Florian Liers

Forwarding on Gates
A flexible and scalable inter-network layer supporting in-network functions

Forwarding on Gates

A flexible and scalable inter-network layer supporting in-network functions

Florian Liers

Universitätsverlag Ilmenau
2014

Impressum

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Angaben sind im Internet über http://dnb.d-nb.de abrufbar.

Diese Arbeit hat der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau als Dissertation vorgelegen.

Tag der Einreichung:	1. Juli 2013
1. Gutachter:	Prof. Dr.-Ing. habil. Andreas Mitschele-Thiel (Technische Universität Ilmenau)
2. Gutachter:	Prof. Dr. rer. nat. Paul Müller (Technische Universität Kaiserslautern)
3. Gutachter:	PD Dr. rer. nat. Oliver P. Waldhorst (Karlsruher Institut für Technologie)
Tag der Verteidigung:	16. Dezember 2013

Technische Universität Ilmenau/Universitätsbibliothek
Universitätsverlag Ilmenau
Postfach 10 05 65
98684 Ilmenau
www.tu-ilmenau.de/universitaetsverlag

Herstellung und Auslieferung
Verlagshaus Monsenstein und Vannerdat OHG
Am Hawerkamp 31
48155 Münster
www.mv-verlag.de

ISBN 978-3-86360-094-5 (Druckausgabe)
URN urn:nbn:de:gbv:ilm1-2013000657

Coverfoto: photocase.com

Ohne euch würde es das Schiff nicht geben.
Ohne euch wäre es nie seetüchtig geworden.
Ohne euch hätte es nie Segel gesetzt.
Ohne euch hätte es keine Karte gehabt.
Ohne euch hätte es nicht die Gnade einer Wahl gehabt.

Und auch wenn das Schiff in der Welt unterwegs ist,
erinnert es sich doch immer an den schönsten aller Ankerpunkte.

Ohne dich wäre es auf Riffe gelaufen.
Ohne dich wäre es gestrandet.
Ohne dich hätte es nicht seine Anker gelichtet.
Ohne dich hätte es nicht alle Segel gesetzt.
Ohne dich hätte es vertraute Gewässer nicht verlassen.

Und so ist es gemeinsam mit dir auf neuem Kurs,
der Zukunft entgegen.

Ich widme diese Arbeit meinen Eltern
und meiner Lebensgefährtin Susanne.

Acknowledgements

Although I am the only author of this book, a lot of people contributed to it and supported me during my time at the Technische Universität Ilmenau. The order of the following acknowledgments does not express a judgment about their importance.

Many thanks go to the supervisor of my dissertation Prof. Dr.-Ing. Andreas Mitschele-Thiel. He trusted in my attitude to scientific work and allowed me to do research in unexpected directions. This academic freedom was essential for my time at the Integrated Communication Systems group. Thank you also for the constructive feedback to my work and for simple but challenging questions.

I like to thank Prof. Dr. Paul Müller and PD Dr. Oliver Waldhorst for taking over the review of my thesis. I hope you enjoyed the reading of my thesis, although it is a bit longer than promised.

I would like to thank my colleague Thomas Volkert for the constructive cooperation. Together, we invented our baby FoG and accomplished it over nearly five years. It started in late summer 2008 during a discussion between Thomas and me. We had been sitting outside in the sun in front of the Humboldt building arguing about the missing innovations in networking research. Thank you also for the name "FoG".

Thanks to my colleagues for the inspiring time! In particular, I would like to thank my colleague Mohamed Kalil for the open and relaxed atmosphere in our office in the old Blechhaus.

Daniel Lorenz and Tim Langner read early versions of the thesis. Thank you for working through the text, although you did not know anything about the topic. The remaining errors are all my fault.

I would like to thank Florian Evers as well for reviewing parts of the thesis – some parts even multiple times. Thank you, Florian, also for introducing me to the most addictive hobby. "Searching" is not so far away from research.

A lot of students and colleagues contributed to the project by writing code or conducting conceptual work. I acted as supervisor for the work done by students and as team leader for the colleagues. The following persons contributed to my thesis (in alphabetic order):

- Martin Bengsch with a Diploma thesis about the connection setup in FoG
- Markus Brückner as colleague during the last six month of the project with debugging the rerouting code

- Sebastian Dietzel with a Diploma thesis about a mobility solution comparable to MobileIP
- Daniel Hundsdörfer as student assistant for emulator issues
- Robert Kaltenhäuser with a project seminar about rerouting experiments
- Sebastian Messing with a Diploma thesis about rerouting in networks with explicit routes
- Manuel Osdoba with a project seminar and as student assistant in the context of rerouting experiments
- Udo Peschek with a Studienarbeit about transport protocols
- Daniel Renner with a Bachelor thesis about extensions to the algorithm mapping requirements to chains
- Thomas Volkert with a focus on hierarchical routing management, emulation, interoperability and video transmission. During countless discussions, he contributed to the definition of the FoG layer architecture in its non-recursive form and its packet format as well.

Finally, I would like to thank some people that are more familiar with German.

Vielen Dank auch an die Mensa des Studentenwerkes der TU Ilmenau. Die Auswahl zwischen den Essen war nicht immer einfach; aber ohne sie wäre ich verhungert.

Weiterhin möchte ich meiner Familie für ihre geistige Unterstützung über die lange Zeit danken. Vielen Dank für euer Verständnis, wenn andere Dinge wegen der Dissertation liegen blieben.

Schlussendlich möchte ich meiner Lebensgefährtin Susanne für noch mehr Verständnis für falsche Prioritäten danken. Du gabst mir die Motivation zurück. Und dank deiner Unterstützung ist wider Erwarten doch noch alles fertig geworden.

Contents

1. Introduction

The location of functions is a central theme of computer networks. In the past, unreliable telephone links had to be enhanced with forward error correction and retransmission functions. Since networks had used such links as backbone links, the functions had resided on relay systems in the network. Later, new transmission techniques, like fiber optics, with a much better reliability have been introduced. Their reduced bit error rate influenced the functions required in networks. In such networks, packet loss is mainly caused by congestions rather than transmission errors. Thus, a design with retransmission functions at end systems (or "hosts") rather than within networks became more efficient. With the advent of less reliable wireless links, a need for functions coping with loss or bit errors on these links arises. Solutions such as special transport protocols for wireless links [SS06] basically re-introduce this functionality. This time, however, the function is placed mainly on both sides of the last hop. Other functions are placed at different locations as well. For example, packets of a multicast transmission can be duplicated by relay systems within networks or at end systems by using a multicast overlay.

History shows that there is a tendency to add new functions which are unforeseen by the original design of a communication network in order to support new use cases. The Internet, which started out as a basic "dump" packet forwarding network, was gradually enhanced with various functions. In order to support mobile devices, protocols, like MobileIP [Per10], and special "anchor point" functions on relay systems have been added to the Internet. Other features such as network address translation and firewalls require additional functions within the Internet as well.

The efficiency of some existing features can be improved by adding functions to networks. Examples are the functions mentioned above that reduce bit error rates and that reduce the network load induced by multicast transmissions. Caching of content is another example, which reduces delivery time and network load. Even the non-functional characteristics of packet forwarding – also known as *Quality of Service* (QoS) characteristics – can be influenced by functions. A function can, for example, prioritize packets or relay them with a minimal data rate over a highly utilized link.

The requirements of applications for their data transmissions and the situation of the network influence the amount of required functions. For simple web-browsing and a low loaded Internet with abundant capacity, no additional functions on relay systems are required. However, with increasing load and shrinking free capacities more functions within the network can improve the performance. For example, caches can be introduced to handle web page requests more efficiently. Alternatively or in addition, packets that transport content of web pages or requests for them can be marked and handled with a higher priority than, for example, packets of a file download. The same trade-off applies for live video streaming, but with slightly different functions required in the network. Instead of assigning priorities, resources may have to be reserved to ensure a steady stream of packets in situations with high load.

The situation is even more complex, since today's Internet is a network of subnetworks, called inter-network. Each subnetwork has an operator deciding independently about the equipment, policies, and protocols used within its subnetwork. For example, one subnetwork may use the *Universal Mobile Telecommunications System* (UMTS) access technology in order to connect mobile users to the Internet. Another subnetwork favors Ethernet to connect stationary computers. The inter-network has to enable the interoperability between various types of end systems connected to heterogeneous subnetworks. It has to balance constraints required for the interoperability with the autonomy of each operator to decide about the functions provided by its subnetworks. Thus, the previously mentioned decision of how many functions a network provides has to be made for each subnetwork individually. A UMTS subnetwork may, for example, introduce resource reservation functions for video streaming due to its limited capacity. An Ethernet subnetwork may rely on over-provisioning and does not require such additional functions. For a transmission between end systems from both subnetworks, however, the networks have to interoperate. This may even require the support from relaying (or transit) subnetworks, if the subnetworks are not directly connected to each other.

The provisioning of functions comes along with some cost in terms of memory, throughput, and *Central Processing Unit* (CPU) load. Typically, an instance of a function requires memory to store its current state, buffers, timers, and management information. In order to perform its operations, CPU time is required. Moreover, a system has to know which packets have to be processed by which functions. It has to classify packets according to some pattern and to relay packets of different classes to different functions. For example, *Integrated Service* (IntServ), which is an extension to the *Internet Protocol* (IP), uses the source and destination IP addresses and the protocol field value of a packet as pattern to identify the flow the packet belongs to and, thus, identify the function that has to be performed on the packet. These so called mapping states have to be stored and – even more critical – have to be searched for each packet arriving at a node. If the search algorithm is assumed to be fixed, the amount of mapping states directly influences the delay of a packet. A scalability problem arises: The more states a node has to store, the longer the delay, and the worse the system performance. Besides these cost of functions being in use, cost occur for setting up and maintain them. Typically, the creation and removal of functions requires communication between peers. These messages regarding the maintenance are called signaling messages. They consume transport capacity and increase the CPU load of the systems. Thus, the more functions are maintained, the more signaling messages have to be exchanged, and the higher the load of a network.

In summary, networks should be flexible in order to accommodate a broad variety of functions in various and changing locations. Moreover they have to be scalable in order to be applicable for large-scale setups such as the Internet.

1.1. Research question

Today's networks, however, are far from this goal. They are either flexible or scalable.

They are rather static regarding their set of functions and regarding the possible locations of these functions. Since the Internet design makes it hard to support functions within a network, various problems for newly added functions arise. For example, network address translation and firewalls hamper the direct communication between IP nodes. The combination of functions such as QoS, mobility, and security may lead to undesired feature interactions. The more functions are added to an IP network the more severe the scalability problems seems to be. QoS-add-ons exemplify this: IntServ supports a large set of functions but comes along with a high number of states, which are required on every node along a path. *Differentiated Service* (DiffServ), in contrast, requires only a small amount of states but limits the functionality.

The static nature of networks is not limited to IP. The traditional layered reference models such as the *Open System Interconnection* (OSI) model are part of the problem. They propagate a static placement of functions by defining the set of functions supported by a layer and the arrangement of layers. This hampers the introduction of functions not included in the initial design. Moreover, it limits the placement of functions. For example, end-to-end flow control functions are assigned to the transport layer and, thus, are located on end systems. It is hard for functions on relay systems, which belong to the network layer, to assist directly.[1]

This thesis answers the question of **how to provide arbitrary functions in inter-networks in a scalable and flexible way**.

I approached this question from an architectural standpoint by investigating *invariances* –similarities – of existing solutions. The applicability of this method for network research is discussed in [ABE$^+$04]. Specific invariants are exploited and generalized to a more abstract solution applicable to a larger set of prob-

[1] Since – in theory – they are not aware of the transport connections, they can just drop packets.

lems. The feasibility of my contributions is shown with an implementation. Their performance is studied with simulations.

1.2. Scientific contributions

This book describes the following innovative scientific contributions:

Communication model for combining arbitrary functions: I propose a communication model based purely on functional blocks in order to support arbitrary functions. The model differs from other solutions in two aspects. First, functional blocks represent all kinds of functions residing on end and relay systems as well as links in networks. This allows the integration of new functions without being limited to specific locations for them. Second, it aligns functions on end and relay systems and, thus, merges the mechanism for selecting and composing functional blocks with classical routing. The merge increases the flexibility, enables function reuse, and opens the possibility for algorithms to handle the underlying location-routing problem with all its challenges.

Flexible state placement for scalability: I propose to separate the provisioning of functions from the decision for what the functions are used for. This divides states between the provider of a function and the user of a function, which leads to a flexible placement of states in a network. This flexibility can be used by networks for a scalable function provisioning. The separation complements the traditional method of aggregating states in an orthogonal way and is inspired from schemes like DiffServ and source routing that separate the provisioning of relaying functions from the decision which relaying functions are used for which packets. My proposal represents a more general solution that goes beyond the existing ones by being applicable for larger scopes with independent operators and arbitrary functions.

Layer architecture supporting both of the above proposals: I identified the impact of the two previous proposals – communication model and state placement – on architectures. Since existing architectures do not consider these influences, a new architecture called *Forwarding on Gates* (FoG) has been designed. It is an architecture for a single layer that belongs to a reference model, which is layered according to scope and not according to functionality such as the traditional OSI reference model. The FoG layer comprises several protocols and performs operations that are comparable to the OSI network and transport layer. Its core is a protocol that supports a new hybrid relaying approach by combining explicit and hop-by-hop routes.

"Edge-based" view on networking: From a theoretical standpoint, I propose an "edge-based" middle ground between the node-based view of connectionless networks using hop-by-hop forwarding and the route-based view of connection-oriented networks. The former focuses on routes defined by sequences of nodes. The latter requires sequences of links between end systems. I introduce a classification of relaying approaches according to their protocol format, which indicates that most protocols are designed for one or the other type of network. In contrast, FoG enables a combination of both, which opens new opportunities for an interworking between relaying and routing.

1.3. Research environment

During my research career, I witnessed the rise of a research environment for a Future Internet. In the US, the NSF supported the first GENI/FIND "spiral" in 2008. The European Commission funded around 150 research projects[2] related to Future Internet during the Framework Programme 7, which started in 2007. An overview about these activities is given in [PPJ11a]. Beginning in October 2008, the German ministry for research and education (BMBF) funded the first phase of the German Lab (G-Lab)[3] Future Internet initiative. Together with my colleague Thomas Volkert, I wrote a project proposal for the second phase that described our idea of how a Future Internet might look like. In September 2009, the project called "G-Lab_FoG" (Project number 16BK0935) got accepted even so it was conducted by a single university without industry partners. Thus, I had the pleasure to work on a self-designed project for more than three years, a project allowing me to execute my research on FoG.

The project was embedded in the G-Lab by regular meetings with the other projects of the second phase and with the partners from the first phase. I had the pleasure to act as speaker of one *Special Interest Group* (SIG) of G-Lab, called "Functional composition". Its focus was on interfaces between applications and network stacks, languages for defining requirements, and use cases for a Future Internet. The discussions in the SIG resulted in the definition of an interface between applications and network stacks [LVM$^+$11] and influenced the definition of the FoG layer interface.

1.4. Restrictions on the scope

Future Internet research is a wide field and includes research from boosting the throughput for physical transmission to new applications using it. Interested readers are referred to a survey written by Paul et al. [PPJ11b] that provides a comprehensive overview. This book focuses on the technical issues related to protocols for inter-networks. My results do not enhance physical data transmissions, transport protocols, routing algorithms, service level

[2] `http://www.future-internet.eu/home/future-internet-assembly.html`
[3] `http://www.german-lab.de/phase-1/`

agreements, requirement definitions, quality of service enforcement, public-key-infrastructures, network test-beds, and signature generation. Furthermore, this book does not introduce new ("killer") applications for networks.

Moreover, the economic usage and social impact of the Internet are part of the Future Internet research area. I assume that functions within the network are useful elements for a more efficient service provisioning by networks. The political aspects of the assumption are discussed in Section 3.5. However, my book does not contribute to the research fields of market analysis, game theory, and insights for regulators. Interested readers are deferred to [DSW09].

Although FoG operates with functional blocks, the functions themselves are not the focus of this book.

The FoG architecture opens new possibilities for algorithms. Routing algorithms in particular profit from the greater flexibility. However, new algorithms or protocols exploiting this flexibility are not part of this book. Interested readers are deferred to the work on routing of my colleague Thomas Volkert [VMT12, VOBMT13]. Other algorithms, such as the algorithm that maps application requirements to functional blocks, prove the feasibility of some algorithmic aspects of the architecture. However, they might not be optimal and, thus, are only starting points for future work.

This is not the solution your are searching for...

At the PIMRC 2007, a speaker was asking the audience about innovations in Future Internet research. The gist of what he said was: "Circuit switching was first. The Internet brought us packet switching. What is the next step? [dramatic pause] And cell switching is not the answer!" No one in the audience had an answer. Unfortunately, I do not provide an answer either. Since there is no radically new traffic pattern, such as the burst traffic pattern leading to packet switching (cp. Section 2.2.2.1), I stick with packet switching.

1.5. Chapter overview

After the introduction, Chapter 2 sets up the basics for this book. It defines the terms and introduces the reference layer model for FoG. Afterwards, the basics and the state of the art in the research field are introduced.

Thereafter, Chapter 3 describes the new FoG architecture. It starts with a motivating use case in Section 3.1, which shows the limitations of today's solutions. Based on these limitations, design ideas for my solution are derived. In order to support arbitrary functions, Section 3.2 introduces the communication model comprising functional blocks and their dependencies. Section 3.3 describes the architecture defining how a network layer using this model has to be structured. Each of these sections includes a review of the related research work. Section 3.4 illustrates the theoretical description with examples. Afterwards, the features of the architecture are reviewed in the context of the political environment in Section 3.5. Section 3.6 discusses deployment and interoperability aspects. Finