z$ Refresh $\rightarrow$ send periodically alert shuttles along *Rtree*

The following actions should be considerend in a detailed specification of the algorithm:

I. ACTIVELY PERFORM A *CE* STATE CHANGE AND SEND AN R-SHUTTLE $\Rightarrow$
include new action *RTree_out_Up*.

II. CONNECT_A_NODE:

1.) establish a contact to a new ship
2.) send a shuttle with the own *Rtree* to the corresponding node
3.) update the own *Rtree* with the new ship/node: *Add1Node*
3.) forward the change along the own *Rtree*.

III. DISCONNECT_A_NODE

1.) remove a node and its children (if any) from the own *Rtree*
2.) send a single "leave" shuttle to the corresponding ship/node
3.) forward the change along the own *Rtree*.

IV. SELF-INITIATED "Leave"

1.) fire a "leave" shuttle along the own *Rtree* and leave the network
2.) clean its own *Rtree* except for its own presence: {m}

Figure 88: The routing behaviour of a netbot's communication environment, section B

Further Detail A: $RBuf\_Next \triangleq (\exists\, d \in Message : RBuf\_Send(d)) \lor RBuf\_Rcv$)
Further Detail B: $RBuf\_Spec \triangleq RBuf\_Init \land \Box[RBuf\_Next]_q$

The rest of the module is almost the same as module *InnerFIFO*, except for the new *CE* logic (*RBufExec*) and that each "!" is replaced by "_".

$$Init \triangleq \land InChan\_Init \land OutChan\_Init \land RBuf\_Init \land RTree\_Init$$

$$TypeInvariant \triangleq \land InChan\_TypeInvariant \land OutChan\_TypeInvariant \land RBuf\_TypeInvariant \land RTree\_TypeInvariant \land q \in Seq(Message)$$

$$SSend(msg) \triangleq \land InChan\_Send(msg) \land \text{UNCHANGED } \langle out, q\rangle$$

Send *msg* on channel *in* .

$$RBufRcv \triangleq \land InChan\_Rcv \land q' = Append(q, in.val) \land \text{UNCHANGED } out$$

Receive message from channel *in* . and append it to tail of *q*.

$$RBufExec \triangleq \land RBuf\_Proc(Head(q)) \land q' = Tail(q) \land \text{UNCHANGED } \langle in, out\rangle$$

Execute an operation upon analysis of the shuttle.

$$RBufSend \triangleq \land q \neq \langle\rangle \land OutChan\_Send(Head(q)) \land q' = Tail(q) \land \text{UNCHANGED } in$$

Enabled only if *q* is nonempty. Send *Head*(*q*) on channel *out* and remove it from *q*.

$$RRcv \triangleq \land OutChan\_Rcv \land \text{UNCHANGED } \langle in, q\rangle$$

Receive message from channel *out* .

$$Next \triangleq \lor \exists\, msg \in Message : SSend(msg) \lor RBufRcv \lor RBufExec \lor RBufSend \lor RRcv$$

$$Spec \triangleq Init \land \Box[Next]_{\langle in,\, out,\, q\rangle}$$

THEOREM $Spec \Rightarrow \Box TypeInvariant$

Figure 89: The routing behaviour of a netbot's communication environment, section C

6.3.3 CONCLUSIONS

Ad-hoc networks provide a completely new way for realizing mobile communication when no fixed infrastructure is available. Path selection is one of the key points in designing such networks, since there is no base station as in cellular networks, which can access all other stations via broadcast. Traditional routing algorithms are not very effective in ad-hoc networks with dynamic topologies. For this reason, existing algorithms has to be expanded or new algorithms developed. Big clusters of mobile nodes are usually supported by hierarchical approaches in routing, since algorithms like DSDV cannot scale very well. In particular, the transmission benchmarks in wired and wireless networks such as authentication/security, timing and QoS differ for knowledge about the characteristics of the lower layers. For instance, information about interferences at the physical and the data link layers can essentially contribute to finding a suitable route between two mobile stations. This task becomes even more difficult to treat when the fluctuations in node connectivity (e.g. channel and bandwidth availability) spontaneously change. Therefore, adaptive routing algorithms, which are capable to adjust to changing environmental conditions in order to maintain a negotiated QoS, are highly desired in ad-hoc networking.

In this chapter, we demonstrated the application of the WLI approach for autopoietic (self-creating) reachability tree maintenance to support adaptive routing in active mobile ad-hoc networks. We proposed Wandering Network architecture, a methodology and a high-level distributed routing algorithm, which fulfill the requirements for fast update and propagation of the connectivity information in a dynamically changing environment. This pure application layer algorithm does not consider any low layer feedback information such as wireless channel interference labeling and weighting to select a route. It is not an optimization algorithm either. The WARAAN algorithm solely demonstrates the applicability of active network technology, extended by the WLI framework, to maintaining and distributing connectivity information along existing links in a dynamically changing network topology. The algorithm was tested for correctness and can be implemented next in an arbitrary distributed programming platform.

Nevertheless, it is wrong to expect that there will be a universal ad-hoc routing algorithm for any kind of application networking scenario with minimal computing overhead and maximum performance and reliability. Much more, we should work towards an expanding and on-demand configurable set of routing plug-in modules reflecting the particular conditions of the network. Therefore, in order to provide a custom solution which includes additional networking information such as the direction/symmetry, synchronicity or the quality ranking (interferences, noise/signal ratio) of a connection, the r-tree update algorithm, and hence a netbot's CE functionality, presented on Figure 87 through Figure 89 have to be extended by further modules addressing the specific features of the desired methodology which can be activated on demand or when certain conditions are fulfilled.

Active networking, the WLI model frame and the TLA formal technique provide excellent means for generating solutions in such a complex application domain as mobile ad-hoc networking. The *autonomous dynamic updates* of a netbot's reachability tree are only an example for the suitability of the WLI approach to addressing autopoietic processes in evolutionary networking.

6.4 SUMMARY

Science is a method of perceiving, describing and verifying. In this regard, it also determines how we see the world and what we reason about it. In that the perceiver in science is a human being, by virtue of our sensory organs, there are limitations constraining what can be and is seen at any given time. As such, science changes as our ability to perceive changes. As technology and engineering are our natural means to expand our senses and explore the world, they enable science or development of thought. Science in turn generates a new order of things and results obtained by technology; it provides the new goals and directions.

Until recently, both software and hardware were separated concepts. The design principle was: "Stop the hardware to run/configure the software, or stop the software to run/configure the hardware", at least in terms of *evolutionary* technology. This is because software is more flexible and hardware is more powerful (by default). Active networking and configurable computing already began to integrate the design in parallel networking solutions. The advantages of these approaches have been presented in the previous sections. Doubtlessly, top-front research areas such as *software radios* and *virtual hardware* are very challenging. Yet, at this point, we started asking WHY and HOW.

Therefore, we defined our future scientific task as to discover and keep track of the *synthetic patterns of networking*, both in software and hardware design, by using formal methods and developing their apparatus.

We defined the *Wandering Logic Intelligence (WLI)* as an evolving model of Wandering Networks. WLI generalizes AN capsules in *shuttles* as relatively autonomous mobile components including both programs and data possibly encoded in a language with (semantic) references[153] to netbots and other shuttles within the same or a different domain/flow (protocol).

The following enhancements of the traditional AN models are achieved with the Wandering Network approach:

- Active nodes may be mobile, - hence the name *netbots* -, and re-configurable (in terms of software and hardware). In addition to traditional active nodes, shuttles can also *modify*[154] netbots.

- A netbot's runtime re-configuration can be invoked by internal procedures or upon execution of newly arrived shuttles. Autonomous mobile hardware components (*netbots*) take care for delivering their own "driver" routines (mobile code) at "docking time" on the netbot.

[153] This language should be capable to address in a uniform way even such issues as hypermedia content (e.g. MPEG-4/7) and related knowledge-based management systems along with the corresponding encoding/decoding routines or references to them in dedicated active network nodes or protocols.

[154] The capsule APIs and the execution environments can be extended by special routines allowing the accommodation and execution of code that changes a netbot's configuration and resources. In this way, new functionality can not only be delivered to and injected into the active node, but also distributed and optimized throughout the node itself.

- Active packets are called *shuttles* and carry code and data for the upgrade/degrade and re-configuration of netbots. In addition, shuttles can carry genetic information about the netbots' architecture and their communication patterns.

- A *code distribution mechanism* ensures that shuttle-processing routines are transferred automatically and dynamically to the netbots where they are required. In a Wandering Network, code distribution throughout the network and inside the netbots can be *maintained by the shuttles themselves*.

In addition, the WLI model allows the creation of new capsules/shuttles (or the replication of "old" ones) in the intermediate active nodes under the supervision of the NodeOS. In addition, a special class of shuttles, called *pilots* are allowed to replicate themselves and to *create/remove/modify other capsules and resources* in the network.

The essential contributions of the WLI approach in this work are listed as follows:

- *Role Change*: The *role* of the network node within a particular virtual architecture can change during its operation. The new functionality is either resident on the node and waiting to be activated, i.e. it is not yet involved in the next step virtual scheme, or *transferred* via Active Networking to the destination node.

- *Parallel Roles*: The execution of the parts of a distributed algorithm can be performed within the different roles of an active node's/ netbot's configuration.

- *Node Genesis ("N"-geneering):* encoding and embedding the structural information about a mobile node, the netbot, and its environment, such as e.g. rooting tree information, into a secondary level of virtualization of the active packets/ shuttles composed of *n*-genes

We hope that with this work we were able to describe what we have perceived about the nature of the phenomenon called networking. In fact, most of the ideas appear to be familiar and intuitive to anyone. Yet, it is difficult to specify and arrange them in a beautiful model, and then start investigating it by asking questions, hypothesizing and providing both experimental and theoretical proofs to fill up the gaps of such an overall principle as *active information* (to quote David Bohm) when applying it to recent developments in communication technology. In the next chapter, we will provide the formal specification and the verification of the WLI model within the context of an UMTS engineering scenario.

Yet, a lot more has to be done from now on to stabilize the base of this emerging formal explanation of networking w. r. t. the empirical results provided by *enabling engineering technologies* such as Java/JINI, CORBA, TINA and WAVE ([Sap96], [Sap99]), and to prove the feasibility of this approach in a series of experiments. Perhaps we should start first with integrating concepts from "more humanitarian" computing disciplines such as artificial intelligence and neuroscience, which already have their own methods of approaching network intelligence. This is why we finish this chapter with citing Kepler: "All scientific statements *must* be testable by observation."

CHAPTER 7: EVALUATION AND OUTLOOK

"... mathematical logic itself looks like a language that is naturally capable of evolution. ... In this context, ... it seems possible that a system of logic of the future could be translated into a form corresponding to a system of the present time with the addition of a few axioms that give what is needed to give the potentialities of the future system."

John F. Nash Jr., *Hierarchical Introspective Logics*
http://www.math.princeton.edu/jfnj/texts_and_graphics/LOGIC/talk.CMU/

In this chapter, we are going to discuss a few interesting research issues in Active and Programmable Networking, which are closely related to the WLI model and its implementation. System layering and *end-to-end arguments* (E2EA) are common design guidelines which are widely accepted in present day networks. In Chapter 5 we have provided four additional principles for evolutionary design associated with the Wandering Network. Therefore, section 7.1 is devoted to the principles of network design and discusses the WLI implementation framework w. r. t. system layering, E2EA and the WN principles. Next, section 7.2 reviews the practical realization of the Wandering Network. It presents a possible scenario for the WLI evolution from present day programmable architectures followed by a discussion about the realization of the shuttle addressing scheme. Then, section 7.3 provides a summary and conclusions of the thesis by defining the network technology interfaces required to open the path towards WLI architectures. Finally, section 7.4 concludes this chapter with directions for future research.

7.1 EXTENDING THE PRINCIPLES OF NETWORK DESIGN

Communication networks are layered systems with physical transmission links as their lowermost layer. While most of the networks in use today have been developed independently and without a common model, their various layers can be reasonably mapped onto the seven-layer Open Systems Interconnection (OSI) Reference Model proposed by ISO, [OSIRM]. Layered system design is an approach of designing a large system by partitioning its functions into a hierarchy of layers. It provides important advantages to the system designer such as separation of concern, multiple levels of abstraction and layer isolation which simplify the complexity of large systems.

On the other hand, discussions of the implementation of various functions in a communication network often include some form of an "*end-to-end argument*" which is in fact a set of architectural principles that guide the placement of functions within a distributed system [Saltz81]. Such principles are often interpreted *to prevent the implementation of any kind of higher-level function within a network*.

The layered system model and the end-to-end arguments have been helpful in the design of modern layered protocols. In the context of active networks, however, which take the non-traditional view of a programmable network infrastructure, the interpretation and application of these design principles is an open question. In this section we are going to answer three important questions:

1. Do active networks fulfil the *End-to-End Design Arguments ?*
2. Does an active network violate the layering principles ?
3. How can we consolidate classical network design with active networking ?

7.1.1 INTRODUCTION: DESIGN MODELS

In the following, the architectures we talk about represent theoretical models, recommendations and guidelines which are and can be particularly implemented in practice.

A system is expected to enable the application to deliver the required application by the user *efficiently*. Therefore, the design of a system is a significant factor influencing its applications' efficiency. It is desirable to divide the system into manageable parts to reduce its complexity. Layered system design is one such way of decomposing a large system by partitioning its function into a hierarchy of layers such that the functions at layer *N* are implemented in terms of the functions provided by layer *N-1*, with the bottom layer being the basis of this vertical recursion, Figure 90. Each layer in the hierarchy typically provides its function at a higher level of abstraction than the layer below. The layered approach provides such advantages like separation of concern, multiple levels of abstraction, and isolation among layers.

In general, system layering serves is a good design model. However, there are situations where a particular layered system design may have limitations which can restrict application efficiency. In this context, it is important to note that while layering is critical to the description of protocols and protocol families, it is not necessary for implementation, and indeed may be harmful to a high-performance implementation. In such cases, applications naturally expect the system function to be modified or customized to deliver better application efficiency. Therefore, a system's implementation may occasionally violate model-based system layering in order to address a system function efficiently. This holds not only for current proprietary products (e.g. routers by Cisco, 3Com, etc.), as well as for prototype solutions such as implementations of the DARPA's Active Network program, and of course for the WLI framework presented in this thesis work.

The reason for this can be e.g. the additions to a piece of data as it passes down the protocol stack. If the implementation is strictly layered as illustrated on Figure 90, then the user data has to be copied three or more times in the process. A truly efficient implementation can generally come along only with one copy, RFC 817. In UNIX terms, this should also be the copy from user to kernel space. For instance, in a special-purpose router it is normal to arrange that most packet data be never copied, at least for straightforward cases.

Another example of the violation of layering for performance, in this case overhead minimisation is given in the header compression. For instance, a compressed 256+ header byte packet has 1.9% header overload, whereas an uncompressed 256+ header byte packet has 13.5% overhead. Therefore, this method is very valuable. Nevertheless, it is a *complete violation of the layering principle*, since the CSLIP/PPP implementation has to look up the protocol stack to the IP and TCP layers to perform the compression. (In fact, only TCP, which is connection-oriented, benefits from this compression. NFS over UDP, which sends many packets to and from the file server, does not, since although there's a logical connection, there isn't one at the UDP level, since UDP is not connection-oriented.) Furthermore, firewalls and proxies also tend to violate not only the “network layering”, but also the “end-to-end principle” as do some routers.

In fact, from the viewpoint of implementation, Active Networks and WLI also tend to violate the layering principle, which is also usual for other techniques practiced today. Nevertheless, as a (formal) specification approach, both approaches underlie the same common rules which open systems have been designed until now.

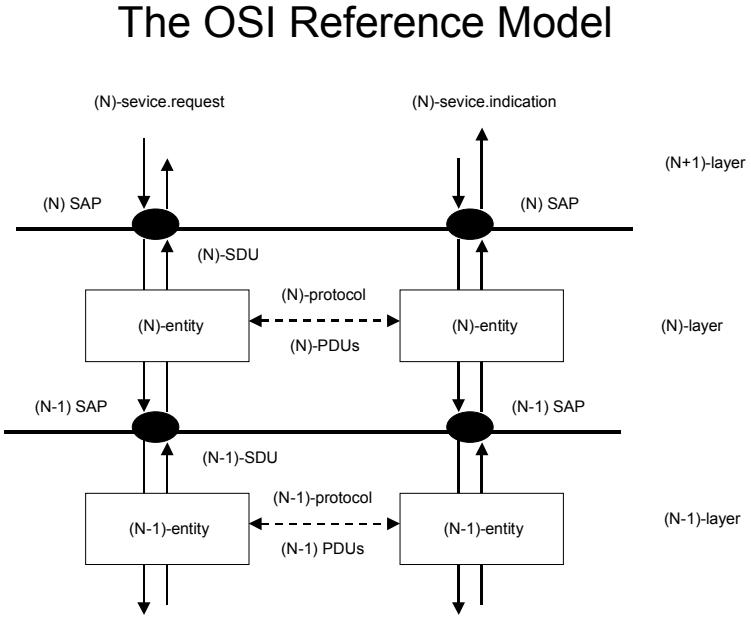


Figure 90 Protocol entity mappings in the OSI-RM

In a study on the interaction between layered system design and application efficiency spanned over the domains of computer architecture, computer operating systems and communication network, performed by Sawant [Saw01], the analysis shows that *better application efficiency is achieved if the system exposes its low level function to the application, and allows the application to do the customisation suitable for its requirements*.

This result is known as the “API framework” (API = Application Programming Interface). In his thesis, Sawant demonstrates that the End-to-End Arguments principle provides a very practical framework and a useful set of criteria to determine *which functions to be placed a priori at lower layers for better application efficiency*.

This is another welcome argument in favor with the present-day programmable active networks paradigm. However, API’s are usually placed in the higher system layers and are therefore in a way “pre-programmed”, and thus fixed, non-extensible which requires careful function placement at the lower layer a priory to programming it.

In order to be able to accommodate any new application efficiently and to maintain it during the lifetime of a system, it is necessary:

- to program deeper layers of the system design (including the bare machine, the hardware), and
- to reconfigure the function placement during deployment of the system.

Therefore, the WLI approach addresses these two issues in a straightforward manner. In the following, we discuss layered system design in some more detail w. r. t. Active Networks and WLI.

7.1.2 *THE HORIZONTAL STATICS* OF NETWORK DESIGN: END-TO-END ARGUMENTS

The end-to-end arguments are system design principles intended to help determine where to place services in a subsystem. The key points of the classical end-to-end arguments can be summarized as follows[155], [Saltz81]:

1. ***Allocation (A1)*** **–** A function should be placed at a lower layer only if it can be completely defined at that layer.

2. ***Adequacy (A2)*** **–** A lower layer implementation of a function, based on incomplete information about upper layers’ requirements, can turn out inadequate.

3. ***Arrangement (A3)*** **–** Partial implementation of a function in a lower layer for performance reasons is allowed, if this layer substantially improves the performance of other parts of the system. The design must be done carefully because,
    a) the lower layer may not be able to achieve the performance target for the reason given in A1.
    b) the lower layer being common to many applications, the applications not needing the function can end up paying for it anyway.

[155] The naming was provided by the author for the purpose of comparison, s. ff.

At first sight, it might seem that network programmability is the true antithesis of the end-to-end arguments which state that "a function or service should be carried out within a network layer *only if it is needed by all clients* of that layer, and it can be completely implemented in that layer"; hence the assumption that adequate function placement is a design issue rather than a deployment characteristic.

End-to-end arguments address design more than implementation and implementation more than execution; they suggest *who* should provide the code, not *where* it should run. On the other hand, programmability may allow a network client to implement precisely the service it needs even in a short-term planning, an outcome that is consonant with end-to-end arguments. Therefore, according to Saltzer, Reed and Clark [E2EC], applying end-to-end arguments to network programmability in a general, yet definitive, way may be impossible, because of the wide range of realization possibilities. Instead, they suggest that the specifics of each particular active networking idea would benefit from evaluation in light of the end-to-end principle. Thus, "activation" and programmability of networks can be also regarded as a natural extension of these well-accepted design principles.

In particular, we consider the following arguments, [BCZ97C]:

1. *In active networks*, *the network service can be tailored to the user's requirements*[156]. By definition, the end-to-end argument views the network as a monolithic entity that provides a single type or quality of service to all users, i.e. either reliable or best-effort transport. Active networks allow service customisation by providing an interface which supports multiple configurable or programmable services. Originally, "active networking" refers to the placement of *user-controllable computing* and other resources in the communication network, where they can be utilized by applications that need those capabilities [Tenn97]. An active network supports a user–network interface allowing the nodes of the network to be *programmed* by the user/application to provide a desired functionality, such as routing. The level of programmability might range from a Turing-complete programming language to a set of predefined, user-selectable functions whose behaviour can be controlled through parameters.
2. *An "end-to-end argument "provides a rationale for moving a function upward in a layered system closer to the application that uses the function" [Saltz81].* According to this principle, a computer network, as part of the "lower layers" of a distributed system, should avoid[157] attempting to provide functions that can be better implemented in the end systems, especially if some applications might not benefit from such functions at all. The classical example of such a moving function is *reliable transfer*. The network can go to great lengths to protect against and recover from losses in the network, but an application that requires reliability will generally have to protect against other sources of error; in that case the network's efforts are redundant. On the other hand, applications that do not need reliability would still have to pay for it (e.g., through reduced throughput and latency).

[156] This aspect of active networks relates to end-to-end arguments in general.

[157] This argument is controversial to the Intelligent Network (IN) model, [Fayn97], in circuit switched telecommunication networks.

3. *It is desirable to combine application and network information to optimise application performance.* Some services can best be supported or enhanced using*:*
    - ***a)*** *information that is only available inside the network, i.e*. the network may have information not available to the application, and the timely use of that information can significantly enhance the service seen by the application. Examples of information that is first (or only) available to the nodes of the network include:
        - the time and place where congestion occurs, [Tenn97];
        - global patterns of access to objects retrieved over the network (e.g., Web pages); in particular, the location of "hot spots," or points in the network where requests for objects are highly correlated in time and space, [ClTe90];
        - the location of packet losses within multicast distribution trees.
    - ***b)*** *information that is only available in the applications,* i.e. applications may have information that is needed by the network in order to fully optimise performance. Examples of this type of information include:
        - existence of *dependencies among application data units* (e.g. audio and video in a MPEG stream);
        - *variations in importance of data units*, incl. retransmission if lost;
        - other transmission context information such as whether or not it is acceptable to service a request using cached data.

The essence of a good end-to-end argument is that the **performance cost** of *using* an interface should vary with the application. For example, applications that need only a best-effort datagram delivery service should not suffer reduced performance because of the increased flexibility of the interface. On the other hand, any performance penalty for customizing network behaviour (e.g., signalling overhead, or taking packets off the "fast path") must be more than offset by improved end-to-end performance delivered to the ultimate users. These performance costs will be determined ultimately by the primitives and composition mechanisms provided by the active network architecture. Therefore, Bhattacharjee, Calvert, and Zegura identify the following principles as the key end-to-end arguments applying to the placement of functionality in [active] networks, [E2EC]:

1. ***Active Allocation (AA1) –*** Some services require the knowledge and help of the endsystem-resident application or user to implement, and thus *cannot* be implemented entirely within the network.
2. ***Active Adequacy (AA2) –*** *{absent}.*
3. ***Active Arrangement (AA3) –***
    - The amount of support for any given end-to-end service in the network is an engineering trade-off between the *performance* seen by the application and the *cost* of implementing the support. However, these principles do not rule out support for higher-level functionality within the network. Rather, they require that the interface to such functionality be carefully designed; that costs and benefits of such support be calculated; and that the engineering trade-off be carefully evaluated.
    - If not all applications will make use of a service, it should be implemented in such a way that only those applications using it have to pay the price of supporting it in the network.

7.1.3 THE *VERTICAL STATICS OF NETWORK DESIGN*: SYSTEM LAYERING

The layered model has the following qualities which make it a helpful system design tool:

1. *Separation of concerns.* Each layer provides a solution to a certain part of the overall problem, and the layer above it can build upon that solution to address another part of the problem. For instance, the lowermost layer in a communication network provides the physical transmission link and the data link layer makes this link appear free of transmission errors. Also, the lower layer of an operating system can provide a generic, secure mechanism for processor control transfer which can be used for implementing various scheduling policies at a higher layer.

2. *Successively higher levels of abstraction.* A network protocol stack converts a physical bit transmission facility into a communication channel for exchanging application level data units. Computer architectures can be utilized at increasing levels of abstractions via machine language, assembly language and high-level language.

3. *Flexibility through multiple alternatives.* An upper layer can provide various abstractions for a specific function from the layer below. A transport layer can provide connection-less and connection-oriented services using a single network-level packet delivery service (e.g. UDP and TCP in the TCP/IP Internet architecture.) Applications can choose from various high level languages supported on particular computer architectures.

4. *Isolation*. Ideally, a change made to layer *N* would, at most, cause changes to layer *N+1*. All the layers above layer *N+1* are isolated from these changes. The network layer of the TCP/IP network model isolates the transport and application layers from underlying physical network technologies. The machine language of a microprocessor architecture isolates the object code targeted for the architecture from the changes to its implementation. A Java bit code isolates the application from the underlying architecture.

System implementations can benefit from using layering to exploit the above qualities of the layered model. However, there are situations, when a layered design can restrict or reduce application efficiency. The loss of efficiency can have various forms such as reduced flexibility, lost of performance, or even lack of function, and it is commonly attributed to causes like the cost of indirection resulting from *(inadequate) abstractions* which may not reflect the specifics of the particular implementation.

For instance, to ensure *secure operation*, operating systems commonly restrict applications' access to OS resources and services through the *system call* interface.

The applications are restricted to use the abstractions provided by this interface and are required to pay the cost of the indirection. This can lead to loss of performance and reduced flexibility for some applications. Besides, if these abstractions are not adequate for an application, this restricted access can mean a choice between using an inadequate function or a complete non-availability of the function. Another example is the *wireless access*. It is characterized by a completely different behaviour of the network when compared to the wired Internet.

Therefore, the main reason for *QoS degradations* is that **the existing Internet traffic control mechanisms such as TCP and UDP were primarily developed for the use in wired networks**. Therefore, the wireless part of the packet network expects a different traffic control which has to be treated separately in order to address the specifics of the wireless link. The Wireless Access Protocol (WAP) is such an example. It breaks the IP net in two parts and requires a complete exchange of the Internet protocol suite and applications which leads to performance degradation. Besides, WAP is completely orthogonal to the end-to-end IP QoS concept. It cannot provide a lasting solution.

The more fundamental cause of the reduced application efficiency is usually **the lack of access to the low level system function**. The above mentioned causes of lower efficiency can be addressed if the application has access to the low level system function. A system which dynamically extends the low level function for better application efficiency by application-specific customisation, but without tightly binding of any application specific function into the lower layer, is highly desirable.

The layered model represents a bottom-up design. Any high level abstractions provided by a system represent its *assumptions* about application requirements. Therefore, it is reasonable to provide *default* abstractions which are carefully designed to satisfy the requirements of a large number of applications and save them the effort of developing their own abstractions. The widely used TCP/IP Internet protocol suite provides end-to-end transport abstractions for best-effort network service illustrates this point.

On the other hand, when the default high-level abstractions are not adequate, the system should enable applications to develop their own abstractions. These abstractions can hope to be as effective as the default abstractions only if they have an access to the same low level system function as the default abstractions.

Finally, the lower layer functions exposed by system have to be generic enough, and do not make any application-specific assumptions, otherwise we face recursive design.

7.1.4 CONCLUSIONS

The results of the previous two sections can be summarized as follows:

The basic premise with the classical end-to-end arguments and the layering principles for system design is the idea that **a lower layer should support the widest possible variety of services and functions to permit applications that cannot be anticipated at design-time**. In other words, minimize the lower-layer functionality through selection of the most common transport characteristics, take processing out of the way, and let the higher layer do its application adaptation. This widely accepted approach implies that end-to-end arguments have two complementary goals which manifest the organizational differentiation between higher and lower system layers:

- *Higher-level layers*, being more specific to an application, are *free* (and thus expected) to organize lower-level network resources to achieve application-specific design goals efficiently (application autonomy)
- *Lower-level layers*, which support many independent applications, should provide *only resources of broad utility across applications*, while providing to applications a usable means for effective sharing of resources and resolution of resource conflicts (network transparency).

Thus, moving functions and services upward in a layered design, closer to the application(s) that use them, increases the flexibility and autonomy of the application designer to apply those functions and services to the specific needs of the application. Therefore, **programmability in a lower layer can be considered as a means to postpone design choices upward in the layering**, closer to the application, and later in time, even though the resulting functions may actually take place inside the network.

Active networks provide end-to-end system designers with more choices for function placement. Successful system design still requires making correct choices which depend on the details of the particular problem being solved. Yet, the *key task* in active network design is to **identify the cases in which the performance gains and enhanced capabilities justify the cost incurred in deploying an active networking architecture**. Therefore, active networks can be considered as a further enhancement of the network capability.

> *However, while making lower layers more active or programmable is likely to enhance the applications' autonomy and flexibility to adapt to a changing environment, the potential risk accompanying this advantage is that programmable lower layers may reduce the network transparency, i.e. the predictability of the network behaviour.*

`Why is network transparency so important ?`

The end-to-end argument is similar to the RISC[158] principle, [Rad82]: building a *complex function into a network implicitly optimises the network for one particular set of applications* while substantially increasing the cost of a set of potentially valuable applications that may be unknown or unpredictable at design time. **Establishing a general programming interface can therefore lead to complex and unpredictable interactions among independently designed applications and independently acting users.**

Since lower-level network resources are shared among many different users with different applications, the complexity of potential interactions among independent users rises with the complexity of the behaviours that the users or applications can request.

For instance, when the lower layer offers a simple store-and-forward packet transport service, interactions take the form of end-to-end delay that can be modeled by relatively straightforward queuing models.

[158] Reduced Instruction Set Computing (RISC): Computer architectures which expose a minimal instruction set with fast processor implementation to optimize high-level language compilers for using the large number of registers and instruction pipelining to achieve better application efficiency.

Adding priority mechanisms (to limit the impact of congestion) that are fixed at design time adds modest complexity to models that predict the behavior of the system. However, relatively simple programming capabilities, such as allowing packets to change their priority dynamically within the network, may create behaviors that are intractable to model, in the same way that the simple rules of cellular automata such as Conway's Game of Life [BCG82] can lead to remarkably complex behavior.

End-to-end arguments and the layering principle do not oppose active networking per se. Instead, to maintain the largest degree of network transparency, they strongly require that ***the semantics of any active features be carefully constrained*** **so that interactions among different users of a shared lower level can be predicted in order to use the services and functions of that active layer.**

Saltzer, Reed and Clark note that getting the semantics of active enhancements right is a major challenge in programmable networking, since wrong active enhancements are likely to be worse than none at all. They claim that even though active networking is not governed by end-to-end arguments, there are not practical examples of a sufficiently simple, flexible, and transparent programming semantics suitable for use in lower levels of networks, [E2EC]. Nevertheless, we regard this issue is an additional pro-argument for the application of formal techniques in active network design. Engineering with logic improves the predictability which is so important for all users of a shared network, including those that do not use the programmability features.

On the other side, Partridge, Strayer, Schwartz, and Jackson from BBN claim that active networking could have a place in every network layer — except the internet layer — where end-to-end arguments discourage "activation", [E2EC].

In the following, we summarize their arguments which correctly reflect the present situation with active networking. We also deliver our own comments where appropriate.

Application Layer: easy activation (Java applets, CORBA). Active networking clearly enhances the performance for higher layers.

Transport Layer: non-trivial, but a possible case. *Application Layer Framing* (ALF), [ClTe90], argues that applications are better placed than transport protocols to determine how their data should be packetized and transmitted over the network. Current transport protocols offer a limited set of communications paradigms. Hence, the transport layer would benefit from programmability, which in turn would enhance the performance of the application layer. Nevertheless, the experience with fair sharing of network resources among users imposes the restriction that *transport protocols should be constrained* to certain behavioral norms ([Jac88], [JaRa88], [FlJa98], e.g. the transport protocols' reactions to congestion. In particular, while a programmable transport layer could enhance higher-layer performance, it could also require new functionality at higher and lower layers to ensure that erroneous or malicious transport-layer programs do not violate transmission rules.

How to impose *behavioural norms* on a program is an open question, and until it is solved, adding programmability to the transport protocol *potentially violates* the end-to-end argument.

Yet, this is exactly our pro-argument of how network design has to be done. Every single improvement at one layer has to be evaluated at all layers, since behavioural norms require a cross-layer functional consistency.

Network Layer: *most serious conflict***.** The purpose of the network layer is to achieve universal connectivity and communication between arbitrary numbers of heterogeneous devices. The difficulty is that it is hard to find a way that active networking could enhance this service, while easy to find ways that active networking harms this service.

The most important argument contra active networking in the network layer is the following. When a packet's path is affected by some code carried in the packet or, worse — in someone else's packet —, the chances that the packet will reach the destination are reduced considerably. Programs are buggy, and there is still no effective way to prove the correctness of a nontrivial program (except that everybody commits to using formal tools, which still cannot prevent malicious behavior). When the delivery of a packet depends on code execution at each node in the route, the packet is at the mercy of poorly implemented, damaged, or out-of-version execution environments.

Since the code can use any of thousands of variables as conditionals in determining a delivery path, every communications path is therefore unique. If a user's packet program does not work, only this user has the necessary knowledge to debug her mix of programs and data to figure out why. At the same time, because the range of actions a packet can take has been increased, the damage that a malicious program — say, one that copies packets at each node — could inflict would be magnified exponentially and made harder for network operators to stop.

However, this argument is more likely to address a "generation problem" than an engineering constraint. Safety is definitely a serious problem, but nodes in future autonomous self-healing networks will be able to take care about themselves and the programs they are hosting.

Another contra argument of active networking in the network layer is that it adds a great deal of complexity to the very simple process of forwarding a packet. In present day Internet, a router has only three choices when presented with a packet: to transmit the packet, to delay (queue) the packet, or to discard the packet i.e. to throw it away. The simplicity that ensures interconnectivity, the fundamental feature of the internet layer, resists the complexity that programmability brings.

Partridge et al., [E2EC] claim that all active networking can do in the network layer is to increase the customer's flexibility to choose among these options, and thus to increase the risk that the choice is the incorrect one. Nevertheless, the above argument does not represent an engineering constraint either, but rather reflects a *belief* based on the present day state-of-the-art networking. Thus, fears of too complex programmable systems in the late 1980s, which led to selecting device polling as network management policy, are overcome now.

Today, programmability of the network management system is considered as an enormous advantage, which enhances performance and eliminates needless data mining and analysis work by sending the program directly to the data it needs to observe and report about.

For instance, the CANEs architecture, [Meru99], allows the users to select from available set of functions to be computed on their data at the network nodes and to supply parameters to their computations. The functions are chosen and implemented by the network service provider. In this way, CANEs exposes the low-level network function in a controlled manner, not only for access, but also for manipulation by higher layers. Therefore, the CANEs approach to programming the network layer does not contradict the original E2E argument which suggests *who* should provide the function, not *where* it should run. In fact, CANEs dynamically extends the low-level function for better application efficiency by application-specific customization of system function, but without tightly binding any application specific function into the lower layer.

The CANEs model can be used by applications for influencing network's response to congestion control. The packet loss recovery due to congestion control often renders other related data which are useless for the application. For example, an MPEG-2 stream consists of I, P and B frames, where P and B frames can possibly require other frames in order to be properly decoded. Thus, if an I-frame has been discarded, the corresponding P and B frames should be better discarded as well. The CANEs programmable congestion control defines *reduction techniques* (e.g. partial packet discard, group of pictures level discard), and makes them available to applications such as video streaming and conferencing. An application can define data units based on application semantics, and an application *flow* can provide *advice* to the network about which reduction technique to use to discard application data units for that flow. When triggers indicate the need for congestion control, flow state is examined for advice about how to reduce the quantity of data. The network node is not required to take the advice, and may apply generic bandwidth reduction techniques.

As the results of experiments in [BCZ97C] show, this approach significantly reduces the amount of data discarded at the receiver. In other words, less network bandwidth is consumed carrying the useless traffic, which in turn, helps to reduce the congestion.

In addition, by using formal verification techniques in future network design, selecting the right choice will not be a serious burden anymore, even at much higher complexity of the packet-operating program. System verification, cooperative customer interfaces and improved network technology are the contra-arguments of the contra-argument. There is nothing wrong with a higher complexity as long as it delivers a better application performance. There is only work ahead to do.

The Subnet and Link Layers — active networking is a useful feature. Adding programmability to the subnet and link layers, e.g. downloading code to update and run the appropriate signaling protocol, has the potential to both enhance performance and eliminate some higher-layer functions.

The Physical Layer — a positive logical conclusion. In a Wandering Network we envisage even more sophisticated systems at the physical layer where e.g. an autonomous mobile hardware component such as an all-purpose PCMCIA card or a multifunctional self-assembling gallium-arsenide-on-silicon-on-polymer chip can plug in [itself] into a free slot of a [mobile] router, download the appropriate medium access control (MAC) layer protocol, such as 100 Mb/s Ethernet (either Base-T or AnyLAN) or FDDI, and proceed to transmit and receive packets.

From the above discussion on system layering and end-to-end arguments, we conclude that Active Networking has yet a well motivated right to live.

> Active Networking, — *if applied correctly* —, represents
> a useful extension of the end-to-end arguments
> at all system layers.

The "correct" implementation of Active Networks can be realized either through explicit functional restriction and control, or via formal verification of the critical network design. Exporting application-specific functionality out to the periphery has been a useful design principle for many years. The end-to-end arguments arose from work on secure operating system kernels in the Multics project ([SCS77], [Reed77]) and work on end-to-end transport protocols in LANs and the Internet experiment [CPR78]. John Cocke took a similar approach and his colleagues on the role of compilers in simplifying processor design which led to the RISC architecture [Rad82] by suggesting to move functionality from lower layers to more application-specific layers.

Security research in Active Networking has been mainly focusing on application layer networking and on the development of safe languages (SNAP, PLAN, etc.) and restricted execution environments (SANE). Until recently, little attention has been paid to the more critical question on the effect of flow manipulation on ***end-to-end security***. In his thesis, Brown, [Brown01], examines the threat model implicit in Active Networks and develops a framework of security protocols in use at various layers of the networking stack, in particular w. r. t. their utilization for multimedia transport and flow processing. After thorough examination of the various problems in providing end-to-end security in Active Networks (vulnerability to attacks on intermediaries, coercion, etc.), the author concludes that it is not reasonable to allow active routers to access to the contents of network flows without seriously degrading the functionality they provide. Therefore, Brown proposes the enhancement of watermarking with the idea of splitting trust throughout the network in order to provide end-to-end security. In sections 7.2.3, 7.2.4 and 7.3.3 we discuss how the WLI model integrates the results of this novel research.

Today, the increased diversity and complexity of the network traffic imposes the requirement for distributed, and thus, for increased, but effective processing inside the network, locally and dedicated to where and when the specific traffic problems emerge.

Therefore, the question of enhancing the effects of system layering and end-to-end arguments gains on importance. Active Networking is a first step in the right direction.

In the previous sections we have seen that even critical network layer programmability such as the CANEs approach to congestion control does not conflict with the E2E arguments, unless it loses control over the constraint of *who* is providing this extra functionality (the application node). Another example is the Exokernel operating system, [Eng95], which securely multiplexes and exports physical resources and allows the library OS to compose them for desired application efficiency. Besides, as long as the active network guarantees that only the applications that use this mechanism are required to pay its cost, while other applications can use the traditional mechanisms, everything should be alright.

Finally, as Bhattacharjee et al. argue, some services can best be supported or enhanced using information that is available only inside the network [BCZ97C], instead of applying an end-to-end control scheme, because of the very little information available at the ends of the network.

The most critical question which remains, however, is the one about the necessity of complete re-design and re-implementation of the existing protocols and applications which are already based on the IP protocol suite architecture in the case that active network architectures prove to be more effective than today's simple Internet. The cost of such an "updating" reform of the Internet to the needs of modern communications is *tremendous*. It would take many years to define a pay-off business model and coordinate the process worldwide. This is the real fear that makes ISPs and network suppliers represented in the telecom standardization oppose a change towards a new generation of radical active network architectures, — sometimes by involving such arguments in the discussion as the "end-to-end" principles, but often using them without a proof. The *cost* of the Internet was nothing, or almost nothing. (It is doubtful that DARPA will ever mention the real cost of the Internet project.) Who is going to oppose such an argument ? The Internet was simply there; ready for use, although not for the same purpose in mind it was once designed.

While taking into account this historical argument, some cooperative AN approaches do not require re-implementation of existing protocols and applications. Protocol booster architectures, also called Performance Enhancing Proxies (PEP), integrate performance-enhancing functionality that can be located at the edges of the wireless part of the network. These protocol boosters operate *transparently* without the need to modify the existing IP suite. To enable efficient operation the boosters have to be designed for specific applications. For the case of TCP applications, IP booster architectures can double TCP throughput even under noise propagation conditions. In fact, the Internet community is already evolving towards activation. For instance, IETF is working on implementing some practical Active Networking ideas such as a recent RFC proposal[159] from Cisco Systems.

Finally, to end, we summarize that the end-to-end arguments are useful engineering guidelines, which are unfortunately not always considered in practical network designs. In the previous sections, we have shown that even such self-evident Internet equipment like firewalls, proxies and some routers violate this principle. The Wireless Access Protocol (WAP) is perhaps the best commerce-driven example, which breaks the rules (and the network in two parts) without ever being able to improve neither performance, nor service usability !

Therefore, this example demonstrates how seriously careful system design and end-to-end interoperability have to be carried out to avoid ending with an immature product. We have shown that Active Networking have the potential to improve system performance by enhancing the existing design principles with more application-specific directives.

For this reason, we decided to investigate in detail the entire discussion around the end-to-end arguments at the end of this work, because they are the premise for the next step we are working to, the Wandering Network.

[159] http://www.ietf.org/internet-drafts/draft-shore-friendly-midcom-00.txt

7.1.5 CAPTURING *HORIZONTAL AND VERTICAL DYNAMICS*: THE WLI PRINCIPLES

Today, it is unclear yet what architectures and service models will define the basis for the next-generation Internet. As Matyasovszki and Flanagan state [MaFl01], the fact that the evolution of the Internet is surrounded by uncertainty is due to **the lack of cohesiveness and clear purpose in network technology design**; the history of MPLS development proves how many misconceptions can arise.

An active network differs from traditional architectures primarily in what it does *not* specify. Instead of defining how the nodes work together to provide the network service, the active network describes functional slots that must be instantiated to provide a particular network service. These slots create an idle functionality that can be invoked on demand in the network architectures. Besides, some on-demand functional blocks could be placed in such a way to unify the functionality of two or more OSI layers.

Therefore, we decided **to extend the ad-hoc rule of letting out specification in network design to an acceptable degree of autonomy, but consistency and self-determination through adaptability** as known from eco-systems and cellular automata such as the Conway's Game of Life [BCG82].

As we found in the previous sections, everything is allowed, as long as we remain conform to the fundaments of system design. Thus, extending and refining these principles is natural. Even Saltzer, Reed and Clark agreed that the **"end-to-end arguments are *one of* several important organizing principles for systems design"**, [E2EC]. While an end-to-end argument can facilitate the design-time function placement that leads to a more flexible and scalable architecture, there can be different situations where other principles or goals have greater weight.

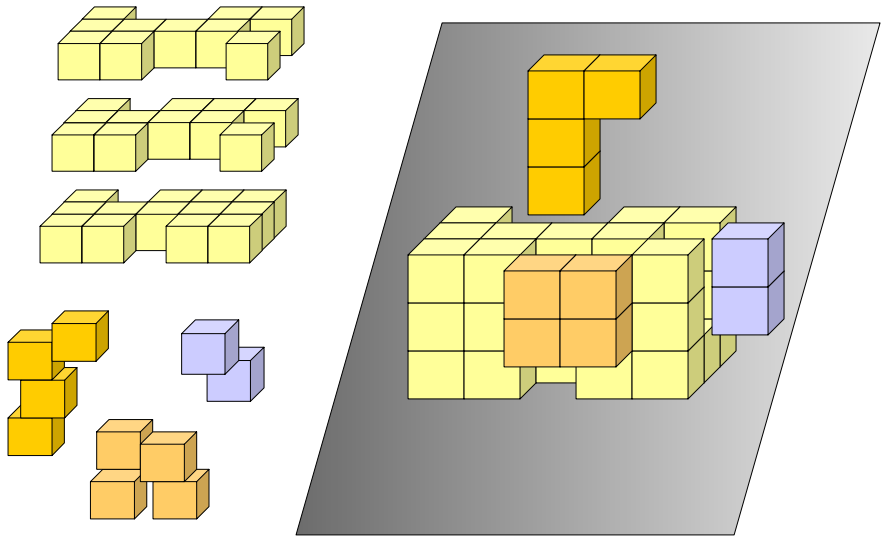
Then, what about evolutionary networking, the WLI domain ? Who can tell which services will be broadly used in 20 years in order to implement their functional interfaces in present-day low network layers ? And what can be stated about a short-term functional dynamics within a rapidly changing environment such as mobile communication?

While generalizing the conclusions of the previous section, we realized that functional slots could be placed anywhere and anytime in an active node as illustrated on the figures two and three below: in the application layer, in the network layer, and even in the physical layer (as assumed in the WLI approach). The only requirement is that each consequent arrangement of the functional blocks of the active element:

a) improves the previous one in terms of performance, capacity, scalability, etc., and
b) retains the operational consistency of the single layers and between the layers as suggested by the system layering and the end-to-end arguments principles.

This is the basic rule of the Wandering Network. In other words, if a particular implementation, such as lossless video transmission, tends to violate the network layering principles, e.g. by establishing explicit communication channels from applications to network-layer middleboxes (routers/switches, firewalls, etc.), this modification is eligible only if the system improves its own performance and the one of the entire network community.

Furthermore, statically designed networks for the worst case conditions do not make the best use of resources which may vary in time. Also, they are not capable to incorporate future technology changes, and thus be long-term adaptive to their environment. WLI reflects the network dynamics of functional split, movement and merge throughout the entire life cycle of an evolutionary network. Therefore, we regard the four principles of the Wandering Network (Section 5.3), — *Dualistic Congruence*, *Self-Reference*, *Multidimensional Feedback* and *Pulsating Metamorphosis* —, as essential enhancements of the basic principles of system layering and E2E design.

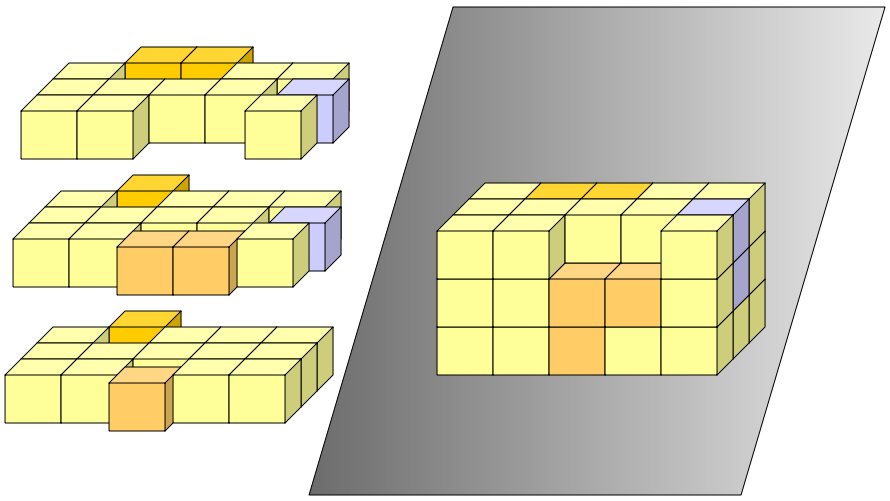


a) Block Composition b) System Composition

Figure 91: A block-oriented approach to WN design and maintenance

Figure 91 and Figure 92 illustrate the two major techniques for construction and maintenance of a Wandering Network functional node (netbot), the *block-oriented* and the *layer-oriented* approach, respectively. Whereas the left-hand side of the figures represents the composition of new functional elements (in colour) corresponding to the particular layers of the node – application, network (OS) and physics (SoC, COTS, etc.) –, the right hand side show the entire system composition.

In the first case, a block composition requires an “off-site” assembly of new functional elements which are then *moved to*, installed, tested and executed at run-time as blocks in the free slots of a Wandering Node. The latter assumes a parallel (CPU) or a replica (memory) architecture to switch between the hot-spot and the “drive-in” plugin during installation and test. In the second case, we have a piece-by-piece layer-wise *on-site* integration of new functionality at each single layer which is moved, installed and verified “in the order of appearance” with the node environment.



a) Layer Composition b) System Composition

Figure 92: A layer-oriented approach to WN design and maintenance

This model holds not only for the internal organization of wandering nodes (netbots), but also for their external organization (and virtualisation) into hierarchical clusters of network processors. Compliance of the new functionality to the programmable interfaces at each layer, in terms of single components and as blocks of them, must be guaranteed a priori to test and deployment.

At the end of this section we provide a concept summary of the *WLI* design principles:

> **The Wandering Logic Intelligence** extends the classical software-based programmable Active Network model with an *introspective* (w. r. t. a information-architecture/shuttle-netbot consistency, *Dualistic Congruence Principle*) *and self-aware* (i.e. state-driven, *Self-Reference Principle*) adaptive enhancement/reduction, exchange, movement, creation and development of functional elements/blocks from the application through the network into the physical platform of a wandering node (netbot) and vice versa using an event-based *hierarchically cascaded addressing scheme*, thus integrating the concepts of reconfigurable and autonomous mobile computing into an overall design framework for self-organizing Wandering Networks which are *transformed* (*Pulsating Metamorphosis Principle*), but not controlled or managed by some authorized network party or entity, — and therefore, *autopoiesized* ([Sim02a], [Mat00], [Luhm86], [MaVa80]) as a response to and *exclusively* by the implicit multiple feedback (*Multiple Feedback Principle*), – by means of *n-genes* transported inside active packets (shuttles) –, of the user community on using the network information content and physical resources.
>
> *A Wandering Network is an evolutionary network.*

The following section provides the practical argumentation behind the WLI concept w. r. t. state-of-the-art technology considerations and results from recent research. Particular attention is given to the hierarchical programmability of *netbots* and the "*n*-geneered" structure of *shuttles*.

Additional information about most recent developments in active networks, parallel and distributed system architectures, nano- and molecular devices, as well as cluster and grid computing can be found in the proceedings of the latest IEEE conferences: [DANCE02], [IPDPS02], [ISVLSI], and [CCGrid02].

7.2 IMPLEMENTATION GUIDELINES: EVOLVING THE WANDERING NETWORK

First, when we speak about reconfiguring, programming and wandering elements and nodes, we mean the physical layer. Today, we are relatively far away from the vision of vivid, autonomous, self-assembling "flying" chips which plug and play themselves in the free slots of communication devices to match customer demands and serve for the benefit of humankind.

However, a first step towards these dynamic physical layers has been made by software radio ([BHM01], [BIW99]) and RadioActive Networks, [BWG99]. A software radio is a wireless communication device in which the physical and data link layer functions are implemented in software. This enables (re-)programming of a single wireless device to use diverse coding, modulation and access protocols. In addition, new services and standards can be easily introduced and deployed in this way as software upgrades without the risk of upgrading the entire physical infrastructure, [ShBo99].

Software radio also provides the flexibility to adapt dynamically any aspects of the physical layer of a wireless communication system to meet the constraints imposed by rapidly changing environments, user demands or administrative regulations by providing improvements in performance as well as bandwidth and battery life utilization, [BHM01].

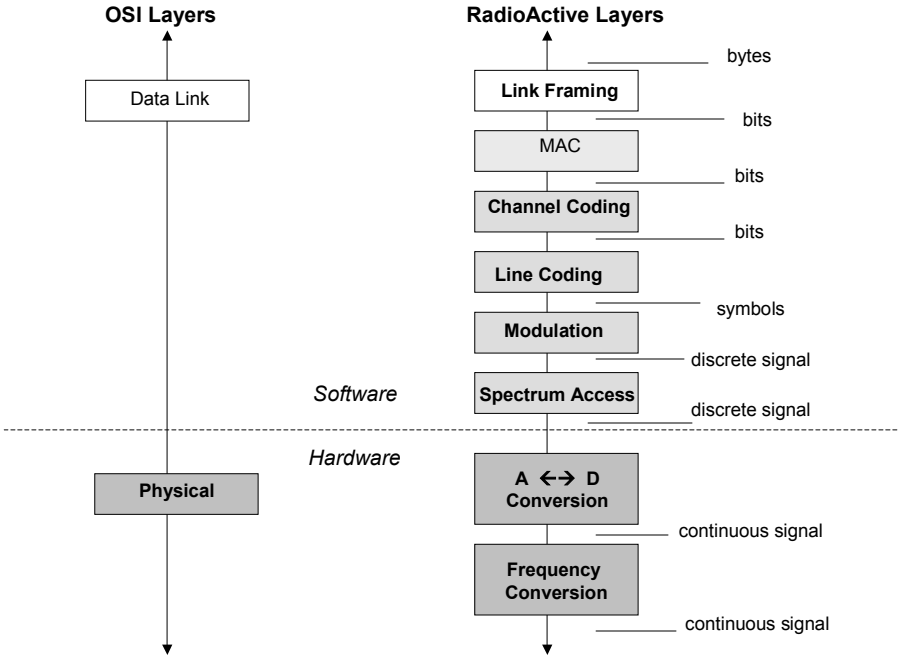


Figure 93: The *RadioActive* network layering model, [BWG99]

Speaking in terms of the network layer model and the end-to-end arguments, software radio exposes more functionality from the physical layer for flexible utilization by the upper layers, Figure 93. Intuitively, this functionality could be not only offered at system set-up, but also during the system deployment phase, thus updating, replacing and combining complete functional blocks in software.

This is already a sort of *network wandering*.

7.2.1 THE PROGRAMMABLE ROUTER REFERENCE IMPLEMENTATION

Today, most Internet routers, such as large backbones increasingly implement standard datagram processing without special features like IP options completely in hardware, – e.g. ASICs on every port which have high-bandwidth access to a local table of routes, – to raise performance and keep up with link speeds. The router CPU is only involved in processing of non-standard datagrams, basically – an Active Networking idea, and in implementing control-path functionality like routing protocols.

However, active networking is *application-specific* for a potentially large number of applications which means that the overall AN architecture should be open and programmable.

On the other hand, AN processing extends the *amount of time* spent on a single packet, which means that performance becomes a critical issue and requires the selective choice between different hardware architecture models such as pipelining and parallelization.

Therefore, to maintain the trade-off between performance and flexibility, the router hardware must be also programmable. For this reason, a variety of high-performance AN router implementations such as P4 for protocol boosters [HaSm97], AMnet for heterogeneous multicast [Metz99], ANPE [Wolf99] for diverse benchmark services (encryption, media transcoding, reliable multicast), as well as dedicated media architectures, – stream processing ([Lee99], [Bust01a]) and multicast video distribution [Kell00], – have been designed (and successfully tested) as a combination of a general-purpose CPU and FPGA circuits in the *Processing Engine* (PE), [Deca99], of the Port Processors (PP), Figure 94. In this way, the CPU takes care of the most active functions applied to a customer packet, whereas the FPGA implements functions which are particularly performance critical; both are programmable on-the-fly and integrated as *Processing Element* structures in the Active Processor Chip (APC).

To understand the role of the programmable hardware components, we start with reviewing the system organization and operation of an active router introduced in Chapter 4. The router architecture is based on a scalable cell switching fabric which connects to external links through *active* Port Processors (PP) (Figure 94, left-hand small image). The switching fabric can be implemented e.g. as a multistage network, thus supporting external bit rates up to 2.4 Gb/s and can be configured to support hundreds or thousands of PPs. The active router's Port Processors perform packet classification, active processing and fair queuing. The Control Processor (CP) provides a control and management interface to the external world and implements routing algorithms and other high-level operations.

Passive flow packets are directly passed from the input port at which they first arrive to the output port where they are to be forwarded. Active flow packets are typically queued for processing at the input port where they arrive, and then forwarded to the corresponding output port (after processing). Note, that active packet processing in pure *programmable software* Active Nodes is performed in the Execution Environments of the application layer. A programmable hardware router, however, can realize a combination of both methods: EE and PE based. Of course, active processing can be also performed at the output port, if appropriate. To provide maximum flexibility, active packets can be sent from the input port where they arrive to another port for active processing, before being finally forwarded to the required outgoing link. This allows a system-wide load balancing.

The Port Processors consist of a *Transmission Interface* (TI), a *Packet Classification and Queuing* chip (PCQ), a *Filter Memory* (FM), a *Queuing Memory* (QM) and *Active Processing Chips* (APC), Figure 94. The Transmission Interface contains the optoelectronic and transmission formatting components. The Packet Classification and Queuing chip performs classification of arriving packets to determine how they are to be processed and where they are to be sent. It also manages queues on both input and output sides of the Port Processor. Packets can be assigned to queues in a fully flexible fashion (e.g. on a per-flow or per-aggregate base). The queues can be rate controlled to provide guaranteed QoS. The PCQ has two memory interfaces, one to a Filter Memory used for packet classification, and one to a Queue Memory used to store packets waiting processing or transmission.

The active processing is realized by one or more Active Processor Chips, each containing several on-chip processors with on-board memory. Each APC has also an external memory interface providing access to additional memory which is shared by the processors on the chip. The APC processors retrieve active packets from the QM, process them and write them back out to the corresponding outgoing queue. The processing elements are arranged in a daisy-chain to avoid multiple APC interfaces to the PCQ (Figure 101 for details). The active processing capacity can be scaled by incorporating fewer or more APCs at each port.

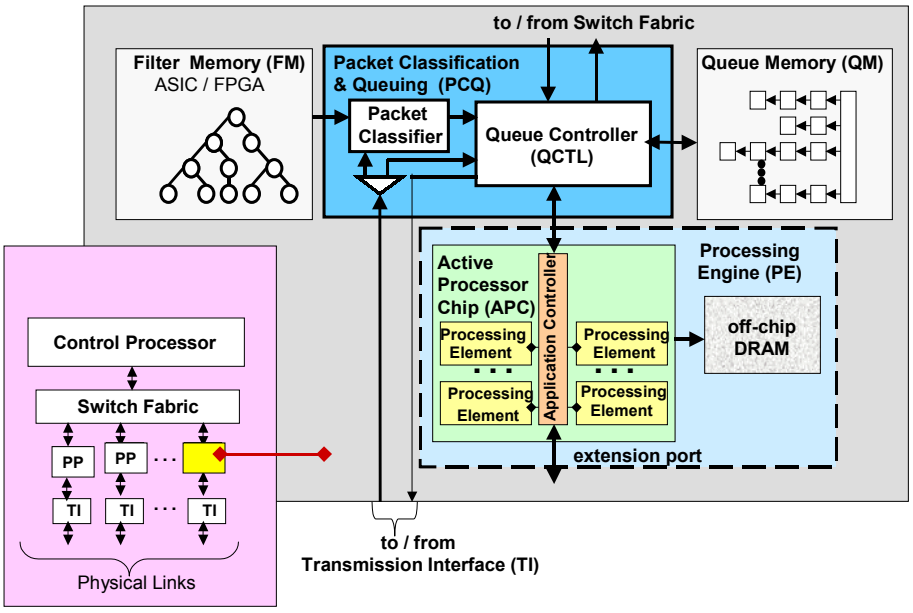


Figure 94: A programmable router/switch and its port processor (PP) architecture

Systems with small amount of active processing can sacrifice APCs at most ports. Instead, active packets can be forwarded to other ports with APCs. However, once the design decision is made, the process is irreversible (at least at run-time) and the router may experience congestion upon a strong continual traffic (whatever the load-balancing scheme might be). At this point, the WLI concept of automatically adaptive processing capacity at run-time by means of a mobile self-organizing hardware regulated via state-feedback between transport infrastructure and processor "farms" could provide the required reversible solution in future.

The Port Processors are operating as follows. When packets are received from the Transmission Interface (TI), the headers are passed to the Packet Classifier (PC) which evaluates the flow and assigns a tag to the packet. To provide the required flexibility, a fast general flow classification algorithm is required, such as the one in [SSV99].

At the same time, the whole packet is passed to the Queue Controller (QCTL) which segments the packets into cells and adds it to the corresponding queue.

Depending on the QoS requirements, either per-flow or aggregate queues are allowed. The filter database (FM) decides whether flows are aggregated or handled separately.

In the following, we discuss in some detail a possible realization scenario of a Wandering Network based on the above architecture. Our intention is to illustrate how WLI extends the idea of exchangeable software and hardware functional blocks towards a flexible, autonomous, self-organizing system and how programming (or adaptability) can be performed "down to the gate" level.

7.2.2 Reference Software and Hardware Execution Environments

We start from a hybrid high-performance communication platform integrating today's IP and ATM worlds, [PST95], which has been extended and optimised to accommodate the Active Networking idea of the DAN project described in Chapter 3, [Deca98], through the *Active Network Node* (ANN) architecture shown on Figure 95, [Deca99]. This platform is also a reference point for the critical overview in section 7.3.1.

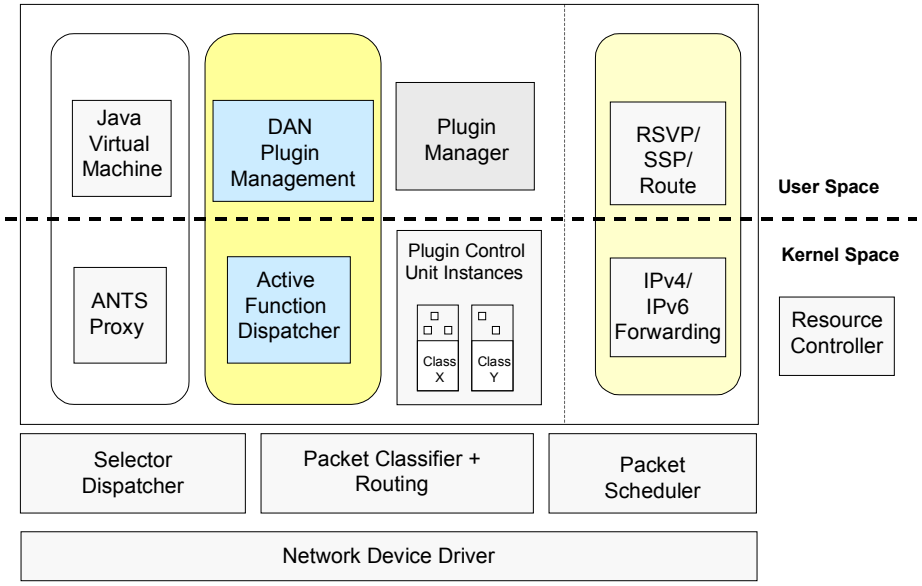


Figure 95: An Active Network Node (ANN) software architecture

We selected ANN/DAN as starting point of our discussion for the following reasons:

1. It is a well-designed and tested AN architecture of ***both a programmable software and high-performance programmable hardware router/switch*** which is representative for the advantages of Active Networks against today's passive networks.

2. It is a technology-aware solution which ***integrates both ATM and IP worlds*** in an evolutionary way (which is also characteristic for the WLI approach) a) through programmable COTS interface hardware, and b) in a typical AN software fashion by virtualizing the IP routing procedure in a special execution environment which implements ***Router Plugins,*** [Deca98b], or active extensions software**.**
3. It maintains ***multiple execution environments*** (ANTS, DAN and IP) which have been demonstrated to interoperate very well for the target domain of high-bandwidth and low-latency applications.

4. It provides the ***reference point*** for the EE based implementation of ***network overlays***; The IP protocol stack is viewed as (yet) another Execution Environment, i.e. a virtual machine, with the special property that the other EEs can not work without IP since they use it for routing and forwarding.

5. It provides the ***reference point*** for the platform based implementation of ***network "underlays"*** as ***Dynamic Hardware Plugins*** (DHP), [TTL01] to realise diverse processing algorithms, protocols and variations of them which we regard as the archetypes of WLI netbots, the building blocks of self-organizing and mutually exchangeable software and hardware components.

The rest of this section is dedicated to a functional review of this architecture and its enhancement with w. r. t. a possible WLI implementation scenario.

The hardware architecture of the DAN's *Active Network Node* consists of a set of *Active Network Processing Elements* (ANPE) connected to a scalable ATM switch fabric, [Chan97], Figure 96. The ANPE comprises a general-purpose processor, a large FPGA and memory; it is connected to the backplane via the *ATM Port Interconnect Controller* (APIC) chip, [Zub95].

Figure 96 illustrates also an example for load balancing which is going to be discussed later in more detail. A data flow comes into the ANN at ANPE *A* and goes out at ANPE *D*. The active processing is done in ANPE *C* since ANPE *A* is heavily loaded and the load-sharing algorithm of the Control Processor (Figure 94) directed the flow to ANPE *C* which finally directs the flow to the ANN connected to ANPE *D.* Thus, ANPE *A* and ANPE *D* switch the flow entirely in hardware without CPU intervention through the APIC. The Control Processor (CP) provides an external control interface and manages the Port Processors (PP); it is responsible for maintaining flow classification data structures and filters, as well as binding flows to applications at each Port Processor via flow identifiers. In larger systems, the CP may be a shared memory multiprocessor (or a network processor, [WoFr02]) matching the needs of the specific configuration.

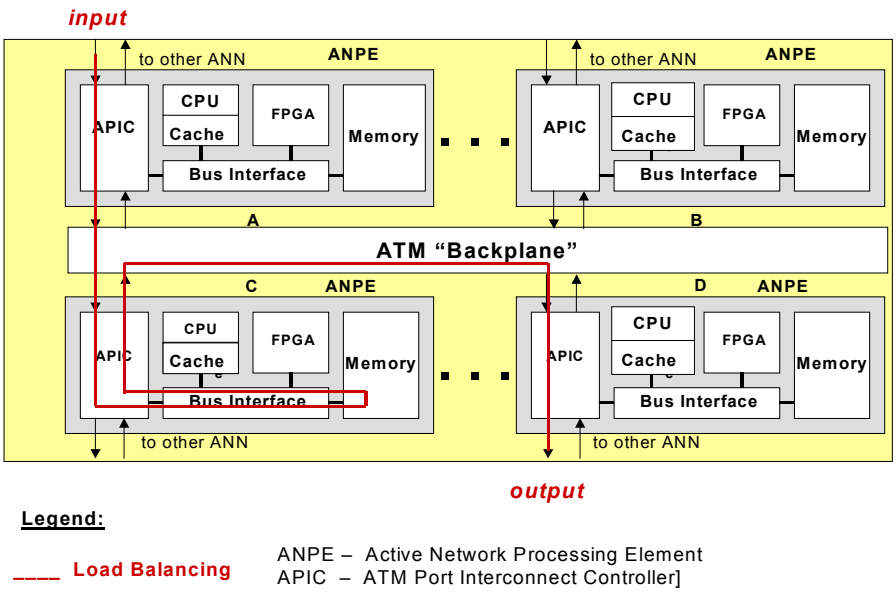


Figure 96: An Active Network Node (ANN) hardware architecture

Within this context, *active plugins* (or Active Applications, AAs), – code blocks implementing application-specific network functions, [Deca99a], – are downloaded and installed on the Active Network Node upon a reference in a datagram, through a special configuration packet, or by an administrator intervention, Figure 97.

Active plugins can create *instances*, – flow-specific configurations of active plugins, where the individual properties of instances are EE and plugin specific. For example, an IP instance consists of the code that forwards the packet and the required information about the interface on which the packet has to be forwarded. However, all instances use the same well-defined API that embeds them into the system.

In Chapter 6 we demonstrated how reachability trees in active ad-hoc networks can be automatically updated through ***incorporating structural changes into r-shuttles*** to enable a vivid routing procedure with the WARAAN algorithm. The ANN/DAN approach uses a similar method for ***forward propagation of state information*** with the packets of a flow.

Here, the EE of an upstream node can request a *selector* from the NodeOS to *label* a chain of AA instances, initiated after plugin download by a local executing environment, [Deca99], [KRWP01] that has been created upon arrival of the first few packets of a packet flow. The selector is included then within a special SAPF packet, (*Simple Active Packet Format*, [TD98]) which is inserted into the subsequent packets of the same flow to enable the downstream node to efficiently lookup the state information using hashing and thus directly assign the flow state information to the packet, Figure 97.

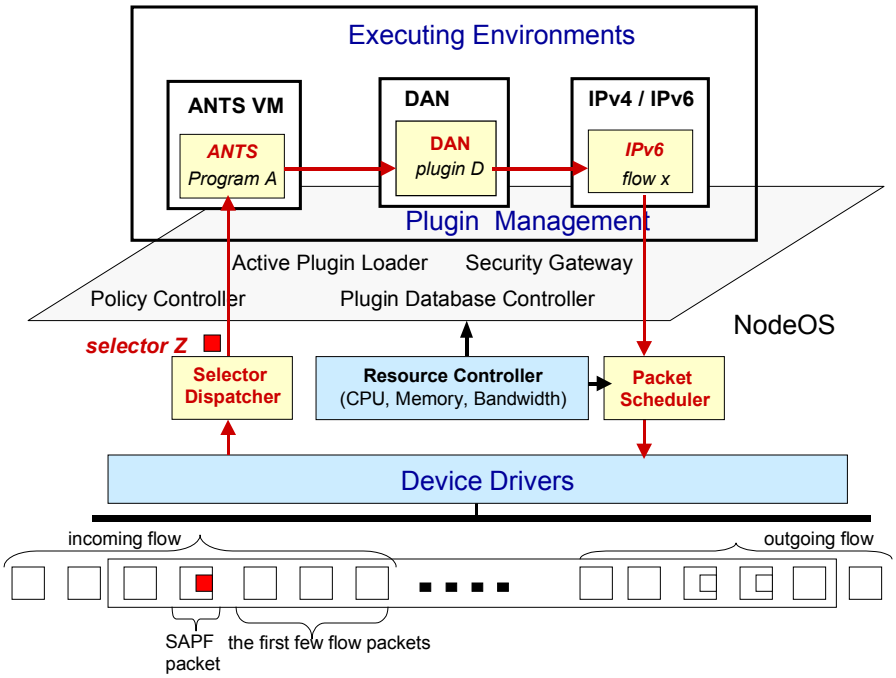


Figure 97: State information lookup through a selector tag labelling pipelined AA instances

The flow lookup based on selectors is implemented by the Selector Dispatcher. Some detail of the selector lookup procedure is explained on Figure 98. While plain IP packets are forwarded according the standard routing procedure (classification, header processing, output queuing), active packets move through configured kernel plugins with the active function dispatcher passing the packets to instances of plugin objects (AAs), instantiates objects or triggers download of plugin classes as needed. In addition, the SAPF packets which propagate the selector label for the pipeline of instantiated active plugins corresponding to the current packet flow are undertaken a streamlined processing using the pre-established state.

Thus, the processing of an active packet in the beginning of the flow determines the path of execution of the subsequent packets in that flow. (The overall procedure of event-based charging and execution of software extensions/plugins upon packet arrival is illustrated on Figure 99. The interested reader is kindly asked to refer to [Deca99] for further details on it.)

This is an essential advantage when compared to the processing overhead of a per-packet initialisation, chaining and execution of AA/plugin instances. However, this characteristic does not appear to be sufficient enough to leverage the performance of an Active Network Node. Therefore, researchers direct their efforts towards optimising also the hardware part of programmable router/switch architectures, e.g. [LMA98], [Hess99], [HMPZ99], [LHAM99].

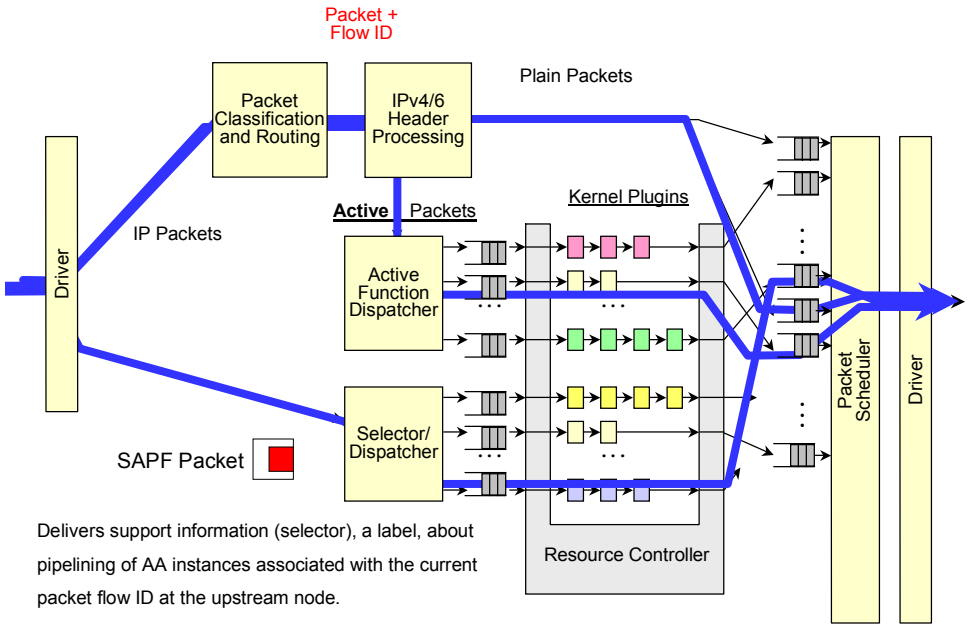


Figure 98: Main information flows through the processing engine kernel

That hardware often replaces software for the reason of performance even in non-traditional areas such as routing [PeZu92] is a well known fact. However, the step from an FPGA switching board (Chapter 3) to on-the-fly programmable routers ([HaSm97], [HaSm98]), and then to high-performance multi-port *extensible* routers, [Kuhns02], is a very small one, but difficult to implement. Nevertheless, apart from implementation details, a *programmable* router is a network device for providing application deployment mechanisms.

In general, there are two basic schemes for delivering network applications to a programmable router today:

1. *passive applications* deployed at session setup via signalling protocols, and

2. *active applications/extensions/plugins* requested by incoming packets or carried by the packet for execution on the programmable router.

Both of them are usually realized in software. However, optimal router architectures must be able to utilize the flexibility available in software and the performance offered by reconfigurable hardware.

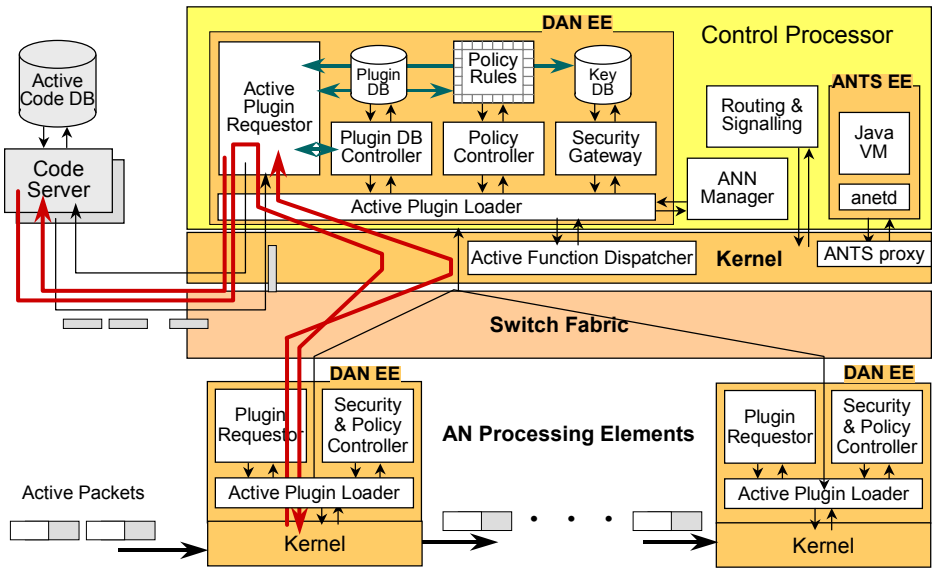


Figure 99: The DAN network level software architecture

Therefore, in the following we discuss in some detail the practical *Dynamic Hardware Plugin* (DHP) idea proposed by Taylor and his colleagues from the Washington University in St. Luis, [TTL01], which provides the basis of high-performance multi-port *extensible* router architecture, [Kuhns02], and hence appears to be the next step on the way towards future autonomous *Wandering Logic Intelligence* architectures. Note, that here we address only WLI related characteristics. Additional information about design issues for high-performance active routers can be found in [Wolf99], [WoTu01] and [WoFr02].

The DHP architecture is based on reconfigurable hardware for Active Processor Chips (APC), Figure 94, which provides a flexible and scalable mechanism for implementing high-performance programmable multi-port routers by *enabling multiple networking applications to be dynamically loaded and run into a single hardware device*, thus providing a flexible functionality and distribution of the flow processing in terms of hardware. DHP can support a broad spectrum of computationally intensive applications such as encryption and streaming data services (e.g. video conferencing) which can benefit from parallelization and pipelining with dedicated on-chip logic and memory resources for each plugin as well as arbitrated access to two types of off-chip memory.

The starting point for discussing the Dynamic Hardware Plugins idea is the router architecture presented in [WoTu99] which already provides *a scalable software processing environment* using elements with multiple RISC cores (Active Processor Chip on Figure 94) on a single device and is suitable to hardware processing integration, Figure 100. The Hardware Processing Element in Figure 101 implements the Dynamic Hardware Plugins (DHP) architecture on a single FPGA or a hybrid ASIC/FPGA chip to add flexible, parallel hardware processing capability to the Port Processor.

The idea of Taylor et al., [TTL01], is to use the hardware processing elements as an extension (or replacement) of software processing elements in a dynamic fashion, i.e. during run-time.

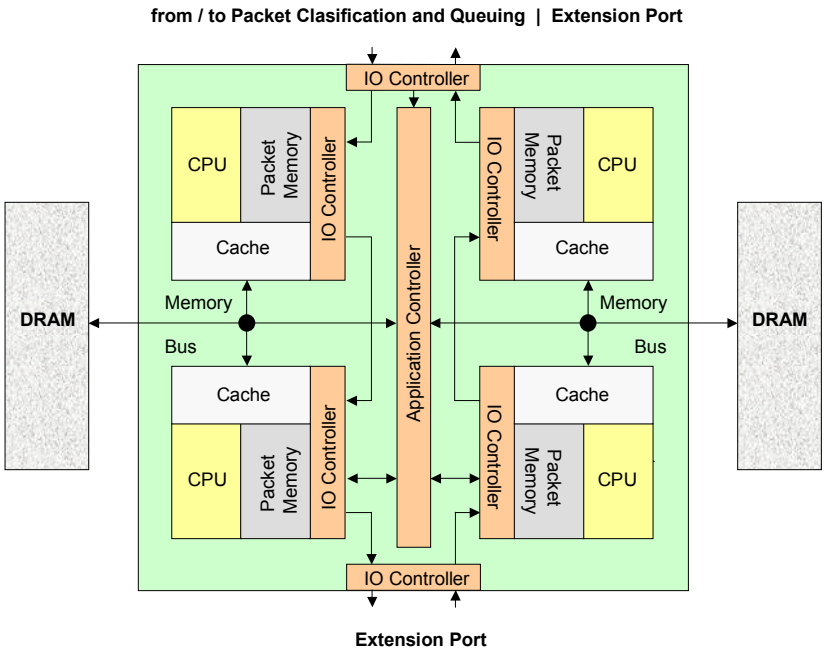


Figure 100: A software-processing element (SPE) of the processing engine

To provide a scalable solution, Taylor and team propose a hot-plugin architecture where HPEs are arranged in a ring via a standard interface over *Input Output Controller* (IOC). Daisy-chain rings can be operated at higher clock frequencies than busses due to their simple point-to-point connections. Also, single plugin can make use of the full ring bandwidth if necessary. Since the bandwidth required between the QCTL and the entire set of HPEs (APC chips, Figure 94) can be bounded by the link bandwidth (assuming that each active packet passes once from the QCTL chip to a HPE/APC and is returned once), this fact does not create a bandwidth bottleneck.

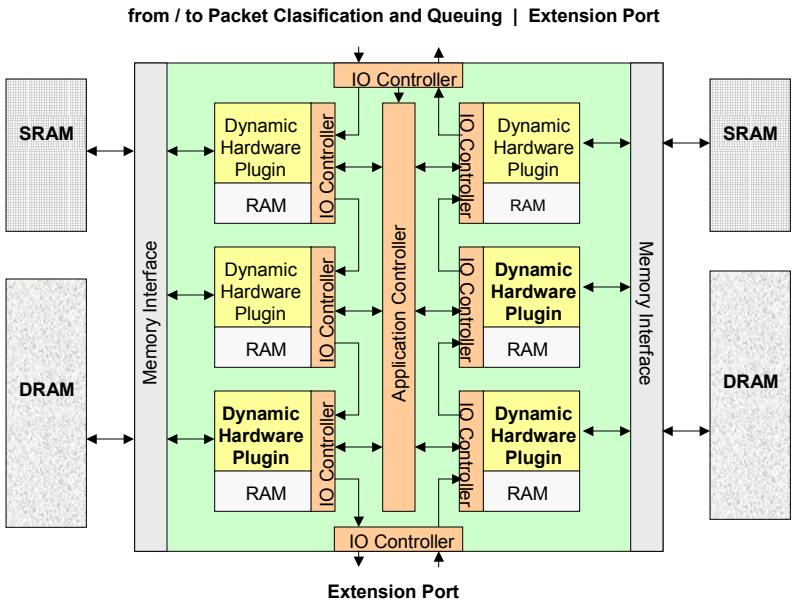


Figure 101: A hardware-processing element (HPE) of the processing engine

While an IOC is provided for each hardware plugin, two IOCs interface to upstream and downstream HPEs. The upstream IOC may interface to another processing element or directly to the PCQ, Figure 94. The downstream IOC interfaces only to other HPEs. The upstream IOC contains an additional port to the Application Controller. When new applications are to be loaded into the hardware plugins, the upstream IOC must pass control messages and application data to the Application Controller. While a hardware plugin undergoes reconfiguration, the associated IOC passes data to the next IOC in the ring. This mechanism allows ***applications to be dynamically loaded into hardware plugins*** without interrupting the flow of data through the processing ring.

Finally, like a software API, hardware plugins must have access to a *static* set of ports for data I/O, control, and external memory through a standardized hardware plugin interface. The latter includes off-chip SRAM and DRAM interfaces, IOC interface, and Application Controller interface. Each application may also define its own interface to on-chip RAM.

In this way, a single SPE of the Processing Engine can be extended by one or more HPEs to raise the performance of the router's ports. Each HPE supports a broad spectrum of applications through dedicated on-chip logic and memory resources, as well as access to two types of arbitrated off-chip memory resources. DHPs provide applications with the reconfigurable logic and memory resources to process data flows. In this context, hardware plugins are the physical hardware structures that may be configured to implement various networking applications.

The reconfigurable logic resources include logic gates, lookup tables, flip-flops, multiplexers, demultiplexers, and signal routing matrices. On-chip Random Access Memory (RAM) may be configured to implement queues and multi-port memories.

The potential advantages of the DHP architecture in gaining performance and flexibility for a diversity of network applications has been demonstrated in a series of implementation case studies of the Advanced Encryption Standard (AES) in terms of software, FPGAs and ASICs.

The question of when it will become possible to automatically physically exchange software and hardware processing elements on “as needed” basis, as the WLI model envisions, is almost intuitive. Recent research is already going in this direction. The mere fact that hardware is going to be exchanged and activated upon network feedback during deployment, even when being a priory mounted “on-board”, testifies that the next step of network evolution will be the one of self-organizing autonomous mobile components as it is pretty usual with satellite and spacecraft equipment today. The complete answer is only a question of time: in particular, if we take into consideration the pace of research in nano-technology.

In October 2001 physicist Hendrik Schon and chemists Zhenan Bao and Hong Meng from Bell Labs unveiled *a transistor with a single-molecule channel length*. But that device could only be fabricated as a matrix of a few thousand molecules that worked in tandem. A month later, the same team has succeeded in fabricating molecular-scale transistors that can be *individually controlled*, [BLA01]. Autopoiesis is also about control, the self-control induced by multiple feedback of the environment, [Bate72]. The new breed of Wandering Networks is not far away.

7.2.3 PACKET ORGANIZATION

The question of whether the shuttle model with its internal complex organization of differently encoded genes is a suitable abstraction for the practical realization of WLI architectures is also critical.

In fact, we cannot find any reasonable arguments against the idea of extending a packet’s contents with additional information (binary or not) about the state of the network. The only contra arguments could arise for implementation considerations related to the current state of technology. The usage of the Active IP Option field at the very beginning of AN research discussed in Chapter 3 confirms this assumption.

This basic research work that has begun long ago with the treatment of diverse *isolated* research problems (IP multicast, QoS, routing, etc.) to monitor the network and deliver the data for its long-term planning, is currently developing towards investigating diverse ad-hoc solutions to understand the subject of short-term network behaviour accompanied by *integration of properties* (because of increased multimedia traffic), along with pattern recognition and prediction of user behaviour. For instance, a multimedia multicast stream is characterized by a number of QoS properties which can be conditionally set “downwards” on a per-user-preference” base and “upwards” on a per-user-membership-base. The user membership status can be defined in turn by other users and/or by the network behaviour.

There are different ways to express this hierarchical packet-user relationship: as options or tags, as strings, tables, trees, graphs, attributes or even grammars. But, without regard to which approach has been selected, at some point the expressional limits of a practical frame such as the IP packet format is reached and other means are required.

The more complex the communication structure becomes, the more inclined is the research community to favour elaborate descriptive means such as specialized protocols and language approaches. Active Networks supporting language based secure EEs such as PLAN (Packet Language for Active Networks), [Hicks98], and SNAP (Safe and Nimble Active Packets), [Hicks01a], are not the exception in this statement.

However, what holds for the plain IP world should hold for the active IP world as well. The EEs and the packet structure should be kept as simple as possible. Unfortunately, this is not always the case. Whereas PLAN is a rich and highly flexible general purpose active packets language, it is too complex to target such problems as ad-hoc routing in mobile networks. Even a compilation into SNAP [Hicks01b], which offers significant resource usage safety and achieves much higher performance compared to PLAN, (of course, at the cost of flexibility and usability), may be inappropriate base to solve certain performance related problems.

Therefore, we claim that packet research is not completed to favour application level processing only. As shown in the previous section, in near future some traditional EE-centric operations on active packets can be shifted and dynamically distributed among the PEs at the switch ports, and namely - *at runtime,* and ***according to WLI - even upon "download/call" of autonomous mobile hardware extensions/plugins***.

This potential *functional shift,* (or *Vertical Function Wandering:* the *Pulsating Metamorphosis Principle*, [Sim02a]), between higher and lower processing layers which can be also intentionally invoked as a part of some (self-)regulation mechanism (e.g. load balancing) and provides an additional degree of freedom to adapt to the changing conditions of the networking environment.

Therefore, **WLI regards the "semantics" of the functional wandering process at a second abstraction level within the structure of a capsule/shuttle as *genes*,** encrypted data structures or "code of the code" which are interpreted or executed at special conditions. Genes are shuttle attributes that have the advantage of providing *dedicated and structured conditional information* in addition to transporting code and data (as usual in active packets) which can change the "purpose" and the "orientation" of a shuttle content. Thus shuttles are able to communicate in a direct efficient way not only WHAT and WHERE, but also ***HOW* their content should be used**.

A typical example of telling *how* a packet should be processed are error detection and protection schemes such as forward and cyclic error correction (FEC, CRC) as well as *watermarking* techniques. Since the compressed video stream is sensitive to error, video transmission over error-prone wireless channels requires both: i) error protection by channel coding and ii) error resilience by source coding. Digital watermarking techniques have shown potential not only in data hiding ([HaGi97], [LLB97]), but can also be used to improve error resiliency as shown in [CHL01], [Chen02].

This essential difference between active packets and shuttles allows to easily integrate (if required) different traffic and workload sensitive state information vectors within a particular shuttle flow which are related to that flow only (e.g. congestion control, resource allocation, routing policy, etc.), by reducing the messaging overhead and latency time for reacting to dynamic changes in the communication environment through more differentiated utilization of the peripheral programmable/exchangeable router/switch circuitry when compared to EE-centric solutions.

In Chapter 6 we demonstrated how a small shift from a more complex descriptive technique such as routing tables into a less complex one - reachability trees, which are encoded and transmitted inside packets, can bring substantial performance benefits to ad-hoc networks. Note, these does not need to be special r-shuttles carrying genes or even capsules in the AN sense; the linear tree encoding can use the IP option field.

Shuttle architectures are shifting the capsule model one step ahead towards active agency. They are an important complementary part of the code mobility infrastructure and deserve special attention.

The WLI approach is oriented towards the integration of different active packet system formats into a two-layer flexible, configurable and efficient transport unit format: i.) *application* (WHAT / WHERE) and ii.) *network* (HOW / WHERE) with the goal to reduce unnecessary processing complexity at the application level (EE-centric) in an optimised way. The design of these components should be derived on a per-function/per-application base form exact specifications and measurements within the target domain.

The efficiency of shuttle processing should be proportional to the conformance degree of the shuttle-netbot architecture to the Dualistic Congruence principle.

Since there is no practical implementation of the WLI model for the time being, there are still no benchmark mechanisms to measure this value. Yet, in the rough approximation of static, i.e. unchangeable architectures we can regard the processing of genetic information inside a shuttle during node traversal as equivalent to encryption/ decryption procedures. Then, we can refer to some elaborate benchmark measurements with security background in Active Networks.

According to a study carried out by Scott Alexander et al. on the SANE architecture, [ANSPrice], the basic operations required for authenticating packets require a *33% overhead* relative to unauthenticated packets. We assume the same maximum overhead with encoding/decoding the second abstraction level in shuttles associated with a gene. With this value as reference, we obtain theoretically already ***66% overhead at each side*** (for the worst case of genes contained inside code, with the assumption of a linear no-loop execution) which represents a substantial difference against plain packets.

Despite this fact, the time required for the additional processing of encrypted genes when measured in CPU cycles appears to be diminishing when compared to the roundtrip latency of a peer-to-peer request-response. Besides, processing overhead can be easily overcome by a higher processor rate and faster algorithms both in software and hardware. The fact that genes may carry important time-critical system information should be also taken into consideration.

Finally, it is interesting to note the effect of the processing overhead on providing security, or "*n*-geneering" in the general encryption scheme in the shuttle architecture in WLI. In their study, [ANSPrice], Alexander et al. identified two architectural paths to high performance:

1. via *sufficient restriction (or richer semantics) of actions* of each active packet, so that no authentication is required - the "RISC" approach in EEs, e.g. the PLAN system.

2. via *active extensions* (active applications or software plugins) as they pay the cost of authentication only once for a flow of packets even if this can be a large number of packets such as a customer's attendance in a ¾ hour video-conferencing session.

In the second case, a "soft-state" active caching scheme such as the one used in the ANTS environment appears to be sufficient for ad-hoc initiation and stable flow transmission over fixed networks. However, a wireless mobile environment provides various temporally unstable sources of disturbance and interference which have to be recognized individually even on a per-packet base, so that the above active cache-control scheme cannot be used with the same success. Therefore, *a second level of virtualization* within the packets themselves in terms of specialized *genes* (for the sources of packet transformation) as proposed in WLI realizes a more target-oriented dynamic traffic registration and response scheme which cannot be only handled by simple changing. Further in their report, the authors talk about a promising research area to leverage performance in secure, and by analogy – WLI, environments. This is the reduction of the packet size through very high-level languages "or some other compression scheme" in order to allow a wider range of active packets/extensions. This is exactly what WLI achieves by means of genes within the shuttles.

> **Conclusion 1:** In WLI, both performance and encryption are achieved by *distribution* and *diversification* of communication sources and network re-sources.

Nevertheless, a particular implementation of the WLI architecture and exact measurements in a test environment are required to verify the above hypothesis. With the same motivation in mind, the US National Institute of Standards and Technology (NIST) has recently initiated the *Active Networks Measurement* project[160] which intends to devise and validate a means to express the CPU time requirements of a mobile-code application in a form that can be meaningfully interpreted among heterogeneous nodes in a network.

However, even if we obtain some reasonable results about measuring performance and other benchmarks in a particular implementation, we cannot be ever certain of where to define functionality, when seen from a long-term development. Therefore, we decided to rely on wandering and its automation.

[160] http://w3.antd.nist.gov/active_nets.shtml

7.2.4 THE WLI ADDRESSING CONCEPT

At this point, we will direct our attention to a key question in the overall concept of the Wandering Network.

`How hardware can be actively managed down to the gate level ?`

In other words, how can we make sure that we can program every single component ?

We are going to illustrate now the WLI addressing concept with an example which is based on a particular solution in the DAN approach. Following the line of thought in [Deca99], we can make an important distinction between the first few packets and subsequent packets of a flow. In ANN/DAN, the reception of the first packets of a flow usually causes the plugins to create an instance for the new flow. If the packet is passed to multiple instances, these instances are chained together in a "soft" pipeline which operates like a programmable hardware vector processor on the passing-through packets.

The interesting discovery that Decasper and his colleagues make with this scheme is the following. While the plugin instance creation per flow introduces a certain processing overhead with the first few packets, the pay-off of this procedure were "dramatic", so the authors, for subsequent packets. In other words, *no demultiplexing* (as in the case with the selector packet) and *no routing lookup* are required ! Note that all flow specific information is that of a "soft-state" and can be automatically removed when no packets of a given flow are received within a given period. The operations of instances are reduced to *only those which vary from packet to packet of the same flow*.

> Now, the **breaking point** in our Gedanken-Experiment with the WLI's "n-geneering" idea, – the introduction of a second virtualisation level within active packets as *genes*, – is the very last sentence. Note, that this is exactly **the same Δ-scheme that media encoding formats** such as MPEG and MJPEG **use to encrypt motion** within subsequent frames of "static" images before being split into packets! Also, another, even more interesting point is that some recent effective compression techniques, and in particular – error-resilient algorithms[161] for video transmission over unreliable connections (e.g. a wireless link) are based on including more *context-sensitive* (in terms of both image context and transmission environment context) information, – on a server-client or hop-by-hop (transcoder proxy) base, – into the image format in order to reproduce it correctly at the destination site.

The only differences between the first and the second approach, – Active Networking and network-centric video transcoding –, are the ones of application domain and operational semantics of the supplied information – communication content (data and more) or communication environment (signaling) as packetized snapshots ("soft states").

[161] e.g. generic ([CCH98], [YWL02]) or specialized methods such as use of real-time labeling [LLB97], scalable compression and TCP-friendly control ([TaZa98], [TaZa99]) or wireless DLL control [MeiFa99], as well as multiple states in streams, [Apo00], and their optimal distribution of intra/inter macro blocks ([ZRR01], [ZT01]).

Note: The Processing Engine at the port level performs the same function, yet less programmable, as the Execution Environment at the application level of an Active Node: it has the task to operate on active packets.

In a packet model, both kinds of information can be transmitted in different flows or multiplexed in the same flow as designated (marked) packets as in the case of the selector discussed above. In both cases, the incoming packets are split and delegated for evaluation to different execution environments or active applications/extensions/plugins inside the Active Node. This may become a fairly complex and time-consuming procedure, especially when a CPU-controlled long virtual pipeline of instantiated software plugins emerge to support the processing of a particular flow at the application layer. Besides, the already processed "marked" packets have to be scheduled again in the outgoing flow which requires additional resources. Therefore, the issue of operating active packets at the application layer creates some serious considerations about the resulting performance of AN implementations.

The download and instantiation of plugins triggered by passing-through packets to organize Active Applications on demand is only one sort of feedback mechanism which leads to a self-regulated traffic – if all bits were equal. (In general, there can be many kinds of feedback schemes working in parallel and influencing each other to conduct and process information about the state of the network, e.g. [JaRa88].) But this is not the case for multimedia in general. So, what does happen with the information content during transmission ? Usually, some packets get lost in a connectionless network. This can be critical in some cases when they contain information about interpreting subsequent and/or preceding packets.

Multimedia packets have *a second level of context-oriented "structural" semantics* (e.g. association with I, B or P frames in MPEG) which has to be treated selectively. How do compression methods and Active Networks realize this ? Perhaps the best recent example in compression techniques that illustrate our idea of integrating instant network behaviour into state "genes" is the *multiple description coding*, [Goy01]. Active Networks, on their part, try to isolate functional layers where a distinct technique can be implemented and justified. For instance, Tschudin et al. ([TLG00], [WDT00], [Tschu00]) propose a *simple forwarding layer* in terms of both implementation and performance to enable delay-sensitive audio data streaming over ad-hoc networks. The method is based on a controlling pattern: active packets permanently monitor the connectivity to setup and modify the routing state.

Conclusion 2: **N-**genes realize a secondary level of virtualization inside the shuttles of a Wandering Network, thus allowing a compact structured representation and multiple descriptions coding of the transported information content.

Do we have any other option to reduce the processing overhead ?

Yes, through packet size reduction, differentiation and distribution by means of layered encoding/encryption inside the packets. What we propose with WLI is a much broader perspective of providing multiple context-sensitive feedbacks for automatic traffic regulation through introducing a second degree of virtualisation at the packet level which is also capable of shaping the network topology itself based on the intrinsic dynamic properties of the involved entities (netbots, shuttles and their interrelationships).

Genes allow us also to integrate multiple feedback mechanisms (states) into individual active packets. Then, the *n*-geneered active packets, shuttles, can be recognized, distributed and directly processed with all their context information at the ports of the Active Node, instead of being delegated for processing to some EE/AA at the application layer. Note, that we do not deny, but only extend the virtualisation model of an Active Network at the packet level which is expected to leverage the performance in combination with EE/AA-oriented processing.

> **Conclusion 3:** Forward and backward propagation of state information and complex structural changes can be realized through specialized *genes* inside the shuttles.

The above idea, when combined with the concept of automatic flow management[162] becomes very powerful; because it allows integrating transmitter and network behavior information into the packet itself on its path to the receiver, which can be instantly used, distribute resources at arrival time, i.e. without a priori reservation procedure. A redistribution of resources, and hence flow redirection (either internal or external), is undertaken only if new information arrives which exceeds the internal resource balancing scheme of the netbot.

Thus, *n*-genes can be regarded as implicit packet/shuttle representational and/or operational mechanisms which:

1. *encode structural information at another abstraction level inside a packet* (shuttle) while hiding details from inadequate processing environments.

2. *enable distributed processing* of this information in dedicated, yet programmable and exchangeable elements of the active node (netbot) *through flexible hierarchical event-based addressing*.

The first point declares that EEs do not need to be available on the node by the time the packet arrives or even later when the communication is finished; i.e. we don't need to make all possible active nodes in a network know any content representation (formats) and transmission (protocols) technique on the active packets that flow through them, unless an expected 'state of the node' is matched by the packet carrying the corresponding *n*-gene.

In the second point, the implicit addressing scheme in WLI introduces a requirement for conformance with the *Self-Reference* and *Dualistic Congruence Principles* where this information is used for verification purposes. It declares a *relative addressing scheme* which can be spread out from network grid architectures to the gate level of port processors. By "relative" we mean that through genes containing themselves data and code (as active packets do), nodes/netbots are able to also change the next-hop address within them, i.e. to select the *type* of a particular component which determines *HOW* to process the shuttle. Note: if the required element cannot be found, or downloaded, it can be created.

[162] e.g., the subsequent packets do not require redirection to another resource once the flow has been identified by the selector, [Deca99].

This option not only allows us to minimize the amount of processing on a single node and distribute it inside the network, [Deca99], but also to spread it out within the node itself, e.g. in the port processors instead of the CPU only, and even to decide about structural changes (e.g. software and hardware download) in the node's architecture when a particular threshold value of "usage" is reached.

> **Conclusion 4:** Reports about local netbot structural changes can be further propagated throughout the network by being encoded within the genes of the shuttles traversing these netbots, thus causing desired (reversible) *mutations* in the entire network topology.

Therefore, the WLI addressing scheme has an instantaneous character; it cannot only be defined explicitly through physical addresses, but also implicitly through the actual state of the node (netbot) and the ones of the arriving shuttles, which is encoded in their genes. Therefore, we call this addressing scheme "autopoietic" and not a controlled one.

A matching procedure of both "states" decides about redirection to other nodes or which resources are going to be used on the same node. In this verification procedure we refer again to the requirement for conformance with the WLI's *Self-Reference* and *Dualistic Congruence Principles*. We regard the matching procedure itself as a kind of dynamic NAT (Network Address Translation) function which can also decide about resources at different hierarchies inside the node.

> **Conclusion 5:** **N-**genes realize an ad-hoc *cascaded addressing scheme* in Wandering Networks upon evaluation of the shuttle and netbot (or clusterhead in case of a virtual sub-network) states.

The addresses of a software or hardware processing elements are encoded in *a*-genes which are altered every time a state matching procedure occurs. Thus, external IP-addressing can be combined with internal, *functional* addressing. For instance, the addressing inside an Active Processing Chip (Figure 94) is carried out according the following scheme, [TTL01]:

1. The ring protocol of a Hardware Processing Element (HPE) on Figure 101 transfers fixed size data units with a *busy/idle bit* in the first word of each transmission slot. The first word also includes a *flow control bit vector* with one bit for each IO Controller (IOC) on the chip; thus, an IOC can set its bit to signal congestion.
2. A second bit vector is used to enable *fair access to the ring*. Each plugin with data queued for transmission on the ring sets its bit and paces its transmissions on the ring based on the number of bits set by other plugins. Additional fields in this word identify *a ring and slot number* of the destination application for the packet.
3. The ring number identifies a unique processing element in the chain, while the slot number specifies the hardware plugin containing the destination application. For packets requiring processing by more than one application, the third bit vector is modified to address the next application. Upon completion of a packet, applications identify the correct ring and slot number of subsequent applications via locally available state information.

The above addressing scheme can be realized through a shuttle-netbot state matching procedure according to the WLI's *Dualistic Congruence Principle*.

Conclusion 6: The WLI model combines the advantages of:

1. *passive* media encoding techniques, recently developing towards multiple description coding, (the *Multidimensional Feedback Principle*), and
2. state-guided *active* processing which should be developed towards functional movement, (the *Pulsating Metamorphosis Principle*) by a programmable compact representation at the packet (shuttle) level composed of:
   - *active content* (transported data and/or code) as in traditional active packets;
   - *active semantics* in terms of:
     - interpretational ***data***, e.g. the address of an executing environment, the identifier of a particular plugin pipeline instantiation, [Deca99], or a linear representation of a tree structure (WARAAN, Chapter 6) used by EEs/AAs or by the semantics code within that gene, and/or
     - executable ***code***, a program which operates directly on the active content or interprets the semantics data in a gene) that modify/restore the active content) in terms of genes. Note that both parts of the shuttle do not need to be together; in this case the "action semantics" is reduced to the packet flow model.

7.3 SUMMARY AND OUTLOOK: NETWORK TECHNOLOGY INTERFACES

This section summarizes previous research and reviews the contribution of the WLI approach.

7.3.1 VISION AND REALITY: A CRITICAL OVERVIEW OF ACTIVE NETWORKING

Let us recall the basic characteristics an Active Network as they are committed to the state-of-the art technology.

1. A ***Passive Network*** (PN) is a packet switched IP network consisting of conventional non-terminal network elements (NEs) such as switches and routers, and terminals, such as clients or servers. The functions of the network components in a Passive Network are *fixed,* i.e. unchangeable.

2. An ***Active Network*** (AN) is an extension of the Passive Network, which permits a more flexible adaptation to a complex and changing communication environment by accommodating at least one programmable and/or reconfigurable non-terminal NE at run-time, known as Active ***Network Node*** (ANN).

3. Active Network Nodes are composed of three layers:
   - Execution Environments (EEs) which host Active Applications (AAs),
   - (Extensible) NodeOS, and
   - Programmable/ reconfigurable hardware.

The presence of at least the first one of the these layers classifies the NE as an Active Node; the ANN layers can be developed, programmed or reconfigured independently from the rest of the system architecture; the communication between the them is realized through *well-defined* interfaces.

4. ***Mobile code*** is transported in an Active Network via ***Active Packets*** (AP) or through downloads of ***active extensions/ plugins*** to build AAs on an Active Node. Of course, this does not exclude the presence of mobile agents in an Active Network, as it has been the case in a passive network.

5. Both active and passive networks are ***control architectures, which*** mean that they are regulating the network traffic based on a set of rules (protocols) which have been established a priori to deployment. Even if some recent research investigates the options of predicting network behavior ([Galt01a], [Galt01b], [Galt02]), it is still directed toward extending the control model framework. In the case of an Active Network, the control plane of an Active Node is split into two other sub-planes, ***network operator control plane*** and ***user control plane***. The latter enables the user to program the behavior of his/her application in response to network-centric signaling and to participate directly in the reservation and distribution of network resources on its path.

Active Networks of the first generation have achieved a substantial progress towards their main goal of providing a more flexible network layer, [TeWe96]. However, performance and security concerns are raised by the presence of *mobile code* in the network.

The most comprehensive evaluation of the experience with the design, implementation and deployment of Active Networks w. r. t. the original vision in their early research stage was given by Wetherall, [Weth99b]. In his report the author reviews the qualities of and the results achieved with the ANTS toolkit at MIT within three years of research (1996-1999).

Wetherall's evaluation is structured along three areas that characterize a "pure" active network:

- the capsule model of programmability;
- the accessibility of that model to all users; and
- the applications that can be constructed in practice.

The results of his analysis are summarized as follows:

1. **Capsules** have proved a worthwhile model, because they provide a clear solution for the automatic upgrade of processing along an entire network path. This model of deployment is *considerably more powerful* than the selective ad-hoc administrative upgrades practiced in packet switched networks today. However, the efficient implementation of capsules depends on the amount and on the demand (i.e. upgrading frequency) of code to be loaded, as well as on traffic patterns for which code caching becomes effective.

2. **Accessibility** by any un-trusted user who can freely customize the network was partly successful. The ANTS developers have managed to isolate different services from each other without trust or centralized control, but not to protect the network as a whole from un-trusted services.

To accomplish the latter in the general case, a fall-back solution of certification by a trusted authority was chosen until better solutions are found[163].

3. **Application** of capsules that appeared to be *most compelling* was the *network layer service evolution*, rather than the migration of application code to locations within the network. It was found that capsule code is well-suited to the task of introducing many variations of a service, and hence valuable for experimentation. Wetherall envisioned that capsule code would act in synergy with network embedded devices such as caches and transcoders, which are deployed by other means, for the benefit of both.

Active Network implementation scenarios, which confirm the above results, can be found elsewhere in the research literature. In the following, we provide a few examples, which illustrate the third argument.

Regarding the first and the third conclusion in [Weth99b], we argue that the enhancement of the capsule model by means of *genes*, as proposed in the WLI approach (Sections 7.2.3 and 7.2.4), will enable the transmission of context-sensitive details of the information content (including multiple feedbacks for its processing at the destination site), which not only extends the *secondary layering* (EEs/AAs to process capsules at the application layer) of the classical 'pure' AN model with a complementary level of virtualisation at the packet/capsule/shuttle side (Dualistic Congruence Principle), but also four characteristics of the genetic structure which provide are special advantages of the WLI model vs. other approaches:

1. Integration of diverse content encoding techniques[164] (incl. encryption).
2. Instant and implicit state transportation within shuttles which enables autopoietic load balancing throughout the different layers of the Wandering Network
3. Event-based hierarchical dynamic addressing of the processing elements
4. Network migration capability (for further study).

7.3.2 THE STEP AHEAD

The enhancements to the Active Network model proposed in the WLI framework are summarized as follows.

1. A *Wandering Network* (WN) is an extension, but an orthogonal construct when compared to an Active Network. *Fairness* as a restrictive rule in formal logic (TLA, [Lamp94]) is a key concept in WLI, which allows guaranteeing simplicity[165] in communications through *distributed*[166] *"self-control"*, i.e. *self-organization* rather than through delegating this function to dedicated elements in the network. It serves to report, verify and punish malicious behavior.

[163] The impact of security on performance issues was thoroughly investigated by Alexander et. al. on the SANE architecture, [ANSPrice]. In Section 7.2.3 we mentioned the evaluation of the processing overhead through encription in connection with the genetic organization of the shuttle. The interested reader is kindly asked to refer for further details on this subject to the above paper and recent studies in this area.

[164] referred in [Weth99b] as network embedded devices which are "employed by other means".

[165] No interactions that favor a "Byzantine General" type of behavior, [LSP82], are tolerated in a Wandering Network.

[166] A centralized control is not possible at all in a Wandering Network which is always "under construction".

Therefore, a WLI network can be realized as an additional overlay of an active ad-hoc mobile network, but all nodes within this overlay, are required to commit to fair behavior[167].

2. *Netbots* are *mobile, self-assembling* Active Nodes accommodating functionality both for their own use and requested by their peers. A netbot is characterized by the same three layers as the ones of an active node; however, *all of them should be present* to classify a network element as a netbot. Although these layers can emerge independently from each other and from and the rest of the system architecture, the communication between them is realized through *interfaces which develop along with the corresponding layers*. Automatic downloads of active extensions/plugins inside a netbot is performed also in terms of functional hardware components, thus enhancing an Active Node's capability.
3. *Shuttles* transport the mobile code in a Wandering Network; they accommodate a second level of encrypted/encoded *structured* soft-state information as n-*genes* along with the plain code-and-data contents of an Active Packet; they report/instruct about conditional changes in WN architectures. Genes enable network virtualization at the packet level in conformance to the *Dualistic Congruence Principle.*
4. Wandering Networks are *autopoietic, i.e. self-organizing and self-replicating/ architectures* which means that they are capable of realizing their own boundaries. The growth and the capacity of such a network, as well as its traffic regulation are based on a set of *network* service *usability rules,* organizational patterns identified as *WLI principles*. There is *no centralized network control plane* in a Wandering Network, because its functionality is in permanent movement and development (*Pulsating Metamorphosis Principle)*. The entire network topology and behavior depend on the *intensity of usage of its components,* guaranteed via a series of feedback mechanisms addressed in the *Multidimensional Feedback Principle*. The latter are organized as autonomous, but cooperating units, which *fairly* report (*Self-Reference Principle*), place requests to and distribute resources among their peers.

There are three principle distinctions between known AN approaches and WLI:

1. Whereas *classical AN approaches* evolve towards a) integration, interoperation and *higher layer virtualization and customization* of diverse execution environments *at the* application level networking which is b) eventually supported by a *moderate degree of hardware programmability* (which is going to grow in future), the *Wandering Logic Intelligence* provides the missing, complementary link in the chain of design solutions following the *Dualistic Congruence Principle* at its foundations – namely, the *specialization and virtualization of communication structures and processes at the packet level networking* through introducing diverse *classes of genetic code* (reflecting these communication structures and processes) to be automatically incorporated into the active packets (*shuttles*) by the active mobile nodes (*netbots*) when reaching certain system threshold values to enable *efficient and instant transfer of dynamic state information through encoding*, thus avoiding the netwide introduction of supplementary protocols, which increases the selective granularity of the programming architecture.

[167] This characterizes a Wandering Network as *autopoietic* structure dependable on realizing its own boundaries.

2. WLI extends the AN paradigm into mutually exchangeable and programmable functional blocks both in software and hardware (NOW) which are possibly interconnected via well defined interfaces at all system levels – (from hierarchies of network grids to molecular structures (and beyond) – which can develop into *autonomous, mobile, self-creating and self-organizing (autopoietic) communication architectures capable to grow, migrate and replicate* (in FUTURE).

3. *WLI is based on four principles for network evolution* which extend the classical end-to-end and layering principles of network design: *Dualistic Congruence, Self-Reference, Multidimensional Feedback and Pulsating Metamorphosis*.

The benefits of Wandering Networks vs. Active Networks were outlined in the last paragraph of section 7.2.3. A concise definition of the WLI innovations w. r. t. present network models was given at the end of Section 7.1.

In the following, we present a few selected practical scenarios in the areas of node and network management, QoS provisioning and ad-hoc routing, which demonstrate the new capabilities of a Wandering Network compared to pure AN based solutions. Finally, we discuss the adequacy of the WLI framework w. r. t. some recent industrial designs in Active Networking.

7.3.2.1 Intra-Node Management

The main limiting factors in Active Networking are *processing power* and *memory*. Therefore, a practical system implementation has to guarantee that these valuable resources are used in the most effective way.

Memory Management

This can be done via *tightly coupling* between a processing engine and the network, as well as between the processing engine and the switch backplane which ensures that packets arrive at an Active Network Node with minimal overhead through zero-copy DMA as proposed in [Deca99].

On the other hand, the WLI approach can provide a more flexible, policy-oriented solution based on the “in-band” exchange of state information contained in the shuttles. Of course, encoding this information per-se requires additional processing and storage capacity. However, this issue can be easily addressed in a practical way through option packet fields and FPGA filters, since the required router state information can be reduced to identifying some common patterns as in the case of intra-flow addressing discussed below.

In fact, because of its “structural” character, WLI can benefit from programming heuristics such as selective measures, e.g. from the fact that most network traffic is flow-oriented. In this case, bursts of packets share important forwarding properties that are, once determined, common to all packets of a particular flow.

This internal addressing scheme of a packet flow can not only be easily implemented within the shuttle model by dynamically accommodating arbitrary relational links which can be very important for multimedia traffic, e.g. the association of a packet content with an I, P or B frame of an MPEG stream, thus allowing the instant identification and processing of different flows (e.g. filtering and cut-through routing for the non-active packets and drop-out or redirection to a dedicated executing environment for the rest of them), but also provide critical per-flow traffic feedback information which can be used to predict undesired behaviour.

Power Management

In a real environment, the CPU power of a programmable switch should be rationalized to meet the demands of active processing of packets. Therefore, computation on active flows must be evenly distributed over the available processing engines. This can be achieved in two ways:

1. "internally", on a hard-wired base which allocates a set of resources a priori to programming (e.g. according one of the two FPGA design schemes shown on Figure 102) which could be then used by the Control Processor to distribute the traffic flows between the port processors; and

2. "externally" via system state traffic reports.

In the following, we discuss these two solutions in some detail because they clearly identify the conceptual and architectural differences between the available research approaches and the *Wandering Logic Intelligence*.

Power Solution 1

Decasper et al. [Deca99] apply a load sharing algorithm which dynamically distributes active flows over the ANPEs by configuring the corresponding APICs (setting/resetting cut-through switching of selected VCs) in order to move active flows from heavily loaded ANPEs to less loaded ones. Figure 102Figure shows an example of a data flow coming into the programmable router at Port Processor 1 (PP1) and going out at Port Processor 2 (PP2). The active processing in the case a) is done in the Processing Engine of the input port (1). Case b) illustrates the same operation with the alternative architecture which assumes a PE allocation from a dedicated pool of engines for active processing.

Each one of the programmable hardware architectures on Figure 102 can be optimized for a particular flow processing, provided that the designer knows about the traffic environment the router is going to be placed in and used. (We do not mean overprovisioning as a "rule of thumb" in this case, but rather a precise design.) In addition, a smart load distribution algorithm can take care for using this local infrastructure to the balance the traffic sharing inside the node. This is how the link capacity can be internally monitored and managed by Central Processor.

However, even such a programmable architecture has a limited capacity and flexibility because of the pre-determined nature of hardware design, which cannot be configured for more than that, that has been once in mind. In the strong sense, this note holds also for software which can be transported, installed and even verified during run-time of a telecommunication system, but cannot automatically “emerge”, i.e. be created on the fly (yet).

Of course, a smart design such as the one in [Deca99] can guarantee scalability through the ability to configure any number of Port Processors which can be added *by someone* to the Active Network Node along with extensions of the switch fabric. But this is all what can be done today.

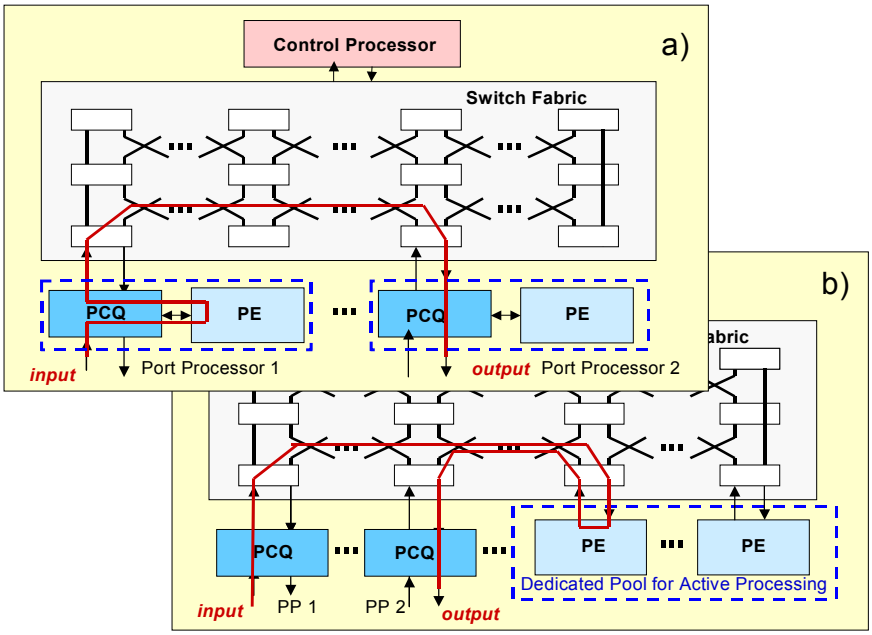


Figure 102: Programmable router architectures: (a) system organization with a processing engine at each port;(b) with a shared pool of processing engines

The WLI approach projects two other “hardware” options in future architectures along with active mobile nodes, which can be also designed as netbots:

1. Autonomous On-Demand Mobile Hardware Self-Plugins
2. *Autopoietic* (Self-Assembling) Nanoscale Processing and Storage Elements

The main argument for such a solution are possible network-centric actions targeting to intelligently upgrade/degrade a netbot’s capacity or performance as result of evaluating multiple feedback mechanisms from the surrounding environment. In this way shuttle processing can not only be distributed and optimised between the Central Processor and the Port Processors using internal monitoring mechanisms, but also extended/enhanced by additional “external” hardware modules in the long run.

Power Solution 2

Originally, this kind of traffic control has been realized through policy management architectures (*IntServ, DiffServ*) and solutions based on standard reservation and synchronization protocols such as RSVP [Brad97], RTSP [SRL98] and RTP [SCFJ99], which negotiate bandwidth and resources above the network and transport layers a priori to transmission. The details of the management are left to the operating system.

The utilization of an Active Node's processing capacity at the network layer, as discussed in the previous "hard-wired" scenario, is not optimal in the general case, because it requires the same knowledge about the same processing elements in the programmable routers/switches of a usually heterogeneous network.

Of course, the selection and the distribution of active packets for the purpose of effective CPU performance can be successfully realized on a per-port base through a particular local load equalizing scheme. For instance, Decasper et al. report about developing an *intra-ANN protocol* to communicate the status of processing engine load between ANPEs on a reserved VC, [Deca99]. Other authors such as Galtier et al. model and measure CPU demand in Active Networks to predict, plan and implement detailed resource distribution policies at he network layer ([Galt01a], [Galt01b], [Galt02]). However, it is not said that this traffic optimisation should be necessarily based on such information as the knowledge about the *origin*, *content* and *type* of the flow/packet, and this – at the port level.

Active Networks allow a novel mechanism for dynamic integration of new protocols for packet processing in the virtual machines (EEs) of the application level, which can be used in addition to the passive techniques. However, virtualization at the application level only may unnecessary increase the processing complexity and the CPU load, because decisions about traffic management are delegated to the EE.

With *Power Solution 2* we do not only mean the usual messaging mechanisms used by higher-level protocols in passive networks or NodeOS pipelining. WLI offers another, more powerful option for "instant messaging" and "adaptive programming" of traffic conditions which can be encoded and transported through *genes*, the second abstraction layer of the shuttle architecture which complements the virtualisation architecture of Active Networks at the network layer.

7.3.2.2 Inter-Node Management

The following example is going to review in some detail the advantages of the WLI approach for distributed network management. The AN architecture is taken from an AT&T Bell Labs research paper, [RaSh00], which illustrates the sustaining industries' view on Active Networking a typical effort (on a system known as ABLE) to enhance legacy routers with an adjunct active engine which enables the safe execution and rapid deployment of new distributed applications in the network layer, Figure 103.

The authors claim that the system "can be gradually integrated in today's IP network" [correct] "to allow smooth migration from IP to programmable networks".

The proposal is realistic and interesting with its SNMP foundations, but in our view runs the danger of presenting a distorted view at the original vision of Active Networks, [TeWe96]. Instead, it demonstrates how future generations of HyperActive and Wandering Networks should not be built and utilized. This is not because the solution is technically bad in any way, but because it could be misleading. Our reasons on this subject are given in Section 7.2.4.
In the following, we review and discuss the above system.

An active node in the [RaSh00} system is comprised of two entities:

- an IP router, and
- an adjunct active engine (AE).

This simple constitution in two “boxes” along the client-server paradigm, which emulates an assisting Intelligent Peripheral known from Intelligent Networks, [Fayn97], although very simple and practical, disintegrates already the Active Networking idea in its core and creates an unnecessary communication overhead between the separated entities. Although the authors claim the clear separation of active and non-active functional blocks throughout all network layers, this approach does not facilitate processing.

The IP router performs the IP forwarding, basic routing, and filtering which is part of the functionality provided by today's COTS IP routers. The IP filtering enables the diversion of active packets (or other packets) based on their IP/UDP headers to the active engine. The latter option is implemented as standard hardware in the new generation of IP routers, [KLS98], and can be performed by edge routers in software. Therefore, the authors propose to use it for the implementation of an “outsourced” EE, which is the Active Engine in fact.

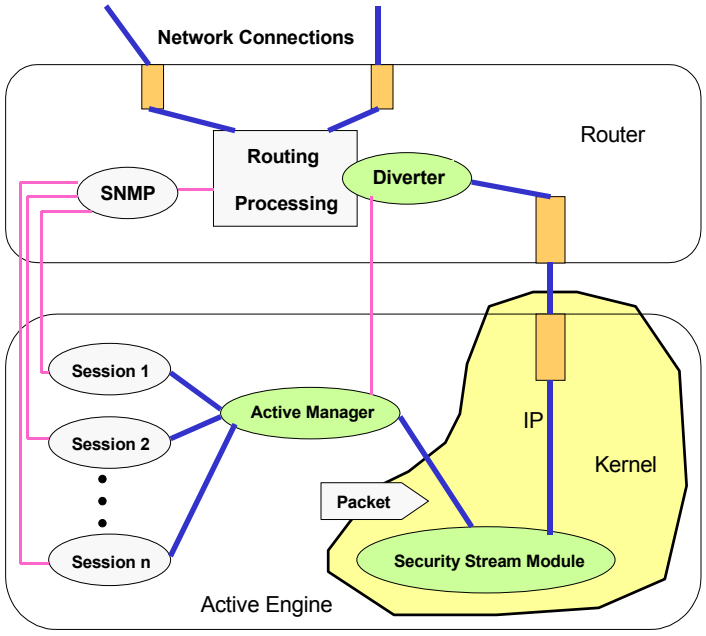


Figure 103: Overall Architecture of the ABLE system, [RaSh00]; thick lines between components illustrate possible flows of data, thin lines – logical connections.

The Active Engine is defined as “an environment in which code encapsulated in active packets can be executed”. This code “can specify *HOW* code and data related to a specific task should be handled”. This function is similar to the specialized IP Execution Environment in [Deca99] discussed earlier in this chapter; therefore it does not represent an innovation, and hence cannot obtain a patent per se.

In terms of WLI, the Active Engine should be a dedicated processing element which interprets shuttle *genes* either at the application layer (as an EE), or as a programmable hardware component at the platform layer (Port Processors). In short, the AE is a “hardware overhead” that can be easily put inside an FPGA of a programmable router, where the entire SNMP signalling on Figure 103 can be resolved on the internal PP bus between the PCQ and the APC, Figure 94.

Hence, the only reason to support this uneasy architecture might be reuse of old infrastructure and protocols which never developed beyond their own limits, a fact that rather created the demand for Active Networking. Furthermore, the AT&T prototype system considers a globally unique number (a session ID) to identify a logical distributed task. When code associated with a non-existing session arrives, it is executed and creates a process that handles all the packets of that session. Note that the code should be “strongly typed”, i.e. the system on Figure 103 can recognize only one active flow. This means that in order to implement the ANN/DAN system on Figure 95 we will need at least three external boxes for each single Execution Environment.

In addition, the AT&T system uses SNMP as the interface between the router and the AE to provide sessions with access to the router's network layer data (neighbour IDs, routing tables, performance statistics, etc. That the SNMP protocol can easily transport useful information between network entities is a well known fact. However, standard SNMP agents on all routers which provide read/write interface to standard management information bases (MIB) does not know anything about the application running on top of them. Therefore, SNMP requires a large communication overhead to discover and transfer e.g. the QoS parameters of the lightest change of a transcoding rate adaptation on an MPEG-4 stream which though cannot be performed in due time in dynamic environments such as ad-hoc mobile networks. However, the same information can be easily delivered within a capsule or shuttle to be processed on-board of the Active Node with access to required network context information (Self-Reference Principle).

To perform distributed tasks, an active node must have means to communicate with other active nodes. However, the authors’ note that most of the research on Active Networks “failed to notice the need to access local network information” (provided via SNMP in their case) is incorrect. Active Networking was not designed to support reuse of old protocols and standards, but to introduce new ones. They provide other means to transfer network layer related information. For instance, in Chapter 6 we demonstrated the easy encryption of reachability trees within shuttles to report topology updates to the neighbourhood.

Therefore, the goal of the AT&T prototype to use AN technology for managing IP networks fails because they can be managed *also without Active Networks*. IP networks become only more complex this way. It is much easier to replace passive with active routers to enhance their flow processing capabilities.

The same reasoning can be applied for the implementation of Wandering Networks. A WLI architecture can be realized following the design guidelines in Section 7.2. However, its success depends on the selected domain of application. Thus, a WN implementation e.g. as an overlay network within the X-bone may show a completely different results in the end-to-end quality and performance evaluation when compared to the ones achieved in a “stand-alone” solution. But this does not necessarily mean that the WLI approach should be wrong. It is only *new, a fairly new* one compared to present day telecommunications architectures.

In Section 7.1 we stated that the four WLI principles of *autopoiesis* presented in Chapter 5 are an extension of today’s end-to-end design and system layering principles. In fact, a Wandering Network *extends the control plane* to each single node without affecting the control planes of other nodes. It is important to realize that this approach is *orthogonal* to today’s control architectures by addressing activity, i.e. *self-organisation in its core*, [Sim02a].

For this reason, a Wandering Network, while being a new class of a *hyperactive* architecture that is closely related to some recent approaches to extended network activation (e.g. RFC proposals such as [SJC00]); it still cannot easily interoperate with the “*control* type” architectures even if they are deployed as network overlays.

This is because of the introspective organization of the Wandering Network which is undeterministic and *event-based* (Section 7.2.4) in its nature, based on the instant local evaluation of shuttle and netbot states, and cannot be “controlled” or predicted via traditional management protocols in the common fashion practiced today in passive (SNMP) and active networks (e.g. AVNMP, [Bush00],[Bush01]). Thus, although providing an extended system model, the closest interface at which WLI can “speak” with other, classical types of packet networks is active ad-hoc mobile networking. This issue was demonstrated in Chapter 6 with the WARAAN routing algorithm.

7.3.2.3 A Comment on Security

The WLI framework complies with the state-of-the-art research in AN security. In fact, the idea of including context-sensitive networking information inside individual active packets of a flow by introducing a second virtualisation layer in terms of specialized *genes*, as WLI proposes, represents a general abstraction for isolating network flows by means of a programmable set of security sources/reasons which can be handled (if desired) on an event base. For instance, WLI can easily accommodate efficient encryption techniques such as *distributed fingerprinting*[168]*,* recently proposed in [Brown01].

However, in the case of a Wandering Network, the genetically "encrypted" information can be used not only for the sophisticated flow authorization, but also for maintaining a *floating* (or “wandering”) system state between the netbots as noted in section 7.2.

[168] A security technique used in particular for the encription of media flows which are marked with the identity of their recipients.

7.3.2.4 A Comment on Performance

As discussed in Chapter 3, Active Networks allow applications to inject customized programs into network nodes, thus enabling faster introduction and deployment of new network protocols and services even over the wide areas.

In [LWG98], Legedza, Wetherall, and Guttag argue that the ability to introduce active protocols offers important opportunities for end-to-end performance improvements of distributed applications. They describe a variety of active protocols that provide novel useful network services (active reliable multicast, online auctions, mixing sensor data) and discuss their potential impact on end-to-end application performance. The authors have shown that such active network services can increase their performance through processing at intermediate nodes of the network. In particular, the performance of an active networking protocol that uses caching within the network backbone was presented and analysed. The study has demonstrated that an Active Network solution is able to support caching of (previously considered for uncacheable) rapidly changing data which leads to load reduction on both servers and backbone routers as well as shorter round-trip hop counts.

The trade-off between performance and security in Active Networks has been addressed by several authors ([ANSPrice], [Weth99b]). In fact, we can apply the above conclusions for the theoretical evaluation of the WLI model which was carried out in Sections 7.2.3 and 7.3.1. Of special importance are the “feasibility” arguments provided at the end of Section 7.3.2.2

7.4 DIRECTIONS FOR FUTURE RESEARCH

This section completes the demonstration of the theoretical aspects of the WLI approach. The major next step is the implementation of the WARAAN routing algorithm presented in the WLI case study (Chapter 6). Further work in terms of test and verification of features is expected and required to prove that the proposed model is powerful enough and well suited to address network evolution at this stage of research. That will be the subject of future research for which a funding proposal is planned.

The WLI approach that has been presented is currently at a theoretical stage of work. It does not yet have any practical value. However, we have provided a well-defined informal model (Chapter 4) with a small part of it demonstrating its hidden capabilities in the body of a formally specified and checked adaptive ad-hoc routing algorithm (Chapter 6).

We have also provided a strong argumentation for and proof of the practical feasibility of the WLI paradigm, which was illustrated with guidelines for implementation discovering the true nature of the model and examples for its application, compared to alternative solutions (Chapter 7). Therefore based on the strength of the arguments put forward in Chapters 4, 6 and 7, we have a solid basis upon which to prepare a research proposal with a good chance of success.

Network evolution means not only a (controlled) gradual improvement of the state-of-the-art solutions, but also discontinuing of the solutions which have significant shortcomings that does not properly match real life needs.

At the same time, this does not mean that well established design principles, technology infrastructures and investments should be disregarded merely for the purpose of innovation. "Service usability"[169] provides the required benchmarks for establishing new technologies. The best way to ensure that this principle is adhered to is to integrate it within the design process as an immediate feedback routine which automatically determines the behaviour of the network. With WLI we present our view at this development.

In our approach, we simply recognize the "rule of purposefulness" stating that by establishing a new paradigm, engineers should ***continue to focus on, but carefully navigate towards the original vision***, – even when meeting at times substantial difficulties to practically realize the new idea –, instead of narrowing the horizon and sticking up to reconciliation with and improvement of the old concept "at any price". This is the special characteristic that distinguishes disruptive technology from the sustaining one, [Chris97].

Finally, we are going to summarize the open research issues which have not been addressed in this thesis work so far and are challenging for future research from our perspective.

The formal part of this thesis (Chapter 6) illustrates the application of the WLI approach in the area of active ad-hoc mobile networking. The WARAAN routing protocol is designed to demonstrate a superior performance (O (4)) over present day routing table updates with the r-shuttle reachability tree encoding scheme. The algorithm was entirely specified and tested in the formal technique TLA (Temporal Logic of Actions), [Lamp94]. However, a detailed proof of the algorithm needs to be provided. This is going to be our next objective.

Furthermore, it is desirable that also other interesting WLI characteristics are specified and verified in TLA or other formal methods. This work should be provided by future research to prove the concept consistency and interdependence of the *Four WLI Principles* w. r. t. their practical usability. The latter should find expression in a set of properties which are related to both functional and non-functional aspects of networking and distributed processing. In particular, network properties for routing, portability, migration, reconfiguration, security and performance in terms of software and hardware should be defined to enable the formal evaluation of system implementations, as well as to provide benchmarks for comparative measurement and analysis of the different solutions at the service level. Promising research in this area has already begun ([Turn01a], [Turn01b]).

Also, it is of particular interest to investigate *how well* TLA and other formal techniques (LOTOS, Chisel, PVS, Maude) stand up to the above objectives.

Special attention deserves the concept strengthening of the autopoietic mechanism for dynamic event-based shuttle-netbot state evaluation for resource addressing and management described in 7.2.4 w. r. t. formal verification, implementation, measurement and test.

Another challenging research topic is the new role of reflection mechanisms in packet networks affected by the introduction of a second level of virtualization in the face of *n*-genes.

---

[169] the tautology of this term is intentional and preferred instead of the even more unusual "serviceability".

Also, some aspects of reconfigurability presented in Chapter 4 are reminiscent of *reflection* as discussed in programming languages. This is an interesting research topic that could be pursued in more detail in the light of a particular implementation of the WLI/WN approach such as the WARAAN routing algorithm (Chapter 6).

Further Interesting research questions are:

- Which role has middleware in AN and WLI ?
- What implications do the concepts of Active Networks and WLI have on:
    - hardware and software used to support network design (e.g. node/netbot architecture, capsule/shuttle code structure, high-level languages and systems used, etc.) and vice versa;
    - portability and migration and vice versa;
    - performance, efficiency, security and vice versa.

* * *

" There is latent in Cybernetics the means of achieving a new and perhaps more human outlook, a means of changing our philosophy of control … "

GREGORY BATESON, "STEPS TO AN ECOLOGY OF MIND", 1972

GLOSSARY

AA	Active Application
AAMN	Active Ad-hoc Mobile Network
Abone	Active Network backbone
ABR	Associatively Based Routing
ACC	Active Congestion Control
ACD	Adaptive Communication Device
AE	Active Engine
AES	Advanced Encryption Standard
AI	Artificial Intelligence
ALAN	Application Level Active Networking
ALF	Application Layer Framing
ALU	Arithmetic-Logic Unit
AMN	Ad-hoc Mobile Network
AN	Active Network
ANEP	Active Network Encapsulation Protocol
ANN	Active Network Node
ANON	Active Network Overlay Network
ANPE	Active Network Processing Element
ANTS	Active Node Transfer System
AODVR	On-demand Distance Vector Routing
AP	Active Packet
APC	Active Processor Chip
API	Application Programming Interface
APIC	ATM Port Interconnect Controller
ARC	Active Router Control
ARM	Active Reliable Multicast
AS1	Active Services version 1
ASIC	Application Specific Integrated Circuit
ASCP	Active Services Control Protocol
ATM	Asynchronous Transfer Mode
ATP	Agent Transfer Protocol
AVNMP	Active Virtual Network Management Prediction
BGP	Border Gateway Protocol
B-IP	Broadband Intelligent Peripheral
CAM	Content Addressable Memories
CAP	CAMEL Application Part (Protocol)
CANE	Composable Active Network Element
CCF	Call Control Function
CCM	Custom/Configurable Computing Machines
CCSS	Centralized Client/Server Subnetwork
CDMA	Code Division Multiple Access
CE	Communication Environment
CF	Configuration Function
CGSR	Clustered Gateway Switch Routing
CISC	Complex Instruction Set Computing
CLB	Configurable Logic Block
CORBA	Common Object Request Broker Architecture
COTS	Components-Of-The-Shelf
CP	Control Processor
CPE	Customer Premises Equipment
CPU	Central Processing Unit
CRC	Cyclic eRror Correction

CS-2	Capability Set 2
CSI	Customer Services Interface
CSLIP	Compressed Serial Line Internet Protocol
CTI	Computer-Telephony Integration
CU	Conference Unit
DAN	Distributed code caching for Active Networks
DARPA	Defence Advanced Research Projects Agency
DAS	Dual Availability Subnetwork
DCP	Dualistic Congruence Principle
DES	Duplex Exchange Subnetwork
DHP	Dynamic Hardware Plugin
DIRS	Distributed Information Retrieval Subnetwork
DLL	Data Link Layer
DPS	Dynamic Proxy Server
DRAM	Dynamic Random Access Memory
DREN	Defense Research and Engineering Network
DRNS	Drift Radio Network System
DSDV(R)	Destination Sequenced Distance Vector (Routing)
DSFS	Distributed Store & Forward Subnetwork
DSP	Digital Signal Processing / Processor
DSR	Dynamic Source Routing
DSS1	Digital Subscriber Signaling System No. 1
E2E	End-to-End
E2EA	End-to-End Argument(s)
EE	Execution Environment
ETSI	European Telecommunication Standards Institute
FDDI	Fiber Distributed Data Interface
FDMA	Frequency Division Multiple Access
FEC	Forward Error Correction
FIFO	First-In-First-Out
FINE	Flexible Intelligent Network Element
FINEA	FINE Architecture
FM	Filter Memory
FORTH	FO(U)RTH generation computers programming language
FPGA	Field Programmable Gate Array
FSL	Flexible Service Logic
GEDIR	GEographic Distance Routing
GIS	Geographic Information System
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HAG	Higher-order Attributed Grammar
HCI	Human Computer Interface
HLR	Home Location Register
HPE	Hardware Processing Element
HTTP	HyperText Transfer Protocol
IC	Integrated Circuit
ID	IDentifier
IDL	Interface Description Language
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMAP	Interactive Mail Access Protocol
IMT-2000	International Mobile Telephone Standard 2000
INAP	Intelligent Network Application Part (Protocol)
I/O	Input / Output
IOC	Input Output Controller

IP	Internet Protocol / Intelligent Peripheral
IPv *	Internet Protocol version * (e.g. 4, 5 or 6)
ISDN	Integrated Services Data Network
ISO	International Standards Organization
ISUP	ISDN User Part (Protocol)
I(P)TO	Information (Processing) Technology Office at DARPA
ITU	International Telecommunications Union
JIT	Just In Time
JTAPI	Java Telephony API
KDE	K Desktop Environment, an Open Source GUI for UNIX
LAN	Local Area Network
LCE	Logical Computing Element
LSI	Large Scale Integration
LUT	Look-Up Table
MAC	Medium Access Control
MAP	Mobile Application Part (Protocol)
Mbone	Multicast backbone
MFP	Multidimensional Feedback Principle
MH	Mobile Host
MIB	Management Information Base
MIMD	Multiple Instruction Multiple Device
MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
MN	Mobile Network
MPoA	Multiple Point of Access
MPEG	Motion Picture Expert Group
MPLS	Multi-Protocol Layer System
M-SCP	Mobile Service Control Point
MT	Mobile Terminal
NASA	National Aeronautics and Space Administration
NAT	Network Address Translation
N-geneering	Node/Network Engineering
NodeOS	Node Operating System
NFS	Network File System
NPE	Network Programming Environment
NPU	Network Processing Unit
NVN	NetScript Virtual Network
ODP	Open Distributed Processing
OFDM	Orthogonal Frequency Division Modulation
ONS	One Number Service
OPENSIG	OPEN SIGnaling
OS	Operating System
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PE	Physical Element
PEP	Performance Enhanced Proxy
PAN	Practical Active Network
PANTS	Python Active Node Transfer System
PAL	Programmable Array Logic
PC	Packet Classifier / Personal Computer
PCI	Programming Control Interface
PCMCIA	Personal Computer Memory Card International Association
PCQ	Packet Classification and Queuing (chip)
PCS	Personal Communication Services
PDCP	Packet Data Control Protocol
PLAN	Packet Language for Active Networks
PLMN	Public Land Mobile Network

PoP	Point of Presence
POP3	Post Office Protocol-3
POTS	Plain Old Telephone Service
PMP	Pulsating Metamorphosis Principle
PN	Passive Network
PP	Port Processor
PPP	Point-to-Point Protocol
PSTN	Public Switched Telephone Network
PVS	Program Verification System
QCTL	Queuing ConTroLler
QM	Queuing Memory
QoS	Quality of Service
RADIUS	Remote Access Data Interface User Service
RAM	Random Access Memory
RCI	Resource Platform Control Interface
RCM	Reconfigurable Computing Machine
RFC	Request For Comments
RISC	Reduced Instruction Set Computing
RLC	Radio Link Control
RNC	Radio Node Controller
RNS	Radio Network System
ROSE	Remote Operations Service Element
RP	Resource Platform
RPC	Remote Procedure Call
RPU	Reconfigurable Processor Unit
RRC	Radio Resource Control
R/S	Router / Switch
RSCP	RADIUS Service Control Point
RSVP	Resource ReSerVation Protocol
RT	Reachability Tree
RTCP	Real Time Control Protocol
RTP	Real Time Protocol
RTR	Run Time Reconfiguration
RTSP	Real Time Streaming Protocol
SANE	Secure Active Network Environment
SAPF	Simple Active Packet Format
SCF	Service Control Function
SCI	Switch Control Interface
SCP	Service Control Point
SDF	Service Data Function
SDU	Service Data Unit
SI(B)B	Service Independent (Building) Block
SIC	Stream Interface Controller
SIMD	Single Instruction Multiple Device
SIOD	Source-Initiated On-Demand Driven
SIP	Session Initiation Protocol
SL	Service Logic
SLIP	Serial Line Internet Protocol
SM	Switching Matrix
SMF	Service Management Function
SMP	Service Management Point
SMTP	Simple Mail Transfer Protocol
SN	Service Node
SNAP	Safe and Nimble Active Packets
SNC	Service Node Controller
SNMP	Simple Network Management Protocol
SoC	System on a Chip

SPE	Software Processing Environment
SRF	Specialized Resource Function
SRAM	Static Random Access Memory
SRNS	Serving Radio Network System
SRP	Self-Reference Principle
SS7	Signaling System 7
SSF	Service Switching Function
SSP	Service Switching Point
SSR	Signal Stability Routing
STM	Synchronous Transfer Mode
TCL	Tool Command Language
TCP	Transmission Control Protocol
TD	Table-Driven
TDMA	Time Division Multiple Access
TI	Transmission Interface
TID	Type IDentifier
TINA	Telecommunications Information Network Architecture
TL	Temporal Logic
TLA	Temporal Logic of Actions
TMN	Telecommunications Management Network
TORA	Temporary Ordered Routing Algorithm
TTL	Time To Live
UDP	User Datagram Protocol
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
VAN	Virtual Active Network
VC	Virtual Circuit
VHE	Virtual Home Environment
VL	Virtual Link
VLIW	Very Large Instruction Word
VLR	Visitor Location Register
VLSI	Very Large Scale Integration
VoD	Video on Demand
VM	Virtual Machine
VNE	Virtual Network Engine
VP	Virtual Path
VPN	Virtual Private Network
VR	Virtual Reality
VRS	Virtual Resource Set
WAN	Wide Area Network
WAP	Wireless Access Protocol
WARAAN	WLI Adaptive Routing Algorithm for Ad-hoc Networks
WebCSC	Web based Customer Services Control
W-CDMA	Wide band CDMA
WLI	Wandering Logic Intelligence
WN	Wandering Network
WRP	Wireless Routing Protocol
X-bone	A System for Automated Overlay Network Deployment
XTP	Xpress Transfer Protocol
2D	Two-Dimensional
1G	First Generation
2G	Second Generation
3G	Third Generation
4G	Fourth Generation

BIBLIOGRAPHY

ACTIVE NETWORKING

[Abone] Abone, Active Network Backbone, http://www.csl.sri.com/ancors/abone/.

[ACKL98] O. Angin, A. T. Campbell, M. E. Kounavis, R. R.-F. Liao, "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking", *IEEE Personal Communications Magazine, Special Issue on Adaptive Mobile Systems*, August 1998.

[Alex96] D. S. Alexander, "A Generalized Computing Model of Active Networks", *Thesis Report*, University of Pennsylvania.

[Alex97] D. S. Alexander, B. Braden, C. A. Gunter, W. A. Jackson, A. D. Keromytis, G. A. Minden, D. A. Wetherall, "Active Network Encapsulation Protocol (ANEP)", http://www.cis.upenn.edu/switchware/ANEP/docs/ANEP.txt, *Active Networks Group Draft*, July 1997.

[Alex98a] D. S. Alexander et al., *"*The SwitchWare Active Network Architecture*"*, *IEEE Network*, Special Issue on Active and Programmable Networks, May/June 1998.

[Alex98b] D. S. Alexander et al., "Safety and Security of Programmable Network Infrastructures*"*, *IEEE Communications*, Oct. 1998, pp. 84-92.

[Alex99] D. S. Alexander et al., "Security in Active Networks," in: *Secure Internet* Programming: Issues in Distributed and Mobile Object Systems, Springer-Verlag, Lecture Notes in Computer Science *State-of-the-Art* Series, 1999.

[ALLM97] C.M. Adam, A. A. Lazar, K.-S. Lim, F. Marconcini, "The Binding Interface Base Specification Revision 2.0", *OPENSIG Workshop on Open Signaling for ATM, Internet and Mobile Networks*, Cambridge, UK, April 1997.

[AMK98] E. Amir, S. McCanne, and R. Katz, "An Active Service Framework and its Application to real-time Multimedia Transcoding", *Proceedings ACM SIGCOMM' 9*8, Vancouver, Canada.

[Arb98] W. A. Arbaugh *et a*l., "Automated Recovery in a Secure Bootstrap Process", *Network and Distributed System Security Symposium*, Internet Society, March 1998.

[ASNS97] D. S. Alexander, M. Shaw, S. M. Nettles, J. M. Smith, *"*Active Bridging*", Proc. SIGCOMM'97*, ACM, Cannes, France, 1997.

[ATab96] A. Adl-Tabatabai, et al., *"*Efficient and Language-Independent Mobile Programs*"*, *ACM SIGPLAN Conf. Programming Language Design and Implementation*, PLDI'96, Philadelphia, ACM, 1996.

[ATP] http://www.trl.ibm.co.jp/aglets/atp/atp.htm.

[AVNMP] http://avnmp.sourceforge.net/.

[Bakin97] D. Bakin, W. Marcus, A. McAuley, T. Raleigh, "An FEC Booster for UDP Application over Terrestrial and Satellite Wireless Networks", *Proc. IMSC'97*, 1997.

[Banchs] Albert Banchs *et a*l., "Multicasting Multimedia Streams with Active Networks", *ICSI Technical Report* 97-050.

[BCK97] A. Balachandran, A. T. Campbell, M. E. Kounavis, "Active Filters: Delivering Scalable Media to Mobile Devices" , *Proc. Seventh International Workshop on Network and Operating System Support for Digital Audio and Vide*o, St Louis, May, 1997.

[BCZ96] S. Bhattacharjee, K. L. Calvert, and E. W. Zequra, "Implementation of an Active Networking Architecture", *Technical Report*, Georgia Institute of Technology, July 1996.

[BCZ97] S. Bhattacharjee, K. L. Calvert, and E W. Zequra, "An Architecture for Active Networking", *Proc. IEEE INFOCOM '97*, April 1997.

[BCZ98] S. Bhattacharjee, K. L. Calvert, E. W. Zegura, *"*Self-Organizing Wide-Area Network Caches*"*, *Proc. IEEE Infocom'98*, San Francisco, CA, March 1998.

[BCZ98a] S. Bhattacharjee, K. L. Calvert, E. W. Zegura, *"*Reasning About Active Network Protocols*"*, *Proc. ICNP'98*, Austin, TX, Oct. 1998.

[Bhat96] Samrat Bhattacharjee, Ken Calvert, and Ellen W. Zegura. On Active Networking and Congestion. Technical report, Georgia Institute of Technology, 1996. Technical Report GIT-CC-96-02.

[Bhat97] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura, "Active Networking and the End-to-End Argument", *Proc. ICNP '97,* Atlanta, GA, Oct. 1997.

[Bhat98] S. Bhattacharjee *et al*., Commentaries on "Active Networking and End-to-End Arguments," *IEEE Networ*k, special issue on Active and Programmable Networks, May/June 1998, vol. 12, no. 3.

[Bis98] J. Biswas et al., "The IEEE P1520 Standards Initiative for Programmable Network Interfaces*", IEEE Communications*, Oct. 1998, pp. 64-70.

[BKEG00] S. F. Bush, A. Kulkarni, S. Evans, L. Galup, "Active Jitter Control", *Proc. of ISN'00* (7th Int. IS&N Conference, Intelligence in Services and Networks), Athens, Greece, Feb. 23-25, 2000.

[BKGM00] J. Border, M. Kojo, Jim Griner, G. Montenegro, "Performance Enhancing Proxies", Internet Engineering Task Force, Internet-Draft (draft-ietf-pilc-pep-03.txt), work in progress, July 2000.

[Bore94] N. Borenstein, *"*Email with a Mind of its Own: The Safe-Tcl Language for Enabled Mail", *Proc. IFIP Int'l. Conf.*, Barcelona, Spain, 1994.

[Bra98] Bob Braden, Correspondence, Another Modest Proposal for the ABone, November 1998.

[Bush00] S. F. Bush, "Islands of Near-Perfect Self-Prediction", *Proc. of VWsim'00* (Virtual Worlds and Simulation Conference), c/o WMC'00 (2000 SCS Western Multi-Conference), San Diego, January, 2000.

[Bust01a] F. E. Bustamante, G. Eisenhauer, P. Widener, K. Schwan, C. Pu. "Active Streams: An Approach to Adaptive Distributed Systems". Proc. 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), Elmau/Oberbayern, Germany, May, 2001.

[Bust01b] F. E. Bustamante, Greg Eisenhauer, Karsten Schwan and Patrick Widener. "Active Streams and the Effects of Stream Specialization". Proc. of Tenth International Symposium on High Performance Distributed Computing (HPDC-2001), San Francisco, California, August 7-9, 2001.

[BuKu01] S. F. Bush, A. Kulkarni, "Active Networks and Active Network Management: A Proactive Management Framework", *Kluwer* Academic/Plenum Pub. Boston, March 2001, 200 pp. Hardbound, ISBN 0-306-46560-4.

[BWG99] V. Bose, D. Wetherall, J. Guttag, *"*Next Century Challenges: RadioActive Networks*"*, *Proc. of the ACM/IEEE Intnl. Conf. on Mobile Computing and Networking Mobicomm'99*, Seattle, WA, August 1999, http://www.vanu.com/techpapers.html.

[Cald98] M. Calderon, M. Sedano, A. Azcorra, C. Alonso, "Active Network Support for Multicast Applications", IEEE Network, 12 930;46-52, Mai/Juni 1998.

[Calv98] K. L. Calvert, S. Bhattacharjee, E. Zegura, J. Sterbenz, "Directions in Active Networks," *IEEE Communications Mag.,* Oct. 1998, pp. 72-78.

[Camp99a] A. Campbell, H. De Meer, M. Kounavis, K. Miki, J. Vicente, D. Villela, *"*A Survey of Programmable Networks*"*. http://www.columbia.comet.edu.

[Camp99b] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela, "The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures", *Second International Conference on Open Architectures and Network Programming (OPENARCH'99*), New York, 1999.

[Camp99c] A. T. Campbell, M. E. Kounavis, D. Villela, J. B. Vicente, K. Miki, H. G. De Meer, K. S. Kalaichelvan "Spawning Networks", *IEEE Network Magazine*, July/August, 1999.

[Camp99d] A. T. Campbell, A.T., J. Vicente, and D. A. Villela, "Managing Spawned Virtual Networks", *Proc. International Working Conference on Active Networks (IWAN'99),* Berlin, July 1999.

[Camp01] A. T. Campbell, S. Chou, M. E. Kounavis and V. D. Stachtos, "Implementing Routelets: Virtual Router Support for the IXP1200 Network Processor", *IXA Univeristy Program Workshop*, Portland, Oregon, June 2001.

[CANES] E. Zegura, "CANEs: Composite Active Network Elements*"*, Georgia Tech., http://www.cc.gatech.edu/projects/canes/.

[Chan96] M.-C. Chan, J.-F. Huard, A. A. Lazar, K.-S. Lim, "On Realizing a Broadband Kernel for Multimedia Networks", *3rd COST 237 Workshop on Multimedia Telecommunications and Application*s, Barcelona, Spain, November 25-27, 1996.

[Chan98a] P. Chandra et *a*l., "Darwin: Customizable Resource Management for Value-Added Network Services", *Proc. Sixth IEEE Int'l Conf. on Network Protocols (ICNP '98),* Austin, TX, Oct. 1998.

[Chan98b] P. Chandra et *a*l., "Network Support for Application-Oriented QoS", *Sixth IEEE/IFIP International Workshop on Quality of Service*, Napa, May 98.

[ChDN96] A. Chankhuntod, P. B. Danzig, and C. Neerdaels, *"*A Hierarchical Internet Object Cache*"*, *Proc. 1996 USENIX*, 1996.

[ChJa98] Chen, T., Jackson, A. W., (Eds), "Commentaries on Active Networks and End-to-End Arguments", *IEEE Network*, May/June 1998, pp. 66-71.

[Chin00] K.-W. Chin, "An Investigation into the Application of Active Networks to Mobile Computing Environments", *Ph.D. Thesis*, Curtin Univ. of Technology, 2000.

[CHLL96] M.-C. Chan, J.-F. Huard, A. A. Lazar, and K.-S. Lim, "On Realizing a Broadband Kernel for Multimedia Networks", *3rd COST 237 Workshop on Multimedia Telecommunications and Application*s, Barcelona, Spain, November 25-27, 1996.

[Clark88] D. Clark, "The design philosophy of DARPA Internet protocols. In*: Proc. of SIGCOMM'88 Symposium*, pp. 106-114, August 1988.

[CPR78] D. D. Clark, K. T. Pogran, and D. P. Reed, “An introduction to local area networks,” Proc. IEEE, vol. 66, no. 11, Nov. 1978, pp. 1497–1516.

[Deca98a] D. Decasper and B. Plattner, “DAN: Distributed Code Cashing for Active Networks”, *Proc. IEEE INFOCOM ‘98*, San Francisco, CA, 29 March–2 April 1998, http://www.arl.wustl.edu/arl/projects/ann/slides/infocom98.pdf.

[Deca98b] Decasper, D., Dittia, Z., Parulkar, G., Plattner, B., “Router Plugins - A Software Architecture for Next Generation Routers”, In *Proceedings of SIGCOMM’98,* September 1998.

[Deca99] D. Decasper, G. Purulkar, S. Choi, J. DeHart, T. Wolf, B. Plattner, “A Scaleable High Performance Active Network Node”, IEEE Network, January/February 1999, pp. 8-19, http://www.arl.wustl.edu/arl/projects/ann/.

[DeFe97] L. Delgrossi, D. Ferrari, “A Virtual Network Service for Integrated-Services Internetworks”, *7th International Workshop on Network and Operating System Support for Digital Audio and Vide*o, St. Louis, May 1997.

[DAN] Washington University, *“*A Scaleable High-Performance Active Node*”,* http://boushi.arl.wustl.edu/arl/projects/ann/.

[DoDAN] US Dept. of Defense, DARPA, *Research Program on Active Networks*, http://www.darpa.mil/ito/ResearchAreas/ActiveNetsList.html.

[Eng95] D. R. Engler et al., “Exokernel: An Operating System Architecture for Application-Level Resource Management”, *Proc. Fifteenth ACM Symposium on Operating System Principles (SOSP’95)*, Copper Mountain, CO, December, 1995, pp. 251-266.

[Fab02] T. Faber, “Experience with Active Congestion Control”, Proc. DARPA Active Networks Conference and Expositions (DANCE), San Francisco, CA, May, 2002.

[Fab98] T. Faber, “ACC: Using Active Networking to Enhance Feedback Congestion Control Mechanisms”, *IEEE Networ*k, 12(3):61–65, May/June 1998.

[Fan98] Z. Fan, A. Mehaoua, „Active Networking: A New Paradigm for Next Generation Networks“, *Second IFIP/IEEE Int. Conference on Management of Multimedia Networks and Services (MMNS’98)*, Versailles, France, Nov. 16-18, 1998.

[Feld98] D. C. Feldmeier, A. J. McAuley, J. M. Smith, D. S. Bakin, W. S. Marcus, and T. M. Raleigh. “Protocol Boosters”, *IEEE J. on Selected Areas in Communications, Special Issue on Protocol Architectures for the 21st Centur*y, 16(3):437–444, April 1998.

[Fern00] A. Fernando, B. Kummerfeld, A. Fekete, M. Hitchens, *“*A New Dynamic Architecture for an Active Network*”, Proc. of IEEE OPENARCH’2000*, Tel-Aviv, March 26-30, 2000, http://comnet.technion.ac.il/infocom2000/.

[FKFH00] A. Fernando, B. Kummerfeld, A. Fekete, M. Hitchens, ”A New Dynamic Architecture for an Active Network“, *Proc. OPENARCH’2000*, Tel-Aviv, April 2000.

[Galt01a] V. Galtier and K. L. Mills, Y. Carlinet, S. F. Bush, A. Kulkarni, “Predicting Resource Demand in Heterogeneous Active Networks”, *Proc. of MILCOM 2001*, McLean, VA, October 28-31.

[Galt01b] V. Galtier, K. Mills, Y. Carlinet, S. Bush, A. Kulkarni, "Predicting and Controlling Resource Usage in a Heterogeneous Active Network", Proc. of the Third International Workshopon Active Middleware Services, August 2001, pp. 35-44.

[Galt02] V. Galtier, K. Mills, and Y. Carlinet, "Modeling CPU Demand in Heterogeneous Active Networks", Proc. of the DARPA Active Networks Conference and Exposition (DANCE'2002), June 2002.

[Gold01] R. Gold, "Self-Organizing Route Aggregation for Active Ad-Hoc Networks", *Proc. IEEE Openarch 2001*, Short Paper Session "Ghosts of the Net!", pp. 5-9, Anchorage, Alaska, April 28, 2001.

[Fry99] M. Fry et al., "Application Level Active Networking", *Computer Networks and ISDN Systems*, May 1999.

[GoYe97] G. Goldszmidt and Y. Yemini, "Delegated Agents for Network Management", *IEEE Commun. Mag.*, March 1998, vol. 36, no. 3, pp. 66–70.

[GMcG95] J. Gosling, H. McGilton, *"*The Java Language Environment: A WhitePaper*", Sun Microsystems*, Mountain View, CA, 1995, http://www.sun.com.

[Green98] M. S. Greenberg, Jennifer C. Byington, and D. G. Harper, "Mobile Agents and Security", *IEEE Commun. Mag*., July 1998, vol. 36, no. 7.

[Gutt97] J. Guttag, *"*From Internet to Active Net*",* slides report, *MIT* Laboratory for Computer Science, Cambridge, MA 02139- 4307, guttag@lcs.mit.edu.

[GuJi97] C. A. Gunter and T. Jim, "Design of an Application-Level Security Infrastructure", *DIMACS Workshop on Design and Formal Verification of Security Protocols*, Sept. 3–5, 1997.

[GNS98] C. A. Gunter, S. M. Nettles, and J. M. Smith, "The SwitchWare Active Network Architecture", *IEEE Networ*k, special issue on Active and Programmable Networks, May/June 1998, vol. 12, no. 3.

[GSTF00] J. Gao, P. Steenkiste, E. Takahashi, A. Fisher, "A Programmable Router Architecture Supporting Control Plane Extensibility", *IEEE Communications Mag.*, March 2000, pp. 152-159.

[Hart96] J. J. Hartman *et a*l., "Liquid Software: A new Paradigm for Networked Systems", *Technical Report TR96-11*, Department of Computer Science, University of Arizona, 1996.

[Hart99] J. J. Hartman et al., "Joust: A Platform for Liquid Software", *IEEE Computer,* Apr. 1999, pp. 50-56.

[Hicks98] M. Hicks, P. Kakkar, J. T. Moore, c. A. Gunter, S. Nettles, *"*PLAN: A Packet Language for Active Networks", *Proc. of the Third ACM SIGPLAN Intnl. Conf. On Functional Programming Languages*, Sept. 1998, http://www.cis.upenn.edu/switchware/papers/plan.ps.

[Hicks99] M. Hicks, "Dynamic Software Updating*", Technical Report*, Department of Computer and Information Science, University of Pennsylvania, Oct. 1999, Thesis Proposal, http://www.cis.upenn.edu/~mwh/proposal.ps.

[Hicks01a] M. Hicks, J. T. Moore, S. Nettles, "Practical Programmable Packets", Proc. of INFOCOM'01, April 2001.

[Hicks01b] M. Hicks, J. T. Moore, S. Nettles, "Compiing PLAN to SNAP", Proc. of Int. Working Conference on Active Networks IWAN'01, September/October 2001.

[IDL] ISO/IEC JTC1/SC21 WG7 N14750, "ISO/OMG IDL -- Interface Definition Language".

[Jaeg99] R. Jaeger 1 , R. Duncan, F. Travostino, T. Lavian, J. Hollingsworth, "An Active Network Services Architecture for Routers with Silicon-Based Forwarding Engines", http://www.cs.umd.edu/~hollings/papers/lanman99.pdf .

[JeWa] A. Jeffrey and I. Wakeman, "A Survey on Semantic Techniques for Active Networks", http://www.cogs.susx.ac.uk/projects/safetynet .

[Jon99a] M. Jones, L. Scharf, J. Scott, C. Twaddle, M. Yaconis, K. Yao, P. Athanas, and B. Schott, "Implementing an API for Distributed Adaptive Computing Systems," Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, CA, April 1999, http://www.east.isi.edu/SLAAC/papers/jones_api.pdf .

[Jon99b] M. T. Jones, J. Hess, D. Lee, S. Harper, and P. Athanas, "Implementation of a Prototype Reconfigurable Router", Proc. of IEEE Symposium on Field-Programmalbe Custom Computing Machines, Napa, CA, April, 1999.

[JTAPI] http://java.sun.com/products/jtapi/index.html.

[Karm98] A. Karmouch, "Mobile Software Agents for Telecommunications", Guest Editorial for *IEEE Commun*. Mag., July 1998 vol. 36 no. 7.

[Kas00] S. K. Kasera et al., "Scalable Fair Reliable Multicast Using Active Services," IEEE Network, Jan./Feb. 2000, pp. 48-56.

[Kell00] R. Keller et al., *"*An Active Router for Multicast Video Distribution*"*, *Proc. of IEEE OPENARCH'2000,* Tel-Aviv, March 26-30, 2000, http://comnet.technion.ac.il/infocom2000/.

[Kulk96] A. B. Kulkarni *et a*l., "An Active Network Architecture for ATM WANs", *Proc. Mobile Multimedia Commun. Conf*., Princeton, NJ, Sept. 25–27, 1996.

[Kulk98] A. Kulkarni et al., "Implementation of a Prototype Active Network," *Proc. of IEEE Conf. on Open Architectures and Network Programming (OPENARCH'98)*, San Francisco, CA, Apr. 1998.

[KuMi99] A. B. Kulkarni, G. J. Mindem, "Active Networking Services for Wired/Wireless Networks*", Proc. INFOCOM'99*, pp. 1116-1123, New York, USA, 1999, http://www.ieee-infocom.org/1999/program.html.

[KuMi00] A. B. Kulkarni and G. J. Minden, "Composing Protocol Frameworks for Active Wireless Networks", *IEEE Comm. Mag.,* March 2000, Vol. 38, No. 3, pp. 131-137.

[Laz97] A. A. Lazar, "Programming Telecommunication Networks", *IEEE Network,* vol.11, no.5, September/October 1997.

[Lee99] D. C. Lee, S. J. Harper, P. M. Athanas, S. F. Midkiff, "A Stream-based Reconfigurable Router Prototype",IEEE International Conference on Communications, Vancouver, B.C., Jun 1999.

[LC97] A. A. Lazar, A.T Campbell, "Spawning Network Architectures", *Technical Report, Center for Telecommunications Researc*h, Columbia University, 1997.

[LC98] R.-F. Liao, A. T. Campbell, , "On Programmable Universal Mobile Channels in a Cellular Internet", *4th ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98)* , Dallas, October, 1998.

[LGT98] L. H. Lehman, Stephen J. Garland, and D. L. Tennenhouse, "Active Reliable Multicast", *Proc. IEEE INFOCOM '98,* San Francisco, CA, 29 March–2 April 1998.

[LWG98] U. Legedza, D. J. Wetherall, J. Guttag, *"*Improving the Performance of Distributed Applications Using Active Networks*"*, *Proc. IEEE Infocom'98*, San Francisco, CA, 29 March – 2 April, 1998

[M0] C. Tschudin, *"The M0 System"*, http://cui.unige.ch/tios/msgr/m0/.

[Marc98] W. S. Marcus, I. Hadzic, A.J. McAuley, J.M. Smith, "Protocol Boosters: Applying Programmability to Network Infrastructures", IEEE Communications Magazine, vol. 36, no. 10, October 1998, pp. 79-83.

[Merw97a] J. E. Van der Merwe, I. M. Leslie, "Switchlets and Dynamic Virtual ATM Networks", *Proc Integrated Network Management* V, May 1997.

[Merw97b] J. E. Van der Merwe, S. Rooney, I. M. Leslie, S. A. Crosby, "The Tempest - A Practical Framework for Network Programmability", *IEEE Networ*k, November 1997. Heterogeneous Environments", *IEEE Communications Magazin*e, Vol. 14, No. 2, February, 1997.

[Metz99] B. Metzler, T. Harbaum, R. Wittmann, M. Zitterbart, "AMnet: Heterogeneous Multicast Services based on Active Networking", Proc. of the 2nd Workshop on Open Architectures and Network Programming (OPENARCH99), New York, NY, USA, March, 1999.

[Mosb96] D. Mosberger et al., "Making Paths Explicit in the Scout Operating System," Proc. *OSDI ' 9*6, Oct. 1996, pp. 153-168.

[Mware] Mobiware Toolkit v1.0 source code distribution, http://www.comet.columbia.edu/mobiware.

[MMN01] J. T. Moore, J. K. Moore, S. Nettles, "Scalable Distributed Management with Lightweight Active Packets", Technical Report MS-CIS-01-26, Dept. of Computer and Information Science, University of Pennsylvania, September 2001.

[Necu97] G. C. Necula, "Proof-Carrying Code", Proc. 24th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, *ACM Press*, 1997.

[NetScript] *"*NetScript: A Language and Environment for Programmable Networks*",* http://www.cs.columbia.edu/dcc/netscript/body_index.html.

[NOSIS] L. Peterson, "NodeOS Interface Specification", Technical Report, AN NodeOS Working Group, July 23, 1999, http://www.cs.princeton.edu/nsg/papers/nodeos.ps*;* November 30, 2001, http://www.ecs.umass.edu/ece/wolf/courses/ECE697J/papers/AN_node_OS.pdf.

[Nygr99] E. Nygren et al., *"*PAN: A High-Performance Active Network Node Supporting Multiple Code Systems*", Proc. 2nd Conf. On Open Architectures and Network Programming (OPENARCH'99),* pp. 78-89, IEEE, New York, NY, March 1999.

[OPENS] Open Signaling Working Group, http://comet.columbia.edu/opensig/.

[OPS97] *"OPENSIG Fall'97 Workshop"*, October 1997, New York, NY, http://comet.ctr.columbia.edu/opensig/activities/fall.97.html.

[PBYY00] D. Putzolu, S. Bakshi, S. Yadav, R. Yavatkar, "The Phoenix Framework: A Practical Architecture for Programmable Networks", *IEEE Communications* Mag., March 2000, pp. 160-165.

[PhKa98] V. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview*"*, *IEEE Commun. Mag*., July 1998, vol. 36, no. 7.

[PLANet] M.Hicks *et al*., "PLANet: An Active Network Testbed, http://www.cis.upenn.edu/switchware/papers/planet.ps .

[PLANet99] M. Hicks, J. T. Moore, D. S. Alexander, C. A. Gunter, S. Nettles, *"*PLANet: An Active Internetwork", *Proc. of the 18th IEEE Computer and Communication Society Conference INFOCOM'99*, March 1999, pp. 1124-1133.

[PMM93] C. Partridge, T. Mendez, and W. Milliken, "Host Anycasting Service", *IETF RFC 1546*, Nov. 1993.

[PPV98] G. Parulkar, C. Papadopoulos, G. Varghese, *"*An Error Control Scheme for Large-Scale Multicast Applications*"*, *Proc. IEEE Infocom'98*, 1998.

[Pso99] K. Psounis, "Active Networks: Applications, Security, Safety and Architecture", *IEEE Communications Surveys*, Vol. 2, No. 1, 1999.

[P1520] IEEE Standards Working Group P1520, "Application Programming Interfaces for Networks", http://www.ieee-pin.org, and http://stdsbbs.ieee.org/groups/index.html.

[Q.1214] ITU Recommendation Q.1214, "Distributed Functional Plane for Intelligent Network CS-1 (10/95)".

[RaSh00] D. Raz, Y. Shavitt, "Active Networks for Efficient Distributed Network Management", *IEEE Comm. Mag*., Vol. 38, No. 3, March 2000.

[RaSh01] D. Raz, Y. Shavitt, "New Models and Algorithms for Programmable Networks", Lucent Technologies Technical Memorandum 10009674-991116-02, Proc. OpenArch 2001, April 2001, Anchorage, AK, USA; http://www.cs.bell-labs.com/who/ABLE/index.html.

[RBP] G. Denker, Specifying a Reliable Broadcasting Protocol in Maude, http://www.csl.sri.com/~denker/publ/DGM+99_abs.html.

[SAAB98] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan, "Detour: a Case for Informed Internet Routing and Transport", *Technical Report UW-CSE-98-10-05,* University of Washington, 1998.

[Saw01] V. Sawant, "Exposing Low Level Function for Application Efficiency", Ph.D. Dissertation, Univ. of North Carolina at Chapel Hill, NC, April 2001.

[SAMS98] J. Smith, D. S. Alexander, W. Marcus, M. Segal, and W. D. Sincoskie. Towards an Active Internet, 1998.

[Scott98] D. Scott *et a*l., "A Secure Active Network Environment Architecture", *IEEE Networ*k, special issue on Active and Programmable Networks, May/June 1998, vol. 12, no. 3.

[ScottTR] D. Scott *et a*l., "Performance Implications of Securing Active Networks", *Technical Report MS-CIS-98-02*.

[ShBo99] A. B. Shah, V. G. Bose, "Accelerating Evolution of the Cellular Infrastructure using Software Radios", Vanu Inc., http://www.vanu.com/techpapers.html.

[SCS77] M. D. Schroeder, D. D. Clark, and J. H. Saltzer, "The Multics kernel design project," 6th ACM Symp. Op. Sys. Principles, ACM Op. Sys. Rev., vol. 11, no. 5, Nov. 1977, pp. 43–56.

[SFY98] S. Da Silva, D. Florissi, and Y. Yemini, "NetScript: A Language-Based Approach to Active Networks", *Technical Report, Computer Science Dept., Columbia University,* January 27, 1998.

[SHB00] R. Sivankumar, S.-W. Han, V. Bharghavan, "A Scaleable Architecture for Active Networks", *Proc. of IEEE OPENARCH '2000*, Tel-Aviv, March 26-30, 2000, http://comnet.technion.ac.il/infocom2000/.

[SmartP99] B. Schwartz W. Zhou, A. Jackson, W. T. Strayer, D. Rockwell, C. Partridge, "Smart Packets for Active Networks", *Proc. of the 1999 2nd Conf. on Open Architectures and Network Programming (OPENARCH'99),* March 1999.

[Smith98] J. M. Smith et al., "Activating Networks: A Progress Report," *IEEE Computer*, Apr. 1999, pp. 32-41.

[SpMe99] O. Spaniol, J. Meggers, "Active Network Nodes for Adaptive Multimedia Communication", *Proc. of SMARTNET'99*, Nov. 1999.

[SRBW01] M. Schläger, B.Rathke, S. Bodenstein, A. Wolisz; “Advocating a Remote Socket Architecture for Internet Access using Wireless LANs”, accepted for publication in Mobile Networks and Applications, Balzer Science Publishers, Special Issue on Wireless Internet and Intranet Access, 2001.

[SRC81] J. H. Saltzer, D. P. Reed, D. D. Clark, “End-to-end arguments in system design,” Proc. 2d Int. Conf. Dist. Comp. Sys., April, 1981; also in: ACM Trans. Comp. Sys., vol. 2, no. 4, 1984, pp. 277-88.

[Tschu97] C. F. Tschudin, “Active Network Overlay Network (ANON)”, *RFC Draft*, December 1997.

[Tschu99a] C. Tschudin, “Active Ad-Hoc Networking”, *Teleinformatics Seminar*, Kungl Tekniska Högskolan Sweden, Nov 22, 1999.

[Tschu99b] C. F. Tschudin, “A Self-Deploying Election Service for Active Networks”, pp. 183-195, *Proc. of COORDINATION '99*, Coordination Languages and Models, Third Int. Conference, Amsterdam, The Netherlands, April 26-28, 1999, Lecture Notes in Computer Science, Vol. 1594, Springer, 1999, ISBN 3-540-65836-X.

[Tschu99c] C. F. Tschudin, “Apoptosis - the Programmed Death of Distributed Services”, pp. 253-260, *Secure Internet Programming, Security Issues for Mobile and Distributed Objects*, J. Bosch, J. Vitek, C. D. Jensen (Eds.), Lecture Notes in Computer Science, Vol. 1603, Springer, 1999, ISBN 3-540-66130-1

[Tenn96a] D. L. Tennenhouse et al, “From Internet to ActiveNet”, *RFC Draft*, Jan. 1996.

[Tenn96b] D. L. Tennenhouse et al., “The Active IP Option,” *Proc. of the 7th ACM SIGOPS European Workshop*, Sep. 1996.

[Tenn96c] D. Tennenhouse et al., *“*Virtual Infrastructure: Putting Information Infrastructure on the Technology Curve*”, Comp. Networks and ISDN Sys*., vol. 26, no. 13, Oct. 1996.

[Tenn97] D. Tennenhouse et al*., “*A Survey of Active Network Research*”, IEEE Comm. Mag.,* Vol. 35, No. 1, Jan. 1997, pp. 80-86.

[Tenn99] D. Tennenhouse, “What’s New at DARPA/ITO ?”, July 28th, 1999, http://www.cra.org/Activities/snowbird/slides/tennenhouse/index.html.

[TeWe96] D. Tennenhouse and D. Wetherall, *"*Towards an Active Network Architecture*”*, *Proc. of Multimedia Computing and Networking (MMCN‘96)*, 1996, San Jose, CA; also in: Computer Communications Review, vol. 26, no. 2, Apr. 1996, also at: http://www.tns.lcs.mit.edu/publications/mmcn96/mmcn96.html.

[ToHo98] J. Touch, S. Hotz, "The X-Bone", Third *Global Internet Mini-Conference in conjunction with Globecom’98* Sydney, Australia, November 1998.

[Turn99] J. Turner et. al, "Design of a Flexible Open Platform for High Performance Active Networks", Proc. of the 37th Annual Allerton Conference on Communication, Control and Computing, Monticello, IL, September, September 1999

[Vin97] S. Vinoski, “CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments”, *IEEE Communications Magazin*e, Vol. 14, No. 2, February, 1997.

[VPK97] A. Vassila, G. Pavlou, and G. Knight, “Active Objects in TMN”, in: *Integrated Network Management*, V. A. Lazar, R. Saracco, R. Stadler (Eds.), pp. 139–150, Chapman & Hall, 1997.

[Wake99] I. Wakeman *et al*., “Designing a Programming Language for Active Networks”, http://www.cogs.susx.ac.uk/projects/safetynet, Hipparch special issue of *Network and ISDN System*s, Jan. 1999.

[Weth99a] D. Wetherall et al., "ANTS: Network Services Without the Red Tape", *IEEE Computer*, Apr.1999, pp. 42-48.

[Weth99b] D. Wetherall, "Active Network Vision and Reality: Lessons from a Capsule-Based System", *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP'99),* Kiawah Island, SC, Dec. 1999, also in: Operating Systems Review 34(5): 64-79, Dec. 1999.

[WeLH94] D. Wetherall, C. Lindblad, and H. Hough, "Active Pages: Intelligent Nodes on the World Wide Web", *Proc. 1994 World Wide Web Conf*., Geneva, Switzerland, May 1994.

[WeTe96] D. Wetherall and D. Tennenhouse, "The ACTIVE IP Option", *Proc. 7th ACM SIGOPS Euro. Workshop*., Connemara, Ireland, 1996.

[WoTu99] T. Wolf, J. S. Turner, "Design Issues for High Performance Active Routers", Tech. Report WUCS-99-19, Dept. of Computer Science, Washington University, June 11, 1999, http://ccrc.wustl.edu/~wolf/app/JSAC.pdf.

[Wolf99] T. Wolf, "A Proposal for a High-Performance Active Hardware Architecture, Tech. Report WUCS-99-08, Dept. of Computer Science, Washington University, February 15, 1999, http://ccrc.wustl.edu/~wolf/papers/wucs-99-08.pdf.

[WGT98] D. J. Wetherall, J. Guttag, D. L. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", *Proc. of IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[WLG98] D. Wetheral, U. Legedza, J. Guttag, "Introducing New *Internet Services: Why and How to Deploy Them",* IEEE Network Mag., Special Issue on Active and Programmable Networks, July 1998.

[Xbind] xbind code http://comet.columbia.edu/xbind, also: Xbind Inc., http://www.xbind.com.

[YeSi96] Y. Yemini and S. daSilva, "Towards Programmable Networks", *Proc. IFIP/IEEE Intnl. Workshop on Distributed Systems*: Operations and Management, L'Aquila, Italy, Oct. 1996.

[YGY91] Y. Yemini, G. Goldszmidt, and S. Yemini, "Network Management by Delegation", *Integrated Network Management II*, I. Krishnan and W. Zimmer, Eds., pp. 95–107, North-Holland, 1991.

[ZaFo83] J. Zander and R. Forchheimer, "SOFTNET An Approach to High Level Packet Communication", Proc. AMRAD Conf. 1983, San Francisco, CA, 1983.

[Zegura96] E. W. Zegura, "Active Networking: An End to the IP / ATM Debate ?", http://emperor.arl.wustl.edu/arl/workshops/atmip/session4/session4.html, Nov. 13th , 1996, slides for the Washington University Workshop in Integration of IP and ATM.

[Zhang98] L. Zhang, S. Michel, K. Nguyen, A. Rosenstein, S. Floyd, V. Jacobson, "Adaptive Web Caching: Towards a New Global Caching Architecture", *Proc. 3d Intnl. WWW Caching Workshop*, 1998.

FORMAL METHODS

[AbLa92] Abadi, M., Lamport, L., "An Old-Fashioned Recipe for Real Time", *Research Report*, Nr. 91, DEC, Systems Research Centre, 1992.

[Card98] L. Cardeli, A. D. Gordon, "Mobile Ambients", Foundations of Software Science and Computational Structures, LNCS, Vol. 1378, Springer-Verlag, 1998, pp. 140-155.

[ChMi88] K. M. Chandy, J. Misra, *"*Parallel Program Design: A Foundation*",* Addison-Wesley, 1988.

[DAW98] D. Dieckman, P. Alexander, P. Wilsey, "ActiveSPEC: A Framework for the Specification and Verifivation of Active Network Services and Security Policies, *Proc. of FMSP"98*, 1998, http://www.ececs.uc.edu/~kbse/projects/activespec/.

[DrPe86] J. Drapkin, D. Perlis, *"*Step-logics: An Alternative Approach to Limited Reasoning*"*, *Proc. of the European Conf. On Artificial Intelligence*, 1986.

[Eng92] Engberg, U., Groenning P., and Lamport, L., "Mechanical Verification of Concurrent Systems with TLA", In: *Logics of Programs*, LNCS, Springer-Verlag, June, 1992.

[Four96] C. Fournet, G. Gonthier, J. Levy, L. Marnaget, D. Remy, "A Calculus of Mobile Agents", *Proc. of CONCUR'96*, LNCS, Vol. 1119, Springer-Verlag, 1996.

[Harry96] A. Harry, *"*Formal Methods: Fact File, VDM and Z*"*, John Wiley & Sons, 1996, ISBN 0-471-94006-2.

[HeMa96] C. Heitmeyer, D. Mandrioli (Eds.), *"*Formal Methods for Real-Time Computing*",* John Wiley & Sons, 1996, ISBN 0-471-95835-2.

[Kamp68] A.W Kamp, "Tense Logic and the Theory of Order", Ph.D. thesis, UCLA, 1968.

[KAD00] C. Kong, P. Alexander, D. Dieckmann, "Formal Modeling of Active Network Nodes Using PVS", Proc. of FMSP'00, ACM, Portland, OR, USA.

[Lamp91] L. Lamport, *"*The Temporal Logic of Actions*",* *SRC Res. Report 79*, DEC, Sys. Res. Center, Dec. 1991.

[Lamp93] L. Lamport, "Hybrid Systems in TLA$^+$", In: *Hybrid Systems* (Proc. of a Workshop on Theory of Hybrid Systems, Lyngby, Denmark, 19-21 Oct., 1992), H. Rischel, A. Ravn (Eds.), LNCS, 1993.

[Lamp94] L. Lamport, *"*The Temporal Logic of Actions*",* *ACM Toplas*, 16, 3, pp. 872-923, May 1994; also in: http://www.research.digital.com/SRC/personal/lamport/tla/.

[Lamp01] L. Lamport, "Specifying Systems", Draft, August 2001, http://www.research.compaq.com/SRC/personal/lamport/tla/book.html.

[Lynch97] N. Lynch, *"Mathematical Modelling/Specification/Verification/Performance Analysis/Fault-Tolerance Analysis for Network Services"*, ARPA Active Nets Workshop, March 1997.

[LSP82] L. Lamport, R. Shostak, M. Pease, "The Byzantine Generals Problem", *ACM Trans. On Programming Languages and Systems*, Vol. 4, No. 3, July 1982, pp. 382-401.

[MaPn92] Z. Manna and A. Pnueli, "The Temporal Logic of Reactive and Concurrent Systems", Springer-Verlag, 1992, ISBN 0-387-97664-7.

[Mark98] F. Markopoulou, "The Internal Description of a Causal Set: What the Universe Looks Like from the Inside", 18. November, 1999, http://xxx.lanl.gov/gr-qc/9811053.

[Mark99] F. Markopoulou, "Quantum Causal Histories", *Class. Quan. Grav.*, Vol. 17, 28. January, 2000, http://xxx.lanl.gov/hep-th/9904009.

[MAUDE] C. Talcott, "MAUDE: A Wide-Spectrum Formal Language for Secure Active Networks", http://www-formal.stanford.edu/clt/ArpaActive/index.html.

[Mut99] I. Mutabanna. ""Exploring Formal Models for Active Network Security", M*aster's Thesis*, The University of Cincinnati, 1999.

[MTH90] R. Milner, M. Tofte, R. Harper, "The Definition of Standard ML", MIT Press, 1990.

[NASAL] NASA Langley's Formal Methods Program, http://shemesh.larc.nasa.gov/fm.

[Pnu77] Pnueli, A., "The Temporal Logic of Programs", *Proc. 18'th IEEE Symp. Found. of Computer*. Science, pp., 46-57, 1977.

[PVS] J. Crow, J. Rushby, N. Shankar, and M. Srivas, "A Tutorial Introduction to PVS", presented at WIFT'95, SRI Intnl., Menlo Park, CA, June 1995.

[RRH93] Ravn, A. P., Rischel, H., Hansen, K. M., "Specifying and Verifying Requirements of Real-Time Systems", *IEEE Trans. on Software Engineering*, vol. 19 (1), January 1993, pp. 41-55.

[RPM00] G.-C. Roman, G. Picco, A. Murphy, "Software Engineering for Mobility: A Roadmap", In: *Future of Software Engineering*, A. Finkelstein (Ed.), 22^{d} Int. Conf. on Soft. Eng., ACM Press, 2000, pp. 241-258.

[Sat00] I. Satoh, "A Formalism for Hierarchical Mobile Agents", *Proc. of Symp. on Soft. Eng. For Parallel and Distributed Systems* (PDSE'2000), June 2000, IEEE Computer Soc., pp. 165-172.

[Sim88] P. L. Simeonov, "An Algorithm for the Analysis of Schematic Drawings Based on the Topological Cell Complexes", Internal Report ZKI-AdW-BV-043. Zentralinstitut für Kybernetik und Informationsprozesse, Akademie der Wissenschaften, Berlin, August 1988.

[Sim89] P. L. Simeonov, M. Plessow, "Netlike Schematics and Their Structural Description", *Proc. of the VII Workshop on Informatics in Industrial Automation*. Zentralinstitut für Kybernetik und Informationsprozesse, Akademie der Wissenschaften, Berlin, 31 Oct. - 4. Nov 1989, 144-161.

[Sim90] P. L. Simeonov, "On Netlike Schematics, Their Description and Design", *Thesis Proposal*, Internal Report ZKI-AdW-SA-CAS-006. Zentralinstitut für Kybernetik und Informationsprozesse, Akademie der Wissenschaften, Berlin, January 1990.

[SiMi94] P. L. Simeonov, I. Miloucheva, "Prototype Performance Evaluation of Multimedia Service Components", *Proc. of the First Workshop on Architecture and Implementation of High-Performance Communication Systems*, Karlsruhe, January 17-18th, 1994.

[Sim94a] P. L. Simeonov, "A Method for QoS Verification by Measurement of the JVTOS Video Transfer", *Tech. Report R2088/TUB/PRZ/DN/P/041/a1*, Project TOPIC (RACE 2088). TU Berlin, April 1994. (also as paper of the *Montreal Workshop on Multimedia Applications and Quality of Service Verification*, Montreal, Canada, May 31 - June 3, 1994.).

[Sim94b] P. L. Simeonov, I. Miloucheva, K. Rebensburg, K. J. Turner, A. J. M. Donaldson, *Prototype Performance Evaluation of Multimedia Service Components*. Proc. of the ICCCN'94 (Third International Conference on Computer Communications and Networks, IEEE), San Francisco, California, USA, September 11 - 14, 1994, 254-261.

[Turn93] K. J. Turner, "Using Formal Description Techniques – An Introduction to Estelle, LOTOS and SDL", John Wiley and Sons Ltd, 1993, ISBN 0-471-93455-0, also as http://www.cs.stir.ac.uk/~kjt/research/using-fdts/.

[Turn01a] K. J. Turner, "Modular Feature Specification", in: M. Kim (Ed.), *Proc. of Formal Methods for Protocol Engineering and Distributed Systems FORTE XIV/PSTV XXI*, February 2001.

[Turn01b] K. J. Turner, J. He, “Formally-Based Design Evaluation”, in: T. Margaria, T. F. Melham (Eds.), *Proc. of 11th Conference on Correct Hardware Design and Verification Methods CHARME’2001*, LNCS 2144, pp. 104-109, Springer Verlag, Berlin, Germany, September 2001.

[MOBILE] NETWORKING

[Add01] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, “Implementing a Sentient Computing System”, *IEEE Computer*, pp. 50-56, August, 2001.

[Blu95] S. M. Blust, “Software Defined Radio – Industry Request for Information”, Tech, Report, Bell South Cellular, December, 1995.

[BoSh98] Bose, V. G ., Shah, A. B. , "Software Radios for Wireless Networking," *Proc. of Infocomm '98*, San Francisco, April, 1998.

[Bose99] V. G. Bose, *“The Impact of Softare Radios on Wireless Networking”*, Mobile Computing and Communications Review, Vol. 3, No. 1, January 1999.

[BHM01] V. G. Bose, R. Hu, R. Morris, "Dynamic Physical Layers for Wireless Networks Using Software Radio", International Conference on Acoustics, Speech, and Signal Processing (ICASSP’2001), Salt Lake City, UT May 2001. http://www.vanu.com/techpapers.html.

[BIWG99] V. G. Bose, M. Ismert, M. Welborn, J. Guttag, *“Virtual Radios”,* IEEE/JSAG, Special Issue on Software Radios, April 1999.

[BuHu01] L. Buttyan, J.-P. Hubaux, “Nuglets: A Virtual Currency to Stimulate Cooperation in Self-Organized Mobile Ad Hoc Networks”, *Tech. Report*, No. DSC/2001/001, Swiss Federal Institute of Technology (EPFL), Lausanne, Jan. 2001, http://ww.terminodes.org/publications.html.

[ETSI3G] ETSI TS 25.401, “*Universal Mobile Telecommunications System (UMTS)”,* V3.1.0 Release 1999, January 2000.

[GeTs95] M. Gerla, J. T. C. Tsai, “Multicluster, Mobile, Multimedia Radio Network“, *Wireless Networks*, Vol. 1, pp. 255-265, 1995.

[IEEEC01a] IEEE Communications Magazine, *QoS and Resource Allocation in the 3rd Generation Wireless Networks*, February 2001, Vol. 39, No. 2.

[IEEEC01b] IEEE Communications Magazine, Life After Third-Generation Mobile Communications, August 2001, Vol. 39, No. 8.

[IEEEN00] IEEE Network, Next Generation Wireless Broadband Networks, Vol. 14, No. 5, Sept./Oct., 2000.

[Iwa99] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, T.-W. Chen, “Scalable Routing Strategies for Ad-hoc Wireless Networks”, *IEEE Jour. On Selected Areas in Communications* (Special Issue on Ad-Hoc Networks), pp. 1536-1540, August, 1999.

[JuTo87] J. Jubin, J. D. Tornow, "The DARPA Packet Radio Network Protocols", *Proc. of the IEEE*, pp. 21-32, 75 (1), 1987.

[Katz95] Katz., R. H., „Adaptation of Mobility in Wireless Information Systems“, IEEE Personal Communications Magazine, Vol. 1, No. 1, 1995, pp. 6-17.

[Klein95] Leonard Kleinrock, “Nomadic Computing – an Opportunity”, Res. Report CCR-4/95, UCLA, 1995; also as Keynote Address, Proc. of Int. Conf. on Mobile Computing and Networking, Berkeley, CA, ACM Press, 1995; and in IEEE Personal Communications Magazine., April, 1995.

[KGBK78] S. Kahn, J. Gronemeyer, J. Burchfiel, R. Kunzelman, "Advances in Packet Radio Technology", *Proc. of the IEEE*, pp. 1468-1496, 66 (11), 1978.

[LiGe97] C. R. Lin, M. Gerla, “MACA/PR: An Asynchronous Multimedia Multihop Wireless Network“, *Proc. of INFOCOM’97*, March, 1997.

[LMC97] S. S. Lumetta, A. M. Mainwaring, D. E. Culler, “MultiProtocol Active Messages on a Cluster of SMP's”, *Proc. of Supercomputing'97*, Sao Jose, USA, Nov. 1997.

[LNT87] B. Leiner, D. Nielson, F. Tobagi, "Issues in Packet Radio Network Design", Proc. of the IEEE, pp. 6-20, 75 (1), 1987.

[MPI] The MPI Forum, “MPI: A Message Passing Interface”, ACM 0-8186-4340-4/93/0011.

[PeBh94] C. E. Perkins, P. Bhagwat, "Highly Dynamic Destination-sequenced Distance Vector Routing (DSDV) for Mobile Computers", *Proc. ACM SIGCOMM*, London, UK, pp. 234-244, 1994.

[PGH95] Padgett, J. E., Gunther, C. G., Hattori, T., “Overview of Wireless Personal Communications”, IEEE Comm. Mag., Jan. 1995, Vol. 33, No. 1, pp. 28-41.

[RoToh99] E. M. Royer, C.-K. Toh, “A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks”, *IEEE Personal Communications*, pp 46-55, April, 1999.

[Shar96] J. Sharony, “An Architecture for Mobile Radio Networks with Dynamically Changing Topology using Virtual Subnets”, *Mobile Networks and Applications*, Vol. 1, pp. 75-86, J. C. Baltzer AG Science Publishers, 1996.

[ShBo99] Shah, A. B., Bose, V. G., "Accelerating Evolution of the Cellular Infrastructure using Software Radios", *Massachusetts Telecommunications Council 4th Annual R&D Conference*, June 1999, http://www.vanu.com.

[ShWe87] N. Shacham, J. Wescott, "Future Directions in Packet Radio Architectures and Protocols", *Proc. of the IEEE*, pp. 83-99, 75 (1), 1987.

[SLG00] W. Su, S.-J. Lee, M. Gerla, “Mobility Prediction and Routing in Ad-Hoc Wireless Networks”, *Tech. Report*, Wireless Adaptive Mobility Laboratory, UCLA.

[TeBo99] D. L. Tennenhouse, V. G. Bose, "The SpectrumWare Approach to Wireless Signal Processing", *Wireless Networks*, Volume 2, No.1, 1996.

[TuTe97] Turletti, T., Tennenhouse, D. L., “Estimating the Computational requirements of a Software GSM Base Station”, Proc. ICC’97, Montreal, June 8-12, 1997, pp. 169-175.

[Vit97] Andrew J. Viterbi, “Four Laws of Nature and Society and their Impact on Digital Wireless Communications”, Keynote Speech, International Conference on Computer and Communication (ICCC’97), Cannes, France, November 19th, 1997.

RECONFIGURABLE COMPUTING

[ACS] DARPA Information Technology Office, *"Adaptive Computing Systems Program"*, http://www.darpa.mil/ito/acs.

[AtAb95] P. Athanas, A. Abbott, "Real-Time Image Processing on a Custom Computing Platform", IEEE Computer, pp. 16-24, Feb. 1995.

[BeGo00] E. A. Bezerra, M. P. Gough, "A Guide to Migrating from microprocessor to FPGA Coping with the Support Tool Limitations", Microprocessors and Microsystems 23 (2000), October 2000, 561-572.

[BoPr00] K. Bondalapati, V. K. Prasanna, "Reconfigurable Computing: Architectures, Models and Algorithms", Tech. Report, USC, Dept. of EE, http://maarc.usc.edu/.

[BiAt97] R. Bittner and P. Athanas, *"Wormhole Run-time Reconfiguration",* ACM/SIGDA Int. Symposium on FPGAs, Monterey, CA, pp.79-85, Feb. 1997.

[Breb97] G. Brebner, "A Virtual Hardware Operating System for the Xilinx XC6200", *Intnl. Workshop on Field-Programmable Logic and Applications*, September 1997.

[BTCW97] N. Bowden, A. Terfort, J. Carbeck, G. M., Whitesides, "Self-Assembly of Mesoscale Objects into Ordered Two-Dimensional Arrays", *Science*, Vol. 276, pp. 233-235, 1997.

[ChRa92] D. C. Chen, J. M. Rabaey, "*A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed Datapaths",* IEEE Jour. Of Solid State Circuits, Vol. 27, No. 12, Dec. 1992.

[CHW00] T. J. Callahan, J. R. Hauser, J. Wawrzynek, *"The Garp Architecture and C Compiler"*, IEEE Computer, pp. 62-69, April 2000.

[Comp99] K. Compton, "Programming Architectures for Run-Time Reconfigurable Systems", Master's Thesis, Northwestern University, Evanston, IL, USA, December 1999.

[CoHa99] K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", Northwestern University, Dept of ECE, *Tech. Report*, 1999.

[CoHa00] K. Compton, S. Hauck, "An Introduction to Reconfigurable Computing", *IEEE Computer*, April 2000.

[Dand00] A. Dandalis, V. K. Prasanna, J. D. P. Rolim, "An Adaptive Cryptographic Engine for IPSec Architectures", *IEEE Symposium on FPGAs for Custom Computing Machines*, April 2000.

[DBCP97] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, *"Small Forwarding Tables for Fast Routing Lookups",* Computer Communication Review, Vol. 27, No. 4, , pp. 3-14, October 1997.

[DCP95] Z.D. Dittia, J.R. Cox, Jr., and G.M. Parulkar, *"Design of the APIC: A High Performance ATM Host-network Interface Chip", IEEE INFOCOM 1995*, Vol.1, Boston, Massachusetts, pp.179-87, April 2-6, 1995.

[DeHon00] A. DeHon, *"The Density Advantage of Configurable Computing"*, IEEE Computer, pp.41-49, April 2000.

[Ebel97] C. Ebeling, D. Cronquist, P. Franklin, J. Secosky, S. G. Berg, *"Mapping Applications to the RaPiD Configurable Architecture"*, Proc. IEEE Conf. on Field-Programmable Custom Computing Machines, pp. 106-115, Apr. 1997.

[ECF97] C. Ebeling, D. Cronquist, P. Franklin, *"Configurable Computing: The Catalyst for High-Performance Architectures",* Proc. of IEEE Intnl. Conf. On Application-Specific Systems, Architectures and Proc., pp. 364-372, July 1997.

[GePe00] L. Gepert, T. S. Perry, *"Transmeta's Magic Show"*, IEEE Spectrum, pp. 26-33, May 2000.

[Gron96] P. Gronowski et al., *"A 433 MHz 64b Quad-Issue RISC Microprocessor",* Digest of Tech. Papers, Proc. of the 1996 IEEE Intnl. Solid-State Circuits Conf., IEEE CS Press, Los Alamitos, CA, pp. 222-223, 1996.

[GrNe96] P. Graham, B. Nelson, "Generic Algorithms in Hardware and in Software – A Performance Analysis of Workstation and Custom Computing Machine Implementations", *IEEE Symposium on FPGAs for Custom Computing Machines*, April 1996.

[Hauck98] S. Hauck, "The Future of Reconfigurable Systems", *5th Canadian Conference on Field Programmable Devices*, Montreal, June 1998.

[HaKr95] R. Hartenstein, R. Kress, *"A Datapath Synthesis System for the Reconfigurable Datapath Architecture",* Proc. of Asia and South Pacific Design Automation Conf., pp. 479-484, 1995.

[HaAg96] S. Hauck, A. Agarwal, "Software Technologies for Reconfigurable Systems", Northwestern University, Dept. of ECE, Technical Report, 1996.

[HaSk96] J. Haddy and D. Skellern, *"An Asynchronous Transfer Mode (ATM) Stream Demultiplexer and Switch"*, 6th International Workshop on Field-Programmable Logic and Applications, Darmstadt, Germany, pp. 260-269, September 1996.

[HaSm97] I.Hadzic, J.M.Smith, "P4: A platform for FPGA implementation of Protocol Boosters", 7th International Workshop on Field Programmable Logic and Applications (FPL'97), September 1997.

[HaSm98] I. Hadzic and J. M. Smith, "On-the-fly Programmable Hardware for Networks", Proc. of GLOBECOM'98, http://www.cis.upenn.edu/~boosters/globecom98.ps.

[HMPZ99] T. Harbaum, D. Meier, M. Prinke, M. Zitterbart, "Design of a Flexible Coprocessor Unit", Tech Report, Institute of Operating Systems and Computer Networks, TU Braunschweig, 1999.

[HaWa97] J. Hauser, J. Wawrzynek, *"Garp: A MIPS Processor with a Reconfigurable Coprocessor"*, Proc. IEEE Symp. FPGA for Custom Computing Machines, pp. 12-21, Los Alamitos, CA, 1997.

[Hess99] J. Hess, D. Lee, S. Harper, M. Jones, P. Athanas, *"Implementation & Evalulation of a Prototype Reconfigurable Router"*, Proc of FCCM Conference in 1999.

[IBM96] IBM Corp., *"RS/600 Technology Directions"*, Web Document, November 19, 1996, http://www.kinetic.distillery.com/ibmsc96/newibm96.htm.

[Jone99] M. Jones, L. Scharf, J. Scott, C. Twaddle, M. Yaconis, K. Yao, P. Athanas, B. Schott, *"Implementing an API for Distributed Adaptive Computing Systems",* Proc. of the FCCM Conference in 1999.

[Klai00] A. Klaiber, *"The Technology Behind Crusoe Processors",* Jan. 2000, http://www.transmeta.com.

[KVE95] M. Katevenis, P. Vatsolaki, and A. Efthymiou, *"Pipelined Memory Shared Buffer for VLSI Switches", Computer Communication Review*, Vol. 25, No. 4, pp. 39-48, October 1995.

[LCH00] Z. Li, K. Compton, S. Hauck, "Configuration Caching Techniques for FPGA", *IEEE Symposium on FPGAs for Custom Computing Machine*s, 2000.

[LHAM99] D. Lee, S. Harper, P. Athanas, and S. Midkiff, *"A Stream-based Reconfigurable Router Prototype,"* Proc. of the International Conference on Communications (ICC'99), Vancouver, British Columbia, June 1999.

[LeMe95] E. Lemoine, D. Merceron, " Run Time Reconfiguration of FPGA for Scanning Genomic Databases", IEEE Symposium on FPGAs for Custom Computing Machines, 1995.

[LinMcK97] S. Lin and N. McKeown, "A Simulation Study of IP Switching," *Computer Communications Review*, 1997, pp. 15-24.

[LMA98] D. C. Lee, S. Midkiff, P. Athanas, *"Reconfigurable Routers: A New Paradigm for Switching Device Architecture",* April, 1998.

[MaAt99] P. Master, P. Athanas, *"Reconfigurable Computing Offers Options For 3G"* , published as the lead feature in Wireless System Design, January, 1999.

[Man97] W. H. Mangione-Smith et al, "*Seeking Solutions in Configurable Computing,"* IEEE Computer, pp. 38-43, December 1997.

[McH97] J. McHenry, P. Dowd, F. Pellegrino, T. Corrozi, and W. Cocks, *"An FPGA-Based Coprocessor for ATM Firewalls"*, IEEE Symposium on FPGAs for Custom Computing Machines, pp. 30-39, April 1997.

[MiDH96] E. Mirsky, A. DeHon, *"MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources",* Proc. IEEE Symp. on FPGAs for Custom Computing Machines, pp. 157-166, 1996.

[MNK97] N. Morinaga, M. Nikagawa, R. Kohno, *"New Concepts and Technologies for Achieving Highly Reliable and High-Capacity Multimedia Wireless Communications Systems"*, IEEE Communications Mag., pp. 34-40, Jan. 1997.

[MorphoSys] H. Singh et al., *"MorphoSys: An Integrated Reconfigurable Architecture",* http://www.eng.uci.edu/morphosys/docs/nato_paper.html.

[PeZu92] T. Pei and C. Zukowski, *"Putting Routing Tables in Silicon",* IEEE Network, Vol. 6, No. 1, January 1992, pp. 42-50.

[Press99] A. Pressley, Army Research Laboratory, *Key Note,* ICCCN'99, Boston, MA, Oct. 11-13, 1999, apress@avl.mil.

[Rose90] J. Rose et al., "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency", IEEE J. Solid-State Circuits, Oct. 1990, pp. 1217-1225.

[SMP99] R. P. Sidhu, S. Widhwa, A. Mei, V. K. Prasanna, "Genetic Programming using Self-Reconfigurable FPGAs", *Intnl. Workshop on Field-Programmable Logic and Applications*, September 1999.

[SwAt99] S. Swanchara and P. Athanas, *"*A Methodical Approach for Stream-Oriented Configurable Signal Processing*",* 32nd Hawaii International Conference on System Sciences, Wailea, Hawaii, Jan. 1999.

[Tau95] E. Tau, D. Chen, I. Eslick, J. Brown, A. DeHon, *"A First Generation DGPA Implementation"*, Proc. FPD'95, Canadian Workshop on Field-Programmable Devices, May 29-Jun. 1, 1995.

[TeBu00] R. Tessier, W. Burleson, "Reconfigurable Computing for Digital Signal Processing: A Survey", In: "Programmable Digital Signal Processors", Y. Hu (Ed.), Marcel Dekker Inc., 2000.

[Trim92] S. Trimberger, *"Field Programmable Gate Arrays",* Kluver Academic, Norwell, Mass., 1992.

[Vuill96] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard, "Programmable Active Memories: Reconfigurable Systems Come of Age", *IEEE Trans. on VLSI Systems*, 4 (1): 56-69, March 1996.

[Wain97a] E. Waingold et al., *"Baring it All to Software: The Raw Machine"*, IEEE Computer, pp. 86-93, Sept. 1997.

[Wain97b] E. Waingold et al., *"The RAW Benchmark Suite: Computation Structures for General-Purpose Computing"*, Proc. IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'97), pp. 134-143, 1997.

[WVTP97] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, *"Scalable High Speed IP Routing Lookups",* Computer Communication Review, Vol. 27, No. 4, October 1997, pp. 25-36.

[Xilx98] Xilinx Programmable Logic Data Book, Xilinx Inc., San Hose, CA, 1998.

VISIONS

[Bate72] Gregory Bateson, "*Steps to an Ecology of Mind*", Ballantine Books, New York, 1972, (reprint University of Chicago Press, 2000), ISBN 0-226-03905-6.

[Bate79] Gregory Bateson, "*Mind and Nature : A Necessary Unity*", Ballantine Books, New York, 1979.

[Gild96] George Gilder, "*Telecosm*", Simon & Schuster, 1996, ISBN: 0-684-80930-3.

[Hoff98] David Hoffman, *"Visual Intelligence",* W. W. Norton & Co. Inc., 1998, ISBN 0-393-04669-9.

[Jan80] Erich Jantsch, "*The Self-Organizing Universe : Scientific and Human Implications of the Emerging Paradigm of Evolution*", Pergamon Press, New York, 1980, ISBN 0080243126.

[JPL99] Jet Propulsion Laboratory, Space Communication Protocol Standards (SCPS), "Extending the Internet into Space", July 1999, http://bongo.jpl.nasa.gov/scps.

[Kurz99] Ray Kurzweil, "*The Age of Spiritual Machines*", Viking Press, London, 1999.

[Land74] George T. Land, "Grow or Die : The Unifying Principle of Transformation, Delacorte Press, 1974, ISBN 0192860305.

[Luhm86] Niklas Luhmann, "The autopoiesis of social systems." pp. 172-92 in: F. Geyer and J. van der Zouwen (eds.), *Sociocybernetic Paradoxes,* London, Sage, 1986.

[Mat75] H. R. Maturana, "The Organization of the Living: A Theory of the Living Organization", *Int. Journal of Man-Machine Studies*, Vol. 7 (1975), pp. 313-332.

[MaVa80] Humberto R. Maturana, Francisco J. Varela, "*Autopoiesis and Cognition*", D. Reidel Publishing Co., 1980, ISBN 9027710163.

[MaVa84] Humberto R. Maturana, Francisco J. Varela, "*El árbol del conocimiento*", 1984; also as : "*Der Baum der Erkenntnis*", Scherz Verlag, 1987, ISBN 3-502-13440-5; "*The Tree of Knowledge: The Biological Roots of Human Understanding*", Shambala Publications, 1992, ISBN 0877736421.

[Mat00] Humberto R. Maturana, "The Nature of the Laws of Nature," Systems Research and Behavioural Science, 17 (2000), pp. 459-468.

[Min85] Marvin Minsky, "*Society of Mind*", Simon & Shuster, 1985, ISBN 0-671-60740-5.

[Min94] J. Mingers, "*Self-Producing Systems: Implications and Applications of Autopoiesis*", Plenum Publishing, New York, 1994.

[Negr97] Nicolas Negroponte, "Wireless Revisited", Correspondence, Message 50: Date: 8.1.1997, http://www.wired.com/wired/5.08/negroponte_pr.html; From: nicholas@media.mit.edu, To: lr@wired.com.

[Peat96] D. F. Peat, "Blackfoot Physics", Fourth Estate Ltd., May 1996, ISBN 1-85702-456-7, http://www.fdavidpeat.com/bibliography/books/blackfoot.htm.

[Perl94] Donald Perlis, *"An Error-Theory of Consciousness",* Univ. of Meryland, Tech. Report., CS_TR-3324, 1994.

[Shel81] Rupert Sheldrake, "*A New Science of Life: The Hypothesis of Morphing Resonance*", London, Blond & Briggs, 1981, ISBN 0-89281-535-3.

[Schr58] Erwin Schrödinger, "*Mind and Matter*", Cambridge University Press, 1958.

[Smo00] Lee Smolin, "*Three Roads to Quantum Gravity: A New Understanding of Space, Time and the Universe*", Phoenix Science Masters, ISBN 0-75381-261-4, London, 2000.

[Var79] Francisco J. Varela, "Principles of Biological Autonomy", Appleton & Lange, 1979, ISBN 0135009502.

[Var81a] F. Varela, „"Autonomy and Autopoiesis", in: P. Heijl, , W. Kock, G. ROTH (Eds.), *Wahrnehmung und Kommunikation*, Frankfurt: Lang, 1978.

[Var81b] F. Varela, "Describing the Logic of the Living: The Adequacy and Limitations of the Idea of Autopoiesis", in: M. Zeleny (Ed.), Dissipative Structures and Spontaneous Social Orders, AAAS Selected Symposium 55, AAAS National Annual Meeting, Houston, TX, January 3-8, 1979, Boulder CO: Westview Press, 1980.

[VC99] Vincent Cerf, Cerf's UP, Presentation, MCI Worldcom Inc., August, 1999, http://www.wcom.com/about_the_company/cerfs_up/presentations/.

[VTR93] Francisco J. Varela, Evan Thompson, Eleanor Rosch, "*The Embodied Mind : Cognitive Science and Human Experience*", MIT Press, Cambridge, 1993, ISBN 0262720213.

[White00] George Whitesides, "Meso-Scale Self-Assembly", *Fourth International Workshop on Algorithmic Foundations of Robotics (WAFR'2000)*, March 16-18, Dartmouth College, Hanover, NH, USA.

[Wilb96] Ken Wilber, *"A Brief History of Everything",* Gil & Macmillan Ltd., Dublin, 1996, ISBN0-7171-2429-0.

WLI

[ABStat01] "ABone Progress and Status Report", Report to DARPA Active Nets PI Meeting, Orlando, Florida, Dec 3-5, 2001, http://www.isi.edu/abone/DOCUMENTS/anpi.orlando01.abonestatus.pdf.

[ANSPrice] D. S. Alexander, K. G. Anagnostakis, W. A. Arbaugh, A. D. Keromytis, J. M. Smith, "The Price of Safety in an Active Network", http://www.cis.upenn.edu/~switchware/papers/saneimp-jcn.pdf .

[Apo00] J. G. Apostolopoulos, "Error-resilient Video Compression Through the Use of Multiple States", Proc. Int. Conference on Image Processing ICIP'2000, Hewlett-Packard Laboratories, http://www.hpl.hp.com/personal/John_Apostolopoulos/papers/MDVideo_ImprovedRecovery_ICIP2000.pdf.

[BaZa89] L. A. Barford, B. T. V. Zanden, "Attribute Grammars in Constraint-based Graphics Systems", Software – Practice and Experience, Vol. 19(4), pp. 309-328, April, 1989.

[Bush00] S. F. Bush, A. Kulkarni, V. Galtier, Y. Carlinet, K. L. Mills, L. Ricciulli, "Predicting and Controlling Resource Usage in an Active Network", DARPA Active Networks PI Meeting, December 6-9, 2000, Atlanta, GA.

[Bush01] S. F. Bush, A. Kulkarni, "Active Networks and Active Virtual Network Management Prediction: A Proactive Management Framework", *Kluwer Academic / Plenum Publishers*, 2001, ISBN 0-306-46560-4.

[BBR00] S. Berson, B. Braden, L. Riciulli, "Introduction to the ABone", 15 June 2000, http://www.isi.edu/abone/DOCUMENTS/ABoneIntro.pdf.

[BCG82] E. R. Berlekamp, J. H. Conway, and R. K. Guy, "Winning Ways for Your Mathematical Plays", Vol. 2: Games in Particular, Academic Press, 1982, pp. 817–849.

[BCZ97C] S. Bhattacharjee, K. Calvert, and E. Zegura, "Active networking and the end- to-end argument," *Proc. of the Int. Conference on Network Protocols (ICNP'97)*, Atlanta, GA, Oct. 1997, pp. 220-228.

[Ber00] S. Berson, "A Gentle Introduction to the ABone", OPENSIG'2000 Workshop, Napa, CA, 11-12 October 2000, http://www.isi.edu/abone/DOCUMENTS/opensig00.berson.pdf.

[BLA01] Lucent Technologies Press Release, "Bell Labs scientists build the world's smallest transistor, paving the way for 'nanoelectronics' ", Murray Hill, N. J., Thursday, November 8, 2001, http://www.lucent.com/press/1101/011108.bla.html ..

[Brad97] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP)", Version 1, Functional Specification, IETF RFC 2205, September 1997, http://www.ietf.org; http://www.isi.edu/div7/rsvp/overview.html.

[Brown01] I. Brown, "End-to-end Security in Active Networks", PhD. Thesis, University of London, Dept. of Computer Science, September 2001.

[CaDe92] Casner, Stephen and Deering, Stephen. "First IETF Internt Audioast," ACM SIGCOMM Computer Communication Review, San Diego California, July 1992, pp. 92-97.

[CaSch93] S. Casner, H. Schulzrinne, "RTP: A Transport Protocol for Real-Time Applications," IETF Draft, 20 October 1993.

[Chan97] T. Chaney et al., "Design of a Gigabit ATM Switch", Proceedings of INFOCOM'97, April 1997.

[Chen02] M. Chen, “Error Resilient Video Compression Using Watermarking Techniques”, http://buffy.eecs.berkeley.edu/IRO/Summary/02abstracts/minghua.1.html .

[Cher01] A. Chervenak, “An Architecture for Replica Management in Grid Computing Environments”, Global Grid Forum 1, March 4-7, 2001, http://www.isi.edu/~annc/gridforum/ggf1replica.pdf.

[CHL01] M. Chen, Y. He, R. L. Lagendijk, "Error Detection by Fragile Watermarking", Proc. of Picture Coding Symposium, Seoul, Korea, April 2001.

[Chris97] C. M. Christensen, *The Innovator’s Dilemma*, Harvard Business School Press, Boston, MA. 1997, ISBN 0-87584-585-1.

[CiMa99] W. Cirne, K. Marzullo, “The Computational Co-op: Gathering Clusters into a Metacomputer”, Proc. 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, San Juan, Puerto Rico, 12 - 16 April, 1999, http://ipdps.eece.unm.edu/1999/papers/170.pdf.

[CiMa00] W. Cirne, K. Marzullo, “Open Grid: A User-Centric Approach for Grid Computing”, http://www.cs.ucsd.edu/~walfredo/papers/open-grid.pdf.

[ClTe90] D. D. Clark and D. L. Tennenhouse, “Architectural considerations for a new generation of protocols,” Proc. ACM SIGCOMM ’90, Sept. 1990, pp. 200–208.

[CoDo91] D. E. Comer, E. Douglas, “Internetworking with TCP/IP”, vol. I, Prentice-Hall, New Jersey, 1991.

[CCGrid02] 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002), May 2002, ISBN 0-7695-1582-7.

[CCH98] M. Civanlar, G. Cash, B. Haskell, “AT&T's Error Resilient Video Transmission Technique”, AT&T Labs-Research, RFC 2448, November 1998, http://www.faqs.org/rfcs/rfc2448.html.

[CLUS] “Cluster Computing, A Review of Cluster Characteristics Across Several Leading Vendors”, http://www.openvms.compaq.com/openvms/whitepapers/clusters-whitepaper.doc.

[DANCE02] DARPA Active Networks Conference and Exposition 2002 (DANCE 2002), May 2002, IEEE Computer Society Press, ISBN 0-7695-1564-9.

[Deer89] S. Deering, Stephen. "Host Extensions for IP Multicastig", RFC 1112, August 1989.

[DSB96] J.Darling, P.Sapaty, P.Borst, *"Modeling Virtual Worlds Using the WAVE Technology"*, presentation at the 14th Distributed Interactive Simulation Workshop, Orlando, FL, March 1996, http://www.ee.surrey.ac.uk/Research/DKP/.

[Erik94] H. Eriksson, “Mbone: The multicast backbone”,Communications of the ACM, August 1994, at 56-60.

[E2EC] T. M. Chen, A. W. Jackson (Eds.), “Commentaries on the Active Networking and End-To-End Arguments”, IEEE Network, May/June, 1998, pp. 66-71.

[Fayn97] I. Faynberg, L. R. Gabuzda, M. P. Kaplan, N. J. Shah, “The Intelligent Network Standards“, McGraw-Hill, 1997, ISBN 0-07-021422-0.

[GNUT] "The Gnutella Protocol Specification v0.4”, http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[FlJa98] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” *IEEE/ACM Trans. On Networking*, vol. 1, no. 4, Aug. 1998, pp. 397–413.

[Goet88] H. Göttler, “Graphgrammatiken in der Softwaretechnik: Theorie und Anwendungen”, Springer-Verlag, 1988.

[Goy01] V. K. Goyal, “Mutliple Description Coding: Compression Meets the Network, ” IEEE Signal Processing Magazine, pp. 74-93, Sept. 2001.

[HaGi97] F. Hartung, B. Girod, "Digital Watermarking of MPEG-2 Coded Video in Bitstream Domain", Proc. IEEE ICASSP, Munich, Germany, April 1997.

[Hen00] D. Henty, “The Grid, A Critical Review of Current Status and Future Directions in Grid Technology”, The DIRECT Initiative, EPCC, October 2000, http://www.epcc.ed.ac.uk/DIRECT/grid/.

[HGKB98] J.-P. Hubaux, C. Gbaguidi, S. Koppenhoefer, J.-Y. Le Boudec, “The Impact of the Internet on Telecommunication Architectures”, *Computer Networks and ISDN Systems*, Special Issue on Internet Telephony, Oct. 1998.

[Hub01] J.-P. Hubaux, Th. Gross, J.-Y. Le Boudec, M. Vetterli, “Towards Self-Organized Mobile Ad Hoc Networks: The Terminodes Project”, *IEEE Communications*, January, 2001, http://www.terminodes.org/publications/. `

[IPDPS02] 16th, International Parallel and Distributed Procesing Symposium (IPDPS 2002), April 2002, IEEE Computer Society Press, ISBN 0-7695-1573-8.

[ISVLSI02] 2002 IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2002), April 2002, IEEE Computer Society Press, ISBN 0-7695-1486-3.

[Jac88] V. Jacobson, “Congestion Avoidance and Control”, *Proc. ACM SIGCOMM’88*, August 1988, pp. 314–329.

[JaRa88] R. Jain, K. K. Ramakrishnan, “A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer, *Proc. ACM SIGCOMM ’88*, August 1988, pp. 303–313.

[KaPe01] S. Karlin, L. Peterson, “VERA: An Extensible Router Architecture”, Proc. of the 4th IEEE Conference on Open Architectures and Network Programming OPENARCH’2001, Anchorage, Alaska, April 26-28, 2001.

[Kast80] U. Kastens, "Ordered Attributed Grammars", *Acta Informatica 13*, pp. 229-256, 1980.

[Knuth68] Donald E. Knuth, "Semantics of Context-free Languages, *Mathematical System Theory*, 2(2), pp. 127-145, 1968.

[KLS98] V.P. Kumar, T.V. Lakshman, D. Stiliadis, "Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow’s Internet", IEEE Communcations Magazine, Vol. 36, No. 5, May 1998, pp. 152-164, http://www.bell-labs.com/~stiliadi/router/router.html.

[Konst02] A. V. Konstantinou, D. Florissi, Y. Yemini, ”Towards Self-Configuring Networks“, Proc. DARPA Active Networks Conference and Exposition (DANCE), May, 2002, IEEE Computer Society Press, ISBN 0-7695-1564-9.

[KRWP01] R. Keller, J. Ramamirtham, T. Wolf, B. Plattner, "Active Pipes: Program Composition for Programmable Networks", Proc. of IEEE MILCOM 2001, McLean, VA, October 2001.

[Kuhns02] F. Kuhns, J. DeHart, A. Kantawala, R. Keller, J. Lockwood, P. Pappu, D. Richards, D. Taylor, J. Parwatikar, E. Spitznagel, J. Turner, K. Wong, “Design of a High Performance Dynamically Extensible Router”, Proc. of DARPA Active Networks Conference and Exposition (DANCE), San Francisco, May 2002, http://www.tik.ee.ethz.ch/~keller/dance02.pdf.

[LiSt98] X. Lin, I. Stojemovic, “Geographic Distance Routing in Ad-Hoc Wireless Networks”, Tech. Report, Dept. of Computer Science, SITE, Univ. of Ottawa, Canada, Dec. 1998.

[LLB97] G. C. Langelaar, R. L. Lagendijk, J. Biemond, "Real-time Labeling Methods for MPEG Compressed Video", Symp. On Information Theory in the Benelux, Veldhoven, The Netherlands, May 1997.

[MaFl01] I. Matyasovszki, C. Flanagan, “Scalable Routers, Programmable Network Processors, MPLS and Others, are these real solutions ?”, First Joint IEI/IEE Symposium on Telecommunications Systems Research, 27th November 2001, IEI, 22 Clyde Road, Dublin 4, telecoms.eeng.dcu.ie/symposium/papers.

[Moy93] J. Moy, “Multicast Extensions to OSPF”, IETF Draft, July 1993.

[Meru99] S. Merugu, et. al., “Bowman and CANEs: Implementation of an Active Network” , invited paper at the 37th Annual Allerton Conference on Communication, Control and Computing, Monticello, IL, September, 1999.

[MeiFa99] J. Meierhofer, G. Fankhauser, “Error-Resilient, Tagged Video Stream Coding with Wireless Data Link Control”, Proc. IEEE Conf. on Wireless Personal Multimedia Communications WPMC’99, http://www.tik.ee.ethz.ch/~gfa/papers/wpmc99.pdf.

[Mont02] A. Montresor, H. Meling, O. Babaoglu, “Towards Adaptive, Resilient and Self-Organizing Peer-to-Peer Systems”, http://www.newcastle.research.ec.org/cabernet/workshops/radicals/2002/Papers/.

[NieCh00] A. Nielsen, I. L. Chuang, “Quantum Computation and Quantum Information“, Cambridge University Press, 2000, ISBN 0-521-63235-8.

[NoVo01] T. Noll, H. Vogler, "The Universality of Higher-Order Attributed Tree Transducers", Theory Comput. Systems, Vol. 34, pp. 45-75, Springer-Verlag, New York, 2001.

[OSIRM] Information Technology – Open Systems Interconnecton – Basic Reference Model: The Basic Model, ITU-T Recommendation X.200, International Telecommunication Union, July 1994.

[Holm97] F. Holmgren, “A Grammar-based Approach to Design and its Application to Electronics and Logic Programming”, Res. Report, Intelligent Systems Laboratory, Swedish Institute of Computer Science (SICS), December, 1997.

[HuBr98] T. S. Hussain, R. A. Browse, “Attribute Grammars for Representations of Neural Networks and Syntactic Constraints of Genetic Programming” Proc. AIVIGI’98, Workshop on Evolutionary Computation, June 17th, 1998, Vancouver, BC, Canada.

[P2PDes] M. Ripeanu, I. Foster, A. Iamnitchi, "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design", University of Chicago, Computer Science, 2001, http://people.cs.uchicago.edu/~matei/PAPERS/ic.pdf.

[PST95] Parulkar, G.M., Schmidt, D.C., Turner, J.S., “aItPm : A Strategy for Integrating IP with ATM,” Proc. of SIGCOMM’95, August 1995.

[Rad82] G. Radin, “The 801 Minicomputer,” Proc. 1st ACM Symp. Programming Languages and Op. Sys., in Comp. Architecture News, vol. 10, no. 2, Mar. 1982, pp. 39–47.

[Reed76] D. P. Reed, “Processor Multiplexing in a Layered Operating System,” S.M. and E.E. thesis, MIT Dept. of EECS, 1976; available as MIT Laboratory for Computer Science tech. rep. LCS-TR-164, July 1976.

[Rein97] A. Reinefeld, R. Baraglia, T. Decker, J. Gehring, D. Laforenza, F. Ramme, T. Römke, and J. Simon, "The MOL Project: An Open, Extensible Metacomputer", 6th Heterogeneous Computing Workshop (HCW '97) , April 1, 1997 Geneva, Switzerland.

[RFC2460] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," IETF RFC 2460, December 1998.

[Ridge97] D. Ridge, D. Becker, P. Merkey, and T. Sterling, "*Beowulf:* Harnessing the Power of Parallelism in a Pile-of-PCs", Proc. of IEEE Conf. on Aerospace, 1997.

[Rov01] C. Rovelli, "Notes for a Brief History of Quantum Gravity", April 20th , 2001, http://xxx.lanl.gov/gr-qc/0006061.

[Saltz81] H. Saltzer, D. Reed and D. Clark, "End-to-end arguments in system design," Proc. 2d Intn'l Conf. Dist. Comp. Sys., April, 1981; also in: ACM Trans. Comp. Sys., vol. 2, no. 4, 1984.

[Sap96] P.S.Sapaty, *"Mobile WAVE Technology for Distributed Knowledge Processing in Open Networks"*, Proc. Workshop on New Paradigms in Information Visualization and Manipulation", Proc. of the CIKM'95 Conference, Baltimore, MD, Nov.- Dec. 1996, http://www.cs.umbc.edu/~cikm/1995/npiv/sapaty/paper.ps.

[Sap99] P. Sapaty, *"Mobile Processing in Distributed and Open Environments",* John Wiley & Sons, 1999, ISBN 0-471-19572-3.

[SaSw99] J. Saraiva, D. Swierstra, "Generic Attribute Grammars", In: D. Parigot and M. Mernik, (Eds.), "Second Workshop on Attribute Grammars and their Applications", WAGA'99, pp. 185-204, Amsterdam, The Netherlands, March 1999.

[Schr90] Schroeder, M. D., Birrell, A. D., Burrows, M., Murray, H., Needham, R. M., Rodeheffer, T. L., Satterthwaite, E. H., Thacker, C P., "Autonet: a High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links", *Research Report 59*, Digital Equipment Corporation, Systems Research Center, April 1990.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[SFSS00] S. Schmid, J. Finney, A. C. Scott, W. D. Shepherd, "Component-based Active Networks for Mobile Multimedia Systems", http://citeseer.nj.nec.com/415468.html.

[SRL98] H. Schulzrinne, A. Rao, R. Lanphier, "Real Time Streaming Protocol (RTSP)", IETF RFC 2326, April 1998.

[Sim96] P. L. Simeonov, "Add-On IN Gateway Services for Service Node in Distributed Telecom Networks", *Tech. Report*, Nr. P30308-A8720-A000-04-7618, Siemens AG, Oct. 1996.

[SiHo97] P. L. Simeonov, P. Hofmann, "A Distributed Intelligent Computer/Telephony Network Integration Architecture for Unified Media Communication", *Proc. of the 2IN'97 (Intelligent Networks and Intelligence in Networks, IFIP)*, Paris, September 2-5, 1997, pp. 3-8, Chapman & Hall, D. Gaiti (Ed.), ISBN 0-412-82950-9.

[Sim97a] P. L. Simeonov et al., "@INGate: A Distributed Intelligent Network Approach to Bridge Switching and Packet Networks", *Proc. of the ICCCN'97 (Sixth International Conference on Computer Communications and Networks, IEEE)*, Las Vegas, Nevada, September 22-25, 1997, pp. 358-363, IEEE Computer Society, ISBN 0-8186-8186-1.

[Sim97b] P. L. Simeonov, et al., "A Smart Node Architecture Linking Telephony with the Internet", *Proc. of the ICCC'97 (Thirteenth International Conference on Computer Communication, IFIP)*, Cannes, France, November 19-21, 1997, pp. 77-81, International Council for Computer Communication (ICCC), ISBN 2-7261-1104-1.

[Sim97c] P. L. Simeonov, et al., "@INGate: Integrating Telephony and Internet", *Proc. of PROMS-MmNet'97 (International Conference on Protocols for Multimedia Systems and Multimedia Networking, IEEE)*, Santiago de Chile, November 24-27, 1997, pp. 261-264, IEEE Computer Society, ISBN 0-8186-7916-6.

[Sim98] P. L. Simeonov, "The SmartNet Architecture or Pushing Networks beyond Intelligence", *Proc. of ICIN'98 (5th International Conference on Intelligent Networks)*, 12-15 May, Bordeaux, France, pp. 182-185

[SimWLI99] P. L. Simeonov, "Netzelement in einem Intelligenten Telekommunikationsnetz", Europäische Patentanmeldung, Reg. Nr. EP 0-957-645-A2, European Patent Office, Registration Date: Mai 7th 1999.

[Sim99c] P. L. Simeonov, "On Using Nomadic Services for Distributed Intelligence", *Proc. of ICCCN'99 (Eight IEEE International Conference on Computer Communications and Networks)*, October 11 - 13, 1999, Boston, MA, USA, pp. 228-231, IEEE Press, ISBN: 0-7803-5794-9, http://www.icccn99.cstp.umkc.edu/; also in the Journal of Microprocessors and Microsystems, Elsevier Science Publishers, October, 2000.

[Sim99e] P. L. Simeonov, "A Path Towards the Wandering Logic of Intelligence", *Proc. PONMS'99* (Modal & Temporal Logic based Planning for Open Networked Multimedia Systems), AAAI 1999 Fall Symposium Series, November 5-7, 1999, North Falmouth, MA, USA, Working Notes, pp. 78-84, AAAI Press, http://www.cs.toronto.edu/~daoud/AAAI99/.

[Sim99f] P. L. Simeonov, "The Wandering Logic of Intelligence: Or Yet Another View on Nomadic Communications", *Proc. of SMARTNET'99* (The Fifth IFIP TC6 WG 6.7 Conference on Intelligence in Networks), 22-26 November 1999, Asian Institute of Technology, Pathumthani, Thailand, pp. 293-306, Kluver Academic Publishers, ISBN: 0-7923-8691-1, http://www.cs.ait.ac.th/~ca/smartnet99/.

[Sim99g] P. L. Simeonov, "Towards a Vivid Nomadic Intelligence with the SmartNet Media Architecture", Proc. of *IMSA 1999*, Nassau, Bahamas, Oct. 1999, pp. 249-254, http://www1.acm.org:82/sigmod/dblp/db/indices/a-tree/s/Simeonov:Plamen_L.

[Sim00] P. L. Simeonov, et. al., "*MediaPEP* (Media Performance Enhanced Proxy), A Scaleable and Dependable ActiveUMTS QoS Management Server, An Internet Protocol Booster and An Adaptive Media Transcoder Switch Architecture for E2E-IP Integration of Mobile Wireless Interactive Telepresence Applications, Project "Odysseus-2001", *White Paper*, Siemens AG, Heinrich Hertz Institute, TU Berlin, TU Ilmenau, Northeastern University (Boston), November, 2000.

[Sim01] P. L. Simeonov, "The Wandering Network, a Glance at an Evolving Reality" (in German: "Das Wandernde Netzwerk"), *Proc. of Netobjectdays'2001*, 2. Joint GI conference "Object Orieneted Programming for a Networked World", 11-13 Sept., 2001, Erfurt, Germany, http://www.prakinf.tu-ilmenau.de/IPI/FGT/vertel.html.

[Sim02a] P. L. Simeonov, "The *Viator* Approach: About Four Principles of Autopoietic Growth On the Way to Hyperactive Network Architectures", *Annual IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems (FTPDS'02) at the 2002 IEEE International Symposium on Parallel & Distributed Processing (IPDPS'02)*, April 15-19, 2002, Ft. Lauderdale, FL, USA, http://www.ipdps.org/ipdps2002/.

[Sim02b] P. L. Simeonov, "*WARAAN*, A Higher-Order Adaptive Routing Algorithm for Wireless Multimedia in Wandering Networks", *5th IEEE International Symposium on Wireless Personal Multimedia Communications (WPMC'2002)*, October 27-30, 2002, Honolulu, Hawaii, USA, http://www.wpmc02.gatech.edu/technical_WP.html.

[SiRe02a] P. L. Simeonov, D. Reschke, "Supporting Adaptive Multimedia with the Wandering Network Model", *4th Int. Forum on Multimedia and Image Processing (IFMIP'2002) at the World Automation Congress (WAC'2002),* June 9-13, 2002, Orlando, Florida, USA, http://wacong.com.

[SiRe02b] P. L. Simeonov, D. Reschke, "*Viator*: A Hyperactive Network Architecture for Multimedia Telematics Applications", *6th IASTED Conference Internet and Multimedia Systems and Applications (IMSA'2002),* August 12-14, 2002, Kauai, Hawaii, USA, http://www.iasted.org/conferences/2002/hawaii/imsa.htm.

[Stev00] R. Stevens, "Future Directions in Computer and Systems Architecture for Scientific Computing", 05/24/2000, http://www-fp.mcs.anl.gov/~stevens/all-presentations/ITAMP-master-00/.

[SJC00] J. P. G. Sterbenz, Alden W. Jackson, M. N. Condell, "HyperActive Networking Architecture", INTERNET-DRAFT, 1 April 2000, http://www.ir.bbn.com/projects/sencomm/doc/draft-hypean.txt.

[SSV99] V. Srinivasan, S. Suri, G. Vaghese, "Packet Classification using Tuple Space Search", Proc. of ACM SIGCOMM'99, ACM Press, Cambridge, Mass., USA, 1999.

[Tan96] A. S. Tanenbaum, "Computer Networks", 3d Edition, Prentice Hall, 1996, ISBN 0-13-3499-45-6.

[TaZa98] W. Tan, A. Zakhor, "Internet Video Using Error Resilient Scalable Compression and Cooperative Transport Protocol ", Proc. of IEEE Int. Conference on Image Processing ICIP'98, http://www-video.eecs.berkeley.edu/~dtan/icip98.pdf.

[TaZa99] W. Tan, A. Zakhor, "Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol", IEEE Trans. On Multimedia, Vol. 1, No. 2, June 1999, http://www-video.eecs.berkeley.edu/~dtan/mm99.pdf.

[Tschu00] C. Tschudin, "Header Hopping and Packet Mixers", Proc. of ICCCN'2000, October 2000.

[TsFu80] W.-H. Tsai, K.-S. Fu, "Attributed Grammar – A Tool for Combining Syntactic and Statistical Approaches in Pattern Recognition", IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-10, No. 12, Dec. 1980.

[TD98] C. Tschudin, D. Decasper, "Simple Active Packet Format (SAPF)", Experimental RFC, August 1998, http://abone.ifi.unizh.ch/sapf/.

[TLG00] C. Tschudin, H. Lundgren, H. Gulbrandsen, "Active Routing for Ad Hoc Networks", IEEE Communications Magazine, April 2000.

[TTL01] D. E. Taylor, J. S. Tuner, J. W. Lockwood, "Dynamic Hardware Plugins (DHP): Exployting Reconfigurable Hardware for High-Performance Programmable Routers", Proc. IEEE OPENARCH'2001, Anchorage, Alaska, April 2001.

[VSK89] H. H. Vogt, S. D. Swierstra, M. F. Kuiper, "Higher Order Attribute Grammars", Proc. of the ACM SIGPLAN'89 Conference on Programming Language Design and Implementation, ACM SIGPLAN Notices, 24(7), 1989.

[WDT00] T. Wolf, D. Decasper, C. Tschudin, "Tags for High Performance Active Networks", Proc. OPENARCH' 2000, Tel Aviv, March 2000.

[Wolf99] T. Wolf, “A Proposal for a High-Performance Active Hardware Architecture, Tech. Report WUCS-99-08, Dept. of Computer Science, Washington University, February 15, 1999, http://ccrc.wustl.edu/~wolf/papers/wucs-99-08.pdf.

[WoFr02] T. Wolf, M. Franklin, "Design Tradeoffs for Embedded Network Processors", Proc. of Int. Conference on Architecture of Computing Systems (ARCS), Karlsruhe, Germany, April 2002.

[WoTu99] T. Wolf, J. S. Turner, “Design Issues for High Performance Active Routers”, Tech. Report WUCS-99-19, Dept. of Computer Science, Washington University, June 11, 1999, http://ccrc.wustl.edu/~wolf/app/JSAC.pdf.

[WoTu01] T. Wolf, J. Turner, "Design Issues for High Performance Active Routers", IEEE Journal on Selected Areas of Communications, Special Issue on Active and Programmable Networks, vol. 19, no. 3, pp. 404-409, March 2001.

[Wu97] T.-H. Wu *et all., “*Distributed Interactive Video System Design and Analysis”, *IEEE Communications Magazine*, March 1997, pp. 100-108.

[YWL02] P. Yin, M. Wu, B. Liu, "A Robust Error Resilient Approach for MPEG Video Transmission over Internet", Proc. of VCIP’02, http://www.ece.umd.edu/~minwu/public_paper/vcip02_errcon.pdf.

[Zub95] Dittia, Zubin, Jerome R. Cox, Jr., and Guru Parulkar. “Design of the APIC: A High Performance ATM Host-Network Interface Chip”, Proc. of INFOCOM’95, April 1995.

[ZRR00] R. Zhang, S. Regunathan, K. Rose, “Video Coding with Optimal Inter/Intra-mode Switching for Packet Loss Resilience”, IEEE Journal of Selected Areas in Communications, Vol. 18, No. 6, pp. 966-976, June 2000.

[ZT01] S. Zhu, B. Todur, “Efficient Error-Resilient Video Compression by Using Optimal Distribution of Intra/Inter Macroblocks for Multi-state Streams”, Cisco Systems, http://www.stanford.edu/class/ee398b/project/14/prop.pdf .

APPENDIX A: MAINTAINING ROUTING INFORMATION IN A WANDERING NETWORK

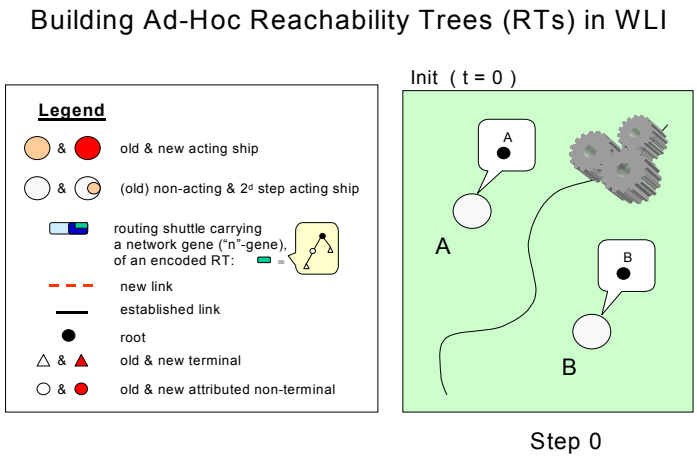


Figure 104: A symbol legend and the initial state (Step 0, t=0) of a Wandering Network[170].

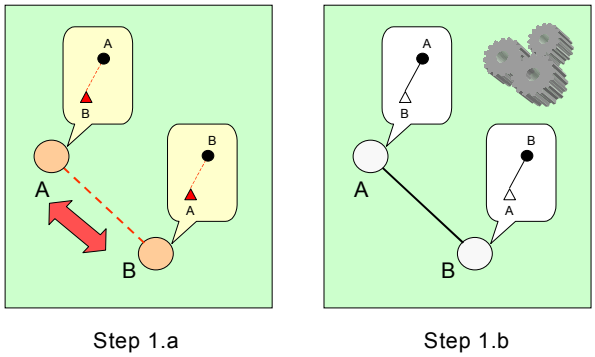


Figure 105: Step 1 (a, b) – Establishing a contact between the first two nodes, A and B, of the wandering network and building/maintaining[170] their initial reachability trees.

[170] The „gears" symbol illustrates a „stuttering" (repeating) state in terms of TLA. In our case, such a state includes the periodical exchange of *presence* shuttles (*p*-shuttles) which confirm and refresh the connectivity and availability of a netbot in the r-trees of its neighbors for routing purposes.

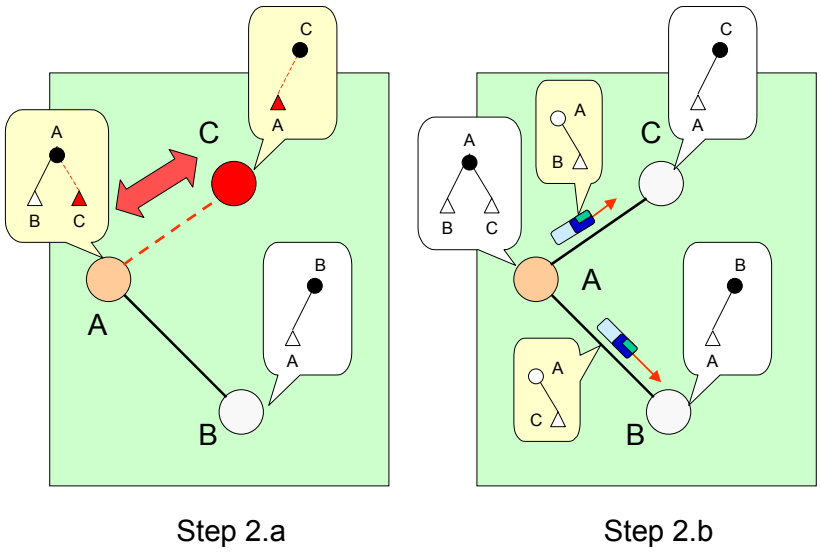


Figure 106: Step 2 (a, b) – Establishing a contact to a new, third netbot C, expanding/building the reachability trees of the corresponding netbots (A and C), and transmitting the new structural information to their neighbors via r-shuttles.

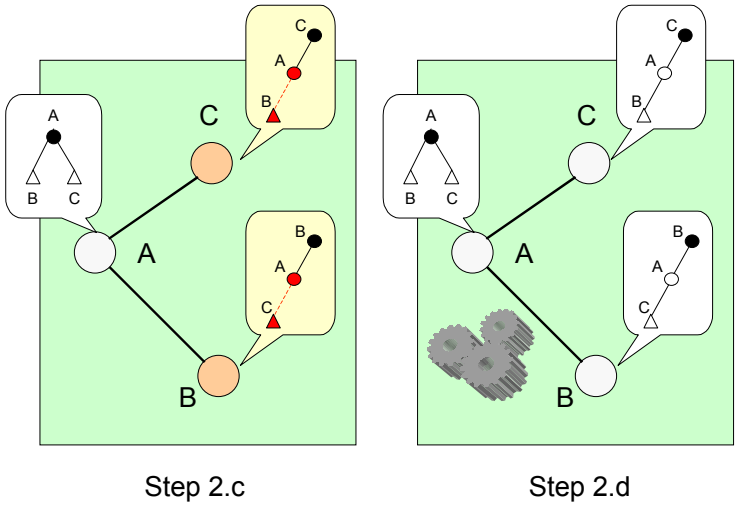


Figure 107: Step 2 (c, d) – Expanding and maintaining the reachability trees in the neighbor netbots (B and C) by the updating information contained in the r-shuttles.

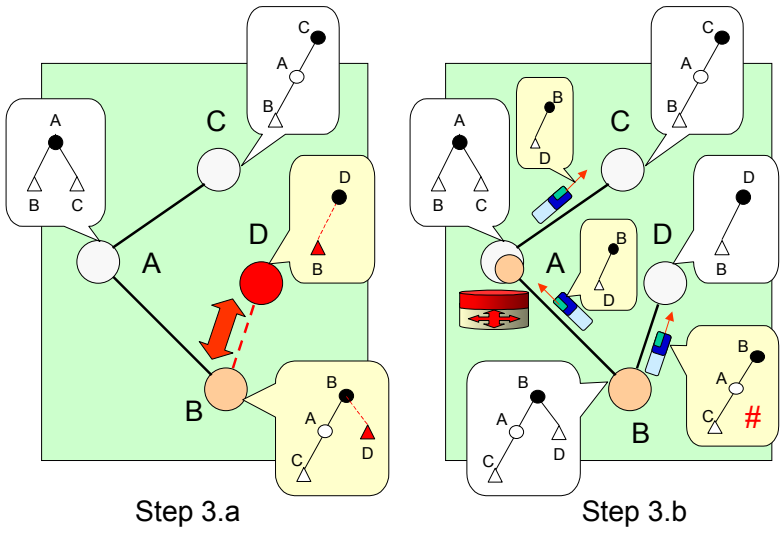


Figure 108: Step 3 (a, b) – Establishing a contact to a new, fourth netbot D, expanding/building the reachability trees of the corresponding netbots (B and D), and transmitting the new structural information to their neighbors via r-shuttles. Furthermore, node A is serving as a router for the r-shuttles from B to C.

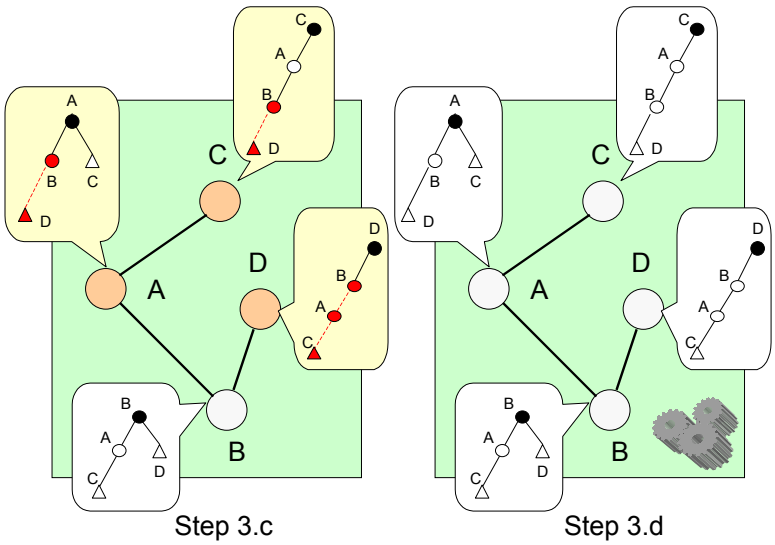


Figure 109: Step 3 (c, d) – Expanding and maintaining the reachability trees in the neighbor netbots (A, C and D) by the updating information contained in the r-shuttles. Note that two nodes in a single step expand the r-tree at node D only (!!).

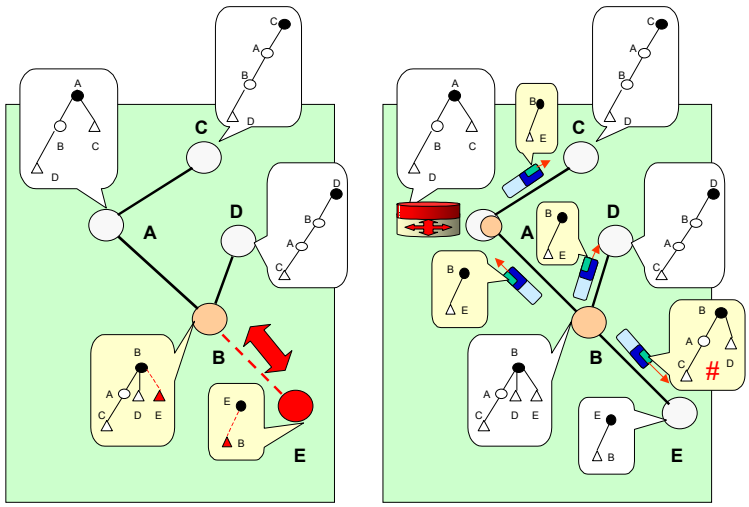


Figure 110: Step 4 (a, b) – Establishing a contact to a new, fifth netbot E, expanding/building the reachability trees of the corresponding netbots (B and E), and transmitting the new structural information to their neighbors via r-shuttles. Furthermore, node A is serving as a router for the r-shuttles from B to C.

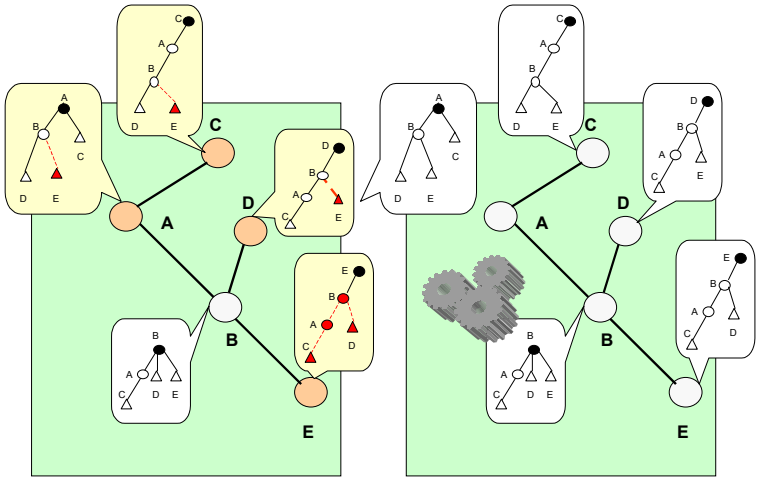


Figure 111: Step 4 (c, d) – Expanding and maintaining the reachability trees in the neighbor netbots (A, C, D and E) by the updating information contained in the r-shuttles. Note that three nodes in a single step expand the r-tree at node E only (!!).

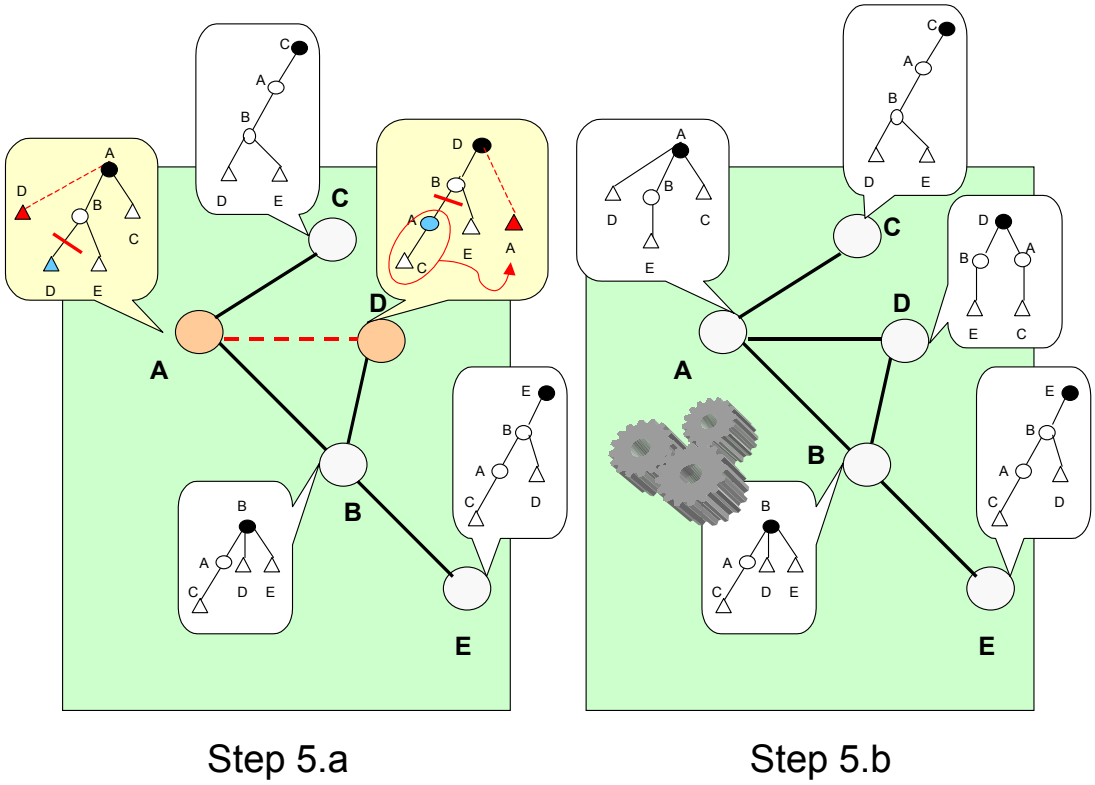


Figure 112: Step 5 (a, b) - Establishing a *new* contact between two present netbots in the network, A and D followed by expanding/building and maintaining their reachability trees. The propagation of the new connectivity information throughout the network is not provided here.

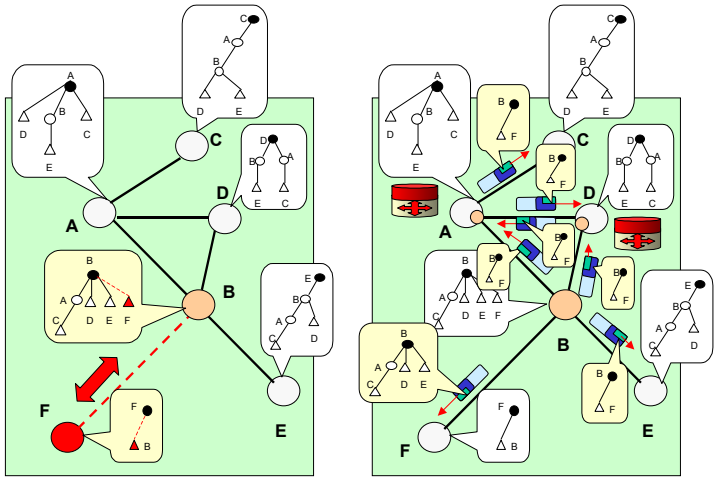


Figure 113: Step 6 (a, b) – Establishing a contact to a new, sixth netbot F, expanding/building the reachability trees of the corresponding netbots (B and F), and transmitting the new structural information to their neighbors via r-shuttles. Furthermore, nodes A and D are serving as routers for the r-shuttles from B to C. Redundant r-shuttle information obtained later at A, C and D is discarded.

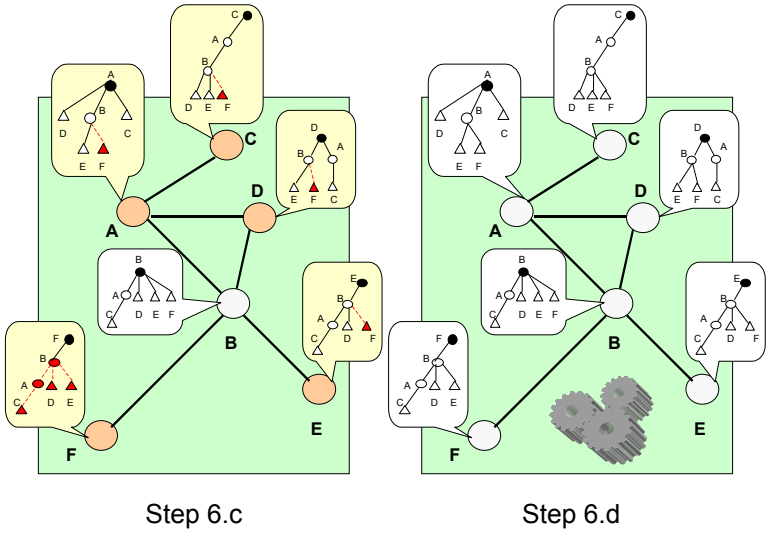


Figure 114: Step 6 (c, d) – Expanding and maintaining the reachability trees in the neighbor netbots (A, C, D, E and F) by the updating information contained in the r-shuttles. Note that four nodes in a single step expand the r-tree at node F only (!!).

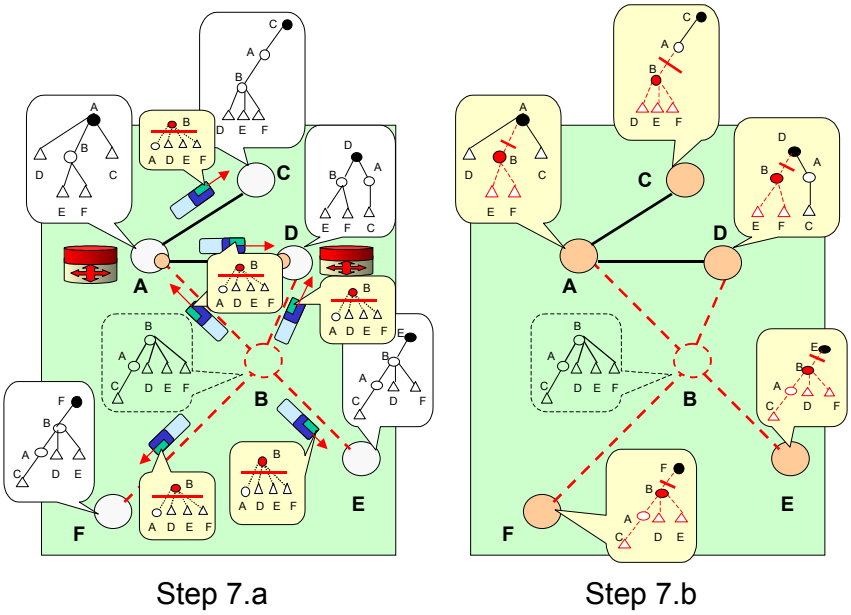


Figure 115: Step 7 (a, b) – Node B leaving the network by reporting the event to its neighbors via x-shuttles; x-shuttle propagation to all present netbots; evaluation and update of the new reachability trees in all netbots of the wandering network in a single step only (!).

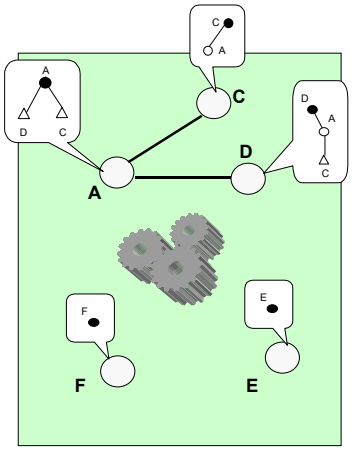


Figure 116: Step 7.c – R-Tree maintenance after having node B left the wondering network.

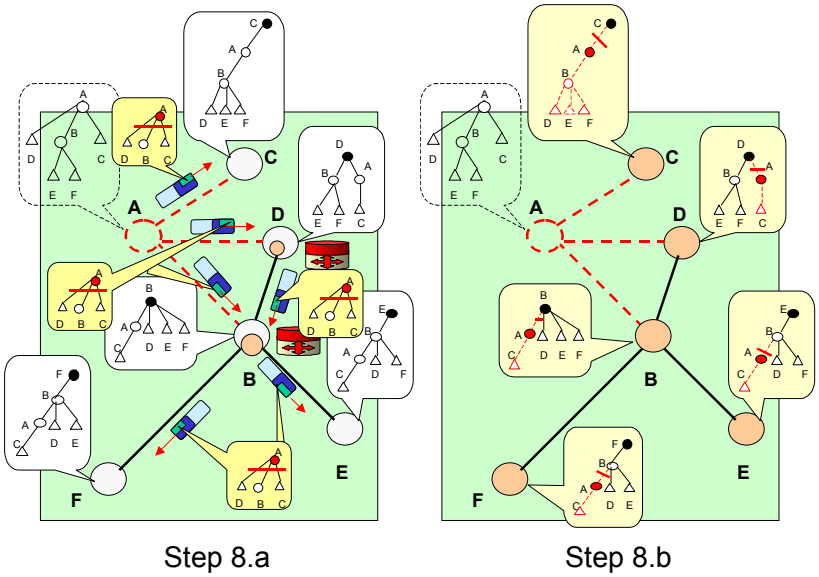


Figure 117: Step 8 (a, b) from Step 6.d – Node A leaving the network by reporting the event to its neighbors via x-shuttles; x-shuttle propagation to all present netbots; evaluation and update of the new reachability trees in all netbots of the network in a single step only (!).

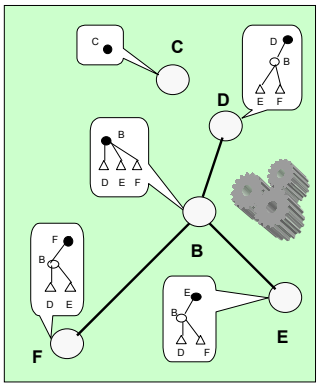


Figure 118: Step 8.c – R-Tree maintenance after having node A left the wondering network.

APPENDIX B: THE TEMPORAL LOGIC OF ACTIONS

TLA defines a collection of values, `Val`, for an infinite set of variable names, `Var`. Further, algorithms manipulate data such as numbers, strings and sets and assign values to variables.

THE LOGIC OF ACTIONS

(Values, variables and state)

A TLA system description consists of a sequence of states and expressions about their changes as a result of actions. A system state `s` assigns[171] values to variables:

- `s \in St = Var -> Val`
- `s[[x]] := s(x)`
- `[[x]] \in St -> Val`

STATE FUNCTIONS AND PREDICATES

A *state function* `f` is an expression built from variables and constant symbols. Its meaning is defined[172] as: `s[[f]] := f(\forall`[173] `'v': s[[v]]/v).`

State functions correspond to program expressions (and sub-expressions of assertions).

A *state predicate* P is a Boolean expression:

- `s[[P]] \in {true, false}`
- `s |- P iff s[[P]] = true`

State predicates correspond to assertions (and Boolean valued program expressions).

ACTIONS

An *action* A is a Boolean valued expression about variables, primed variables and constant symbols. It represents an atomic operation of a concurrent program: `x'+y = y, x-1 \in z'`, etc. Each action represents a relation between an old state and a new state so that:

- unprimed variables (`v`) refer to old (present) states.
- primed variables (`v'`) refer to new (future) states.

The formalization of the action `A` is expressed as follows:

- `[[A]] \in St -> S -> Bool`
- `s:` old state, `t:` new state
- `s[[A]]t \in Bool`
- `s[[A]]t <=> A(\A 'v': s[[v]]/v, t[[v]]/v')`
- `s[[y = x'+1]]t = (s[[y]] = t[[x]]+1)`
- `s,t` is an `A` step `iff s[[A]]t = true`

---

[171] The semantics (meaning) of a syntactic object x is written in TLA as [[x]].

[172] The value obtained from `f` by substituting `s[[v]]` for `v`, for all variables `v`. For instance: `s[[2x+y-3]] = 2(s[[x]])+s[[y]]-3`.

[173] Henceforth, we will encode the expression "`\forall`" as "`\A`".

PREDICATES AS ACTIONS

A predicate `P` can be viewed as an action without any primed variables:

- `s[[P]]`: boolean \A `s`
- `s[[P]]t = s[[P]] \A s,t`
- `s,t is a P step iff s |- P`

The replacement of unprimed variables of a state function or predicate `F` is defined as:

- `F' := F(\A 'v': v'/v)`
- `s[[P']]t = t[[P]]`

THE ENABLED PREDICATE

If `A` represents atomic operation, `Enabled A` is true for those states in which it is possible to perform this operation.

`Enabled A`:

- `s[[Enabled A]] <=> \E t \in St: s[[A]]t`
- `true` for `s` iff it is possible to take an `A` step starting in `s`

Syntactic definition:

- v_i : all (flexible) variables in `A`.
- Enabled A <=> \E c_1, ..., c_n: A(c_1/v'$_1$, ..., c_n/v'$_n$).
- Enabled(y = (x')^2+n) = \E c: y = c^2+n

VALIDITY AND PROVABILITY

An action `A` is valid (`|= A`) :

- every step is an `A` step
- `|= A <=> \A s,t \in St: s[[A]]t`
- `|= P <=> \A s \in St: s[[P]]`
- `true` regardless of what values are substituted for primed and unprimed variables
- `(x'+y in Nat) => (2(x'+y) >= x'+y)`

A formula `F` is valid (`|= F`) :

- `|= F <=>` \A Z \in $St^{infinity}$: Z[[F]]
- ^{infinity} : set of all possible behaviours.

Representation of an algorithm:

- If an algorithm is described by the temporal formula `F`, then
- `Z[[F]] = true` iff `Z` represents a possible execution of the algorithm.

Property G of algorithm:

- Syntax: `|= F => G`
- Semantics: the algorithm represented by F satisfies property G.

A formula `F` is provable (`|- F`) if it can be formally derived by rules of logic. A logic is sound if every provable formula is valid:

- `|- F => |= F`

RIGID VARIABLES AND QUANTIFIERS

TLA uses two kinds of variables:

- *rigid* variables (unknown constants), and
- (*flexible*) variables (program, state-dependent variables).

Constant expressions are built from rigid variables and constant symbols. State functions and actions are extended to contain constant expressions.

Quantification over rigid variables:

- `s[[\exists`[174] `m \in Nat: mx' = n+x]] <=>`
  `\E m \in Nat: m(t[[x]]) = n+s[[x]]`
- `s[[A]]t = true \A s,t \in St, \A` possible values of `A`'s free rigid variables `-> |= A`

TEMPORAL FORMULAS

The basic temporal operator in TLA is *always* (`[]`). If E_1, E_2 are two elementary formulas, then from `not` E_1 `and [](not` E_2`)` follows `[](`E_1 `=> [](`E_1 `or` E_2`)).`

The semantics of temporal formulas is based on behaviours.

If we consider infinite sequences of states, then a behaviour `Z` is defined as:
`Z = <`s_0`,` s_1`, ...>`, where

- `Z[[F]] \in Bool`
- `Z |- F iff Z[[F]] = true`

The meaning of temporal formulas is defined as follows:

```
<s0, s1, ...>[[F]]      <=> s0[[F]], if F elementary.
Z[[F and G]]    <=> Z[[F]] and Z[[G]]
Z[[not F]]      <=> not Z[[F]]
<s0, s1, ...> [[ []F ]] <=> \A n \in Nat: <sn, sn+1, ...>[[F]]
```

DERIVED TEMPORAL FORMULAS

Eventually (`<>`):

- `F` is eventually true
- Syntax: `<> F <=> not [] not F`
- Semantics: `<`s_0`,` s_1`, ...> [[ <>F ]] <=>`
  `\E n \in Nat: <`s_n`,` s_{n+1}`, ...>[[F]]`

Infinitely Often (*always eventually*, `[]<>`):

```
<s0, s1, ...> [[ []<>F ]] <=>
  \A n in Nat: \E m \in Nat: <sn+m, sn+m+1, ...>[[F]]
```

Eventually Always (`<>[]`):

```
<s0, s1, ...>[[ <>[]F ]] <=>
  \E n \in Nat: \A m in Nat: <sn+m, sn+m+1, ...>[[F]]
```

Leads to (`|->`):

Syntax: `F |-> G <=> [](F => <>G)`
Semantics: any time `F` is true, `G` is true then or at some later time.

---

[174] Henceforth, we will encode this as "`\E`" .

ADDITIONAL NOTATIONS

p'	<=>	p(\A 'v': v'/v)
$[A]_f$	<=>	A or (f' = f)
$\langle A \rangle_f$	<=>	A and (f' =\= f)
Unchanged f	<=>	f' = f
<> F	<=>	not [] not F
F |-> G	<=>	[] (F => <> G)

THE RAW TEMPORAL LOGIC OF ACTIONS (RTLA)

TLA formulas are a subset of RTLA formulas: elementary formulas of the form $[][A]_f$.

Elementary temporal formulas are *actions*.

Action A is true on behaviour Z:

- Syntax: $\langle s_0, s_1, ... \rangle [[A]] <=> s_0[[A]]s_1$
- Semantics: the first pair s_0, s_1 of behaviours is an A step.

Temporal operator:

$\langle s_0, s_1, ... \rangle$ [[[]A]]
<=> \A n in Nat: $\langle s_n, s_{n+1}, ... \rangle [[A]]$
<=> \A n \in Nat: $s_n[[A]]s_{n+1}$.

Predicates:

$\langle s_0, s_1, ... \rangle [[P]] <=> s_0[[P]]$
$\langle s_0, s_1, ... \rangle$ [[[]P]] <=> \A n \in Nat: s_n[[P]]

FAIRNESS

Arbitrary liveness properties in a specification are dangerous. They are used to express fairness requirements, but they may unexpectedly[175] add safety properties. Therefore, the solution in TLA is to express liveness by fairness:

Fairness:
If operation is possible, then program must eventually execute it.

Fairness at all times:

- always ((eventually executed) or (eventually impossible))
- always ((eventually executed) or (eventually always impossible))

… equivalent to:

- (always eventually executed) or (always eventually impossible)
- (always eventually executed) or (eventually always impossible))

Formalization:

- executed <=> $\langle A \rangle_f$
- impossible <=> not Enabled $\langle A \rangle_f$

Weak Fairness $WF_f(A)$:

[175] e.g., adding the expression "[]<>(x=0)" leads to the consequence that x never changes!

Operation must be executed if it remains possible to do so *for long enough time*.
(eventually executed) or (eventually impossible)

$WF_f(A) <=> ([] <><A>_f) or ([] <> not Enabled <A>_f)$

Strong Fairness $SF_f(A)$:

Operation must be executed if it is *often enough* possible to do so.
(eventually executed) or (eventually always impossible)

$SF_f(A) <=> ([] <><A>_f) or (<> [] not Enabled <A>_f)$

A strong fairness implies weak fairness:

$SF_f(A) => WF_f(A)$

SIMPLE TEMPORAL LOGIC

The execution of an algorithm is:

- a sequence of steps, where
- each step produces new state changing the values of variables, and
- the semantic meaning of the algorithm is collection of all possible executions.

The Syntax of Simple TLA

TLA is logic without quantification:

```
<formula>                                  <=>          <predicate>
        | '[]'[<action>]state function | not <formula>
        | <formula> and <formula> | '[]' <formula>

<action>                                   <=>          Boolean-valued expression
    of constant symbols, variables, and primed variables

<predicate>                <=>          action with no primed variables
        |                  Enabled <action>

<state function>           <=>          non-Boolean expression
                                         containing constant symbols and variables
```

The Semantics of Simple TLA

- `s[[f]] <=> f(\A 'v': s[[v]]/v)`
- `s[[A]]t <=> A(\A 'v': s[[v]]/v, t[[v]]/v')`
- $\langle s_0, s_1, ...\rangle[[A]] <=> s_0[[A]]s_1$
- `|= A <=> \A s,t \in St: s[[A]]t`
- `s[[Enabled A]] <=> \E t \in St: s[[A]]t`
- $\langle s_0, s_1, ...\rangle[[\ ]F]] <=>$
- $\A\ n \in Nat: \langle s_n, s_{n+1}, ...\rangle[[F]]$
- `Z[[F and G]] <=> Z[[F]] and Z[[G]]`
- `Z[[not F]] <=> not Z[[F]]`
- $|= F <=> \A\ Z \in St^{infinity}: Z[[A]]t$

THE RULES OF SIMPLE TEMPORAL LOGIC

STL1.

$$\frac{F \text{ provable by propositional logic}}{[]\ F}$$

STL2.

$$\vdash []\ F \Rightarrow F$$

STL3.

$$\vdash [][]\ F \Leftrightarrow []\ F$$

STL4.

$$\frac{F \Rightarrow G}{[]F \Rightarrow []G}$$

STL5.

$$\vdash []\ (F \wedge G) \Leftrightarrow ([]\ F) \wedge ([]\ G)$$

STL6.

$$\vdash (<>\ []\ F) \wedge (<>[]\ G) \Leftrightarrow <>[]\ (F \wedge G)$$

LATTICE

$$\frac{F \wedge (c \in S) \Rightarrow (H_c \leadsto (G \vee \exists\, d \in S: (c > d) \wedge H_d))}{F \Rightarrow ((\exists\, c \in S: H_c) \leadsto G)}$$

- a well-founded partial order on set S

The Basic Rules of TLA

TLA1.

$$\frac{P \wedge (f = f') \Rightarrow P'}{[]P \Leftrightarrow P \wedge [][P \Rightarrow P']_f}$$

TLA2.

$$\frac{P \wedge \langle A\rangle_f \Rightarrow Q \wedge [A]_g}{[]P \wedge []\langle A\rangle_f \Rightarrow []Q \wedge [][A]_g}$$

ADDITIONAL RULES

INV1.

$$\frac{I \text{ and } [A]_f \Rightarrow I'}{I \text{ and } [][A]_f \Rightarrow []I}$$

INV2.

$$\vdash [] I \Rightarrow ([][A]f \Leftrightarrow [][A \text{ and } I \text{ and } I']_f)$$

WF1.

$$\frac{\begin{array}{l} P \text{ and } [A]_f \Rightarrow (P' \text{ or } Q') \\ P \text{ and } \langle A \text{ and } A\rangle_f \Rightarrow Q' \\ P \Rightarrow \text{Enabled } \langle A\rangle_f \end{array}}{[][A]_f \text{ and } WF_f(A) \Rightarrow (P \mid\!\!\rightarrow Q)}$$

WF2.

$$\frac{\begin{array}{l} \langle A \text{ and } A\rangle_f \Rightarrow \langle A\rangle_g \\ P \text{ and } P' \text{ and } \langle A \text{ and } A\rangle_f \text{ and } \text{Enabled } \langle A\rangle_g \Rightarrow A \\ P \text{ and } \text{Enabled } \langle A\rangle_g \Rightarrow \text{Enabled } \langle A\rangle_f \\ [] \ [A \text{ and not } A]_f \text{ and } WF_f(A) \text{ and } []F \\ \quad \text{and } \langle\rangle[] \text{ Enabled } \langle A\rangle_g \Rightarrow \langle\rangle[] \ P \end{array}}{[] \ [A]_f \text{ and } WF_f(A) \text{ and } [] \ F \Rightarrow WF_g(A)}$$

SF1.

$$\frac{\begin{array}{l} P \text{ and } [A]_f \Rightarrow (P' \text{ or } Q') \\ P \text{ and } \langle A \text{ and } A\rangle_f \Rightarrow Q' \\ [] \ P \text{ and } [] \ [A]_f \text{ and } [] \ F \Rightarrow \langle\rangle \text{ Enabled } \langle A\rangle_f \end{array}}{[] \ [A]_f \text{ and } SF_f(A) \text{ and } [] \ F \Rightarrow (P \mid\!\!\rightarrow Q)}$$

SF2.

$$\frac{\begin{array}{l} \langle A \text{ and } A\rangle_f \Rightarrow \langle A\rangle_g \\ P \text{ and } P' \text{ and } \langle A \text{ and } A\rangle_f \Rightarrow A \\ P \text{ and } \text{Enabled } \langle A\rangle_g \Rightarrow \text{Enabled } \langle A\rangle_f \\ [] \ [A \text{ and not } A]_f \text{ and } SF_f(A) \text{ and } [] \ F \\ \quad \text{and } [] \langle\rangle \text{ Enabled } \langle A\rangle_g \Rightarrow \langle\rangle[] \ P \end{array}}{[] \ [A]_f \text{ and } SF_f(A) \text{ and } [] \ F \Rightarrow SF_g(A)}$$

APPENDIX C: TLA$^+$ BASIC MODULES

MODULE *Peano*

$$
\begin{array}{l}
PeanoAxioms(N,\ Z,\ Sc) \triangleq \\
\quad \land\ Z \in N \\
\quad \land\ Sc \in [N \to N] \\
\quad \land\ \forall\, n \in N : (\exists\, m \in N : n = Sc[m]) \equiv (n \neq Z) \\
\quad \land\ \forall\, S \in \text{SUBSET } N : (Z \in S) \land (\forall\, n \in S : Sc[n] \in S) \Rightarrow (S = N) \\
\\
\text{ASSUME } \exists\, N,\ Z,\ Sc : PeanoAxioms(N,\ Z,\ Sc) \\
\\
Succ \triangleq \text{CHOOSE } Sc : \exists\, N,\ Z : PeanoAxioms(N,\ Z,\ Sc) \\
Nat \triangleq \text{DOMAIN } Succ \\
Zero \triangleq \text{CHOOSE } Z : PeanoAxioms(Nat,\ Z,\ Succ)
\end{array}
$$

* * *

MODULE *ProtoReals*

EXTENDS *Peano*

$$
\begin{array}{l}
IsModelOfReals(R,\ Plus,\ Times,\ Leq) \triangleq \\
\quad \text{LET } IsAbelianGroup(G,\ Id,\ \_ + \_) \triangleq \\
\qquad \land\ Id \in G \\
\qquad \land\ \forall\, a,\ b \in G : a + b \in G \\
\qquad \land\ \forall\, a \in G : Id + a = a \\
\qquad \land\ \forall\, a,\ b,\ c \in G : (a + b) + c = a + (b + c) \\
\qquad \land\ \forall\, a \in G : \exists\, minusa \in G : a + minusa = Id \\
\qquad \land\ \forall\, a,\ b \in G : a + b = b + a
\end{array}
$$

Plus and *Times* are functions and *Leq* is a set, but it's more convenient to turn them into the infix operators $+$, $*$, and $\leq$

$$
\begin{array}{l}
\quad a + b \triangleq Plus[a,\ b] \\
\quad a * b \triangleq Times[a,\ b] \\
\quad a \leq b \triangleq \langle a,\ b \rangle \in Leq \\
\text{IN} \quad \land\ Nat \subseteq R \\
\qquad \land\ \forall\, n \in Nat : Succ[n] = n + Succ[Zero] \\
\qquad \land\ IsAbelianGroup(R,\ Zero,\ +) \\
\qquad \land\ IsAbelianGroup(R \setminus \{Zero\},\ Succ[Zero],\ *) \\
\qquad \land\ \forall\, a,\ b,\ c \in R : a * (b + c) = (a * b) + (a * c) \\
\qquad \land\ \forall\, a,\ b \in R : \land\ (a \leq b) \lor (b \leq a) \\
\qquad\qquad\qquad\qquad \land\ (a \leq b) \land (b \leq a) \equiv (a = b) \\
\qquad \land\ \forall\, a,\ b,\ c \in R : \land\ (a \leq b) \land (b \leq c) \Rightarrow (a \leq c) \\
\qquad\qquad\qquad\qquad \land\ (a \leq b) \Rightarrow \\
\qquad\qquad\qquad\qquad\qquad \land\ (a + c) \leq (b + c) \\
\qquad\qquad\qquad\qquad\qquad \land\ (Zero \leq c) \Rightarrow (a * c) \leq (b * c) \\
\qquad \land\ \forall\, S \in \text{SUBSET } R : \\
\qquad\quad \text{LET } SBound(a) \triangleq \forall\, s \in S : s \leq a \\
\qquad\quad \text{IN} \quad (\exists\, a \in R : SBound(a)) \Rightarrow \\
\qquad\qquad\qquad (\exists\, sup \in R : \land\ SBound(sup) \\
\qquad\qquad\qquad\qquad\qquad\qquad \land\ \forall\, a \in R : SBound(a) \Rightarrow (sup \leq a))
\end{array}
$$

THEOREM $\exists\, R,\ Plus,\ Times,\ Leq : IsModelOfReals(R,\ Plus,\ Times,\ Leq)$

$RM \triangleq \text{CHOOSE } RM : IsModelOfReals(RM.R,\ RM.Plus,\ RM.Times,\ RM.Leq)$

$Real \triangleq RM.R$

$Infinity \triangleq \text{CHOOSE } x : x \notin Real$

$MinusInfinity \triangleq \text{CHOOSE } x : x \notin Real \cup \{Infinity\}$

$a + b \triangleq RM.Plus[a,\ b]$

$a * b \triangleq RM.Times[a,\ b]$

$a \leq b \triangleq \text{CASE } (a \in Real) \land (b \in Real) \rightarrow \langle a,\ b \rangle \in RM.Leq$

$$\begin{array}{l}
\qquad\quad \Box \quad (a = \mathit{Infinity}) \wedge (b \in \mathit{Real} \cup \{\mathit{MinusInfinity}\}) \ \to \text{FALSE} \\
\qquad\quad \Box \quad (a \in \mathit{Real} \ \cup \{\mathit{MinusInfinity}\}) \wedge (b = \mathit{Infinity}) \to \text{TRUE} \\
\qquad\quad \Box \quad (a = b) \to \text{TRUE}
\end{array}$$

$$\begin{array}{l}
a - b \ \triangleq \ \text{CASE} \ (a \in \mathit{Real}) \wedge (b \in \mathit{Real}) \ \to \text{CHOOSE} \ c \in \mathit{Real} : c + b = a \\
\qquad\quad \Box \quad (a \in \mathit{Real}) \wedge (b = \mathit{Infinity}) \qquad\quad \to \mathit{MinusInfinity} \\
\qquad\quad \Box \quad (a \in \mathit{Real}) \wedge (b = \mathit{MinusInfinity}) \to \mathit{Infinity}
\end{array}$$

$$a/b \ \triangleq \ \text{CHOOSE} \ c \in \mathit{Real} : a = b * c$$

$$\mathit{Int} \ \triangleq \ \mathit{Nat} \cup \{\mathit{Zero} - n \ : n \in \mathit{Nat}\}$$

$$\begin{array}{l}
a^b \ \triangleq \\
\quad \text{LET} \ \mathit{RPos} \ \triangleq \ \{r \in \mathit{Real} \setminus \{\mathit{Zero}\} : \mathit{Zero} \leq r\} \\
\qquad \mathit{exp} \ \triangleq \ \text{CHOOSE} \ f \in [(\mathit{RPos} \times \mathit{Real}) \cup (\mathit{Real} \times \mathit{RPos}) \\
\qquad\qquad\qquad\qquad\qquad\qquad \cup ((\mathit{Real} \setminus \{\mathit{Zero}\}) \times \mathit{Int}) \to \mathit{Real}] : \\
\qquad\qquad \wedge \forall r \quad \in \mathit{Real} : \\
\qquad\qquad\qquad \wedge f[r, \mathit{Succ}[\mathit{Zero}]] = r \\
\qquad\qquad\qquad \wedge \forall m, n \in \mathit{Int} : (r \neq \mathit{Zero}) \Rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad (f[r, m + n] = f[r, m] * f[r, n]) \\
\qquad\qquad \wedge \forall r \in \mathit{RPos} : \\
\qquad\qquad\qquad \wedge f[\mathit{Zero}, r] = \mathit{Zero} \\
\qquad\qquad\qquad \wedge \forall s, t \in \mathit{Real} \ : f[r, s * t] = f[f[r, s], t] \\
\qquad\qquad\qquad \wedge \forall s, t \in \mathit{RPos} : (s \leq t) \Rightarrow (f[r, s] \leq f[r, t]) \\
\quad \text{IN} \quad \mathit{exp}[a, b]
\end{array}$$

---

* * *

MODULE *Naturals*

$$\begin{array}{l}
\text{LOCAL} \ R \ \triangleq \ \text{INSTANCE} \ \mathit{ProtoReals} \\
\\
\mathit{Nat} \ \triangleq \ R!\mathit{Nat} \\
\\
a + b \ \triangleq \ aR! + b \\
a - b \ \triangleq \ aR! - b \\
a * b \ \triangleq \ aR! * b \\
a^b \quad \triangleq \ aR!^b \\
a \leq b \ \triangleq \ aR! \leq b \\
a \geq b \ \triangleq \ b \leq a \\
a < b \ \triangleq \ (a \leq b) \wedge (a \neq b) \\
a > b \ \triangleq \ b < a \\
a \mathinner{..} b \ \triangleq \ \{i \in R!\mathit{Int} : (a \leq i) \wedge (i \leq b)\} \\
a \div b \ \triangleq \ \text{CHOOSE} \ n \in R!\mathit{Int} : \exists r \in 0 \mathinner{..} (b - 1) : a = \ b * n + r \\
a\%b \ \triangleq \ \ a \ - \ b * (a \div b)
\end{array}$$

MODULE *Sequences*

Defines operators on finite sequences, where a sequence of length n is represented as a function whose domain is the set $1 \,..\, n$ (the set $\{1, 2, \ldots, n\}$). This is also how TLA+ defines an n-tuple, so tuples are sequences.

LOCAL INSTANCE *Naturals*

Imports the definitions from *Naturals*, but don't export them.

$Seq(S) \triangleq \text{UNION } \{[1 \,..\, n \to S] : n \in Nat\}$

The set of all sequences of elements in S.

$Len(s) \triangleq \text{CHOOSE } n \in Nat : \text{DOMAIN } s = 1 \,..\, n$

The length of sequence s.

$s \circ t \triangleq [i \in 1 \,..\, (Len(s) + Len(t)) \mapsto \text{IF } i \leq Len(s) \text{ THEN } s[i] \text{ ELSE } t[i - Len(s)]]$

The sequence obtained by concatenating sequences s and t.

$Append(s, e) \triangleq s \circ \langle e \rangle$

The sequence obtained by appending element e to the end of sequence s.

$Head(s) \triangleq s[1]$
$Tail(s) \triangleq [i \in 1 \,..\, (Len(s) - 1) \mapsto s[i + 1]]$

The usual head (first) and tail (rest) operators.

$SubSeq(s, m, n) \triangleq [i \in 1 \,..\, (1 + n - m) \mapsto s[i + m - 1]]$

The sequence $\langle s[m], s[m + 1], \ldots, s[n] \rangle$.

$SelectSeq(s, Test(\_)) \triangleq$

The subsequence of s consisting of all elements $s[i]$ such that $Test(s[i])$ is true.

LET $F[i \in 0 \,..\, Len(s)] \triangleq$

$F[i]$ equals $SelectSeq(SubSeq(s, 1, i), Test]$

IF $i = 0$ THEN $\langle\rangle$
ELSE IF $Test(s[i])$ THEN $Append(F[i - 1], s[i])$
ELSE $F[i - 1]$

IN $F[Len(s)]$

MODULE *InnerFIFOChannelWLI*

EXTENDS *Naturals*, *Sequences*

VARIABLES *shuttle*, *message*

VARIABLES *in*, *out*, *q*

$InChan \triangleq$ INSTANCE *ChannelWLI* WITH *shuttle* $\leftarrow$ *message*, *chan* $\leftarrow$ *in*

$OutChan \triangleq$ INSTANCE *ChannelWLI* WITH *Data* $\leftarrow$ *message*, *chan* $\leftarrow$ *out*

$$Init \triangleq \land InChan!Init \land OutChan!Init \land q = \langle\rangle$$

$$TypeInvariant \triangleq \land InChan!TypeInvariant \land OutChan!TypeInvariant \land q \in Seq(message)$$

$SSend(msg) \triangleq \land InChan!Send(msg) \land$ UNCHANGED $\langle out, q\rangle$ Send *msg* on channel *in* .

$BufRcv \triangleq \land InChan!Rcv \land q' = Append(q, in.val) \land$ UNCHANGED *out* Receive message from channel *in* . and append it to tail of *q*.

$BufSend \triangleq \land q \neq \langle\rangle \land OutChan!Send(Head(q)) \land q' = Tail(q) \land$ UNCHANGED *in* Enabled only if *q* is nonempty. Send *Head*(*q*) on channel *out* and remove it from *q*.

$RRcv \triangleq \land OutChan!Rcv \land$ UNCHANGED $\langle in, q\rangle$ Receive message from channel *out* .

$$Next \triangleq \lor \exists\, msg \in message : SSend(msg) \lor BufRcv \lor BufSend \lor RRcv$$

$$Spec \triangleq Init \land \Box[Next]_{\langle in,\, out,\, q\rangle}$$

THEOREM $Spec \Rightarrow \Box TypeInvariant$

APPENDIX D: AUTOPOIETIC THEORY – DEFINITIONS

For the time being, computer networks and telecommunications systems have been considered as pure state-automata environments where the human-machine interaction was playing a rather phenomenological role by being placed under control of rigorous designer constraints.

Autopoietic theory provides a solemn theoretical basis for addressing people and the social systems in which they participate. This includes not only the interactions of professional, political, cultural and ethnic groups and organizations, but also the means they communicate, such as language, telephones, television and the Internet.

The following arguments support the above conclusion:

- because the autopoietic theory proceeds from formal specifications on systemic unities, its tenets can conceivably be applied to both, biological and artificial systems. Owing to the extent of Maturana and Varela's expansion of the core concepts to describe a phenomenology of living systems, the theory's scope is relatively broad. This permits researchers to apply its principles across a broader range of subject phenomena than is the case for other current approaches.

- because the autopoietic theory is rooted in a formal analysis of living systems and cognition, it can support research focusing e.g. on individual subjects and their activities within an enterprise or an operator's network such as workflow, human factors and human-computer interface (HCI) analyses of specific information system users.

- because the autopoietic theory includes an explanation for linguistic interaction, it can support research focusing on social interactions and communications (e.g., ethnographic studies; qualitative research) which finally have implications on network traffic characteristics.

The autopoietic theory intrinsically supports attention to the three themes in today's social research innovations: systemic perspective, auto-determination, and contextualization. The first occurs by definition, the second by focus, and the third by the manner in which Maturana and Varela lay out the phenomenological aspects of the theory.

The WLI approach presented in this work addresses the technological aspects of the autopoietic theory in the design of new generations of telecommunication architectures.

The definitions below are compiled from "Overview of Autopoietic Theory", http://www.enolagaia.com.

THE OBSERVER

"Everything said is said by an observer." ([MaVa80], p. xix)

Maturana's initial work on cognition emphasized individual living systems. As a result, autopoietic theory has as its foundation the manner in which living systems address and engage the domain(s) in which they operate. This orientation subsumes the manner in which autopoietic theory addresses itself (as a scientific theory) and all other phenomena. A cognizing system engages the 'world' only in terms of the perturbations in its nervous system, which is *'operationally closed'* (i.e., its transformations occur within its bounds). To the extent that the nervous system recursively interconnects its components (as in our brains), the organism is capable of generating, maintaining and re-engaging its own states as if they were literal re-presentations of external phenomena. Such states are 'second-order' in the sense that they are derivative from, rather than literal recordings of, experience. These states are called *descriptions* in autopoietic theory, and an organism operating within the realm of its descriptions is an *observer*.

FUNDAMENTAL SYSTEM ATTRIBUTES: ORGANIZATION AND STRUCTURE

Systems cannot be defined by simply enumerating or tracing the layout of their constituent elements. The definitive attribute of a systemic entity is the set of inter-component relationships which (a) outline its form at any given moment and (b) serve as the core 'identity' which is maintained in spite of dynamic changes over time. In autopoietic theory, this set of defining relationships is termed a system's *organization*.

In effect, a system's organization specifies a category, within which there may be many specifically-realized instantiations. Specific systemic entities exhibit more than just the general pattern of their organization -- they consist of particular components and relations among them. The 'particulars' of a given system's individual realization make up its *structure*.

Maturana and Varela's complementary distinction between organization and structure is very useful in delineating and analyzing systems' form and function. This aspect of autopoietic theory makes it useful in describing enterprises as having generally invariant form in spite of specifically changing components.

AUTOPOIESIS AND AUTONOMY

Autopoietic systems realized in the physical space are living systems. Varela later defined a broader concept of *autonomy*, of which autopoiesis is a special case. Autonomous systems maintain their organization, but do not necessarily regenerate their own components.

Autopoiesis

Maturana and Varela coined the term *autopoiesis* from the Greek *auto* (self-) and *poiesis* (creation; production), [MaVa80], to characterize those systems which (a) maintain their defining organization throughout a history of environmental perturbation and structural change, and (b) regenerate their components in the course of their operation, [MaVa80].

The concept is defined formally as follows, [Var79]:

'An autopoietic system is organized (defined as a unity) as a network of processes of production (transformation and destruction) of components, that produces the components which:

1. through their interactions and transformations continuously regenerate and realize the network of processes (relations) that produced them; and

2. constitute it (the machine) as a concrete unity in the space in which they [the components] exist by specifying the topological domain of its realization as such a network.'

Any unity meeting these specifications is an autopoietic system, and any such autopoietic system realized in physical space is a living system. The particular configuration of a given unity -- its *structure* -- is not sufficient to define it as a unity. The key feature of a living system is maintenance of its *organization*, i.e., preservation of the relational network which defines it as a systemic unity. Phrased another way, '...autopoietic systems operate as homeostatic systems that have their own organization as the critical fundamental variable that they actively maintain constant.' ([Mat75], p. 318)

Autopoietic theory is the primary (perhaps the only...) example of a definition for life which is framed purely with respect to a candidate system in and of itself. If you go back and check most definitions (e.g., in a biology text), you are likely to find nothing more coherent than a list of features and functional attributes (e.g., 'reproduction', 'metabolism') which describe what living systems do, but not what they are. For this reason, autopoiesis has become a topic of interest in the recent field of *Artificial Life* (*ALife*).

Autonomy

During the mid- to late 1970's, Varela expanded on autopoietic theory's original formalizations to delineate the systemic attribute of *autonomy*, of which autopoiesis is a subset. Autonomous systems are ([Var81a], p.15):

> '...defined as a composite unity by a network of interactions of components that (i) through their interactions recursively regenerate the network of interactions that produced them, and (ii) realize the network as a unity in the space in which the components exist by constituting and specifying the unity's boundaries as a cleavage from the background...'

The difference between autonomy and autopoiesis is that autopoietic systems must produce their own components in addition to conserving their organization. As we shall see later, this difference has played a large role in the debates over the extent to which social systems can be characterized as autopoietic.

This more general class of autonomous systems are defined by their *organizational closure***,** ([Var79], p. 55):

> 'That is, their organization is characterized by processes such that
> - the processes are related as a network, so that they recursively depend on each other in the generation and realization of the processes themselves, and
> - they constitute the system as a unity recognizable in the space (domain) in which the processes exist.'

It is important to note that this property of 'closure' does not make autonomous systems 'closed' in the classic cybernetic sense of 'isolated from the environment; impervious to environmental influence'. 'Closure' doesn't mean autonomous systems are unresponsive; it only means that their changes of state in response to changes in their medium are realized and propagated solely within the network of processes constituting them (as they are defined). The difference has more to do with the way a system is defined than how that system (once defined) operates. A fuller explanation of this point can be obtained in [Var79].

DOMAINS AND SPACES

A key concept in Maturana and Varela's writings is *domain*. They use the term generally to connote a 'realm' or 'sphere' circumscribing: (1) the relations among observed systems and the unities (medium) with which they can be observed to engage (e.g., phenomenological domain) or (2) the foregoing plus all potential states of relation and/or activity among the given unities (e.g., domain of interactions). Maturana and Varela reserve the term *space* for the static context in which unities are delineated.

STRUCTURAL DETERMINATION

Structural determination is the principle that the actual course of change in a systemic entity is controlled by its structure (the totality of specific components' individual and synergistic properties within the arrangement by which they constitute the system) rather than direct influence of its environment.

The basic thrust of the principle of structural determination is that the behaviour of a system is constrained by its constitution. The set of potential system changes is circumscribed by:

- the system's range of potential structural transformations
- the set of potential perturbations impinging upon the system

Actual change is compensable behavior by the system's structure under perturbation by the environment and / or other systems in the course of its operation (‘structural coupling', defined below). While a given perturbation may 'trigger' a change of system state, the particular change triggered is a function of the system's own organization and structure. Since 'structure' refers to any constitutive element of a discerned unity, structural determination concerns the manner in which observed (-able) phenomena are explained, not some formalized manner in which those phenomena objectively occur. As such, structural determination is an epistemological qualification, not recourse to materialistic reductionism.

Structural determination should not be equated with strict causal determinism, in which all specific interactions are predetermined. It only means the space of all possible classes of interactions is determined. For example, in re-engineering an enterprise, the subject's structure does not uniquely predict its best new form. However, its structure circumscribes the range of new forms into which it can evolve without violating its organization (i.e., ceasing to exist as its current identity). Structural determination does not constrain the set of interactions in which a system can be observed to engage -- only the set in which that system can observe itself to be engaged.

STRUCTURAL COUPLING

Given the principle of structural determination, interaction among systems is explained as '...a history of recurrent interactions leading to the structural congruence between two (or more) systems' ([MaVa87], p. 75). *Structural coupling* is the label for ongoing engagement between systems, resulting in structural changes in each. Structural coupling describes ongoing mutual co-adaptation without allusion to a transfer of some ephemeral force or information across the boundaries of the engaged systems.

The notions of 'structural determination' and 'structural coupling' provide a basis for analyzing enterprises and their operations in terms of their general and actual form (i.e., their organization and structure). This approach maintains a focus on the subject enterprise and minimizes counterproductive bias toward a priori allusions to abstractions such as 'information flows', 'market forces', and the like.

Structural coupling is the term for structure-determined (and structure-determining) engagement of a given unity with either its environment or another unity. It is '...a historical process leading to the spatio-temporal coincidence between the changes of state....’ ([Mat75], p. 321) in the participants. As such, structural coupling has connotations of both coordination and co-evolution.

Case 1: A System Coupling with its Environment

> 'If one of the plastic systems is an organism and the other its medium, the result is ontogenic adaptation of the organism to its medium: the changes of state of the organism correspond to the change of state of the medium.'
>
> ([Mat75], p. 326)

> ‘(T)he continued interactions of a structurally plastic system in an environment with recurrent perturbations will produce a continual selection of the system's structure. This structure will determine, on the one hand, the state of the system and its domain of allowable perturbations, and on the other hand will allow the system to operate in an environment without disintegration. '
>
> ([Var79], p. 33)

Case 2: A System Coupling with Another System

> 'If the two plastic systems are organisms, the result of the ontogenic structural coupling is a **consensual domain**.'
>
> ([Mat75], p. 326)

A consensual domain is defined as '... a domain of interlocked (intercalated and mutually triggering) sequences of states, established and determined through ontogenic interactions between structurally plastic state-determined systems.' ([Mat75], p. 316). Because consensual domains are defined both by the structures of their participants and the history by which they came to exist, they are not reducible to descriptions framed only in terms of either:

> 'In each interaction the conduct of each organism is constitutively independent in its generation of the conduct of the other, because it is internally determined by the structure of the behaving organism only; but it is for the other organism, while the chain [of interactions] lasts, a source of compensable deformations that can be described as meaningful in the context of the coupled behaviour.' ([Var79], pp. 48- 49)

Phrased in a slightly different way, the participating systems reciprocally serve as sources of *compensable perturbations* for each other. Such interactions are 'perturbations' in the sense of indirect effect or effectuation of change without having penetrated the boundary of the affected system. They are 'compensable' in the senses that (a) there is a range of 'compensation' bounded by the limit beyond which each system ceases to be a functional whole and (b) each iteration of the reciprocal interaction is affected by the one(s) before. The structurally-coupled systems 'will have an interlocked history of structural transformations, selecting each other's trajectories.' ([Var79], pp. 48 - 49)

COGNITION AS (INTER-)ACTIVITY

The attribute *'cognition'* is applied to a system when it is able to discriminate (in terms of response) among unit phenomena in its medium, synchronically (at a given moment) and diachronically (over time). The currently-prevalent cognitivistic viewpoint addresses the capacity for such discrimination in terms of algorithmic procedures for manipulating abstracted 'data' with respect to 'knowledge structures'. To Maturana and Varela, cognition is contingent on embodiment, because this ability to differentiate is a consequence of the organism's specific structure. From their perspective, cognition is what we attribute to systems exhibiting flexible and effective changes during *structural coupling*.

Cognition in the autopoietic view is no more and no less than a living system's effective behaviour within its domain of interactions. In other words, cognition is a matter of interacting in the manner(s) in which one is capable of interacting, not processing, what is objectively there to be seen.

> 'A cognitive system is a system whose organization defines a domain of interactions in which it can act with relevance to the maintenance of itself, and the process of cognition is the actual (inductive) acting or behaving in this domain.'
>
> 'Living systems are cognitive systems, and living as a process is a process of cognition.'
>
> ([MaVa80], p. 13)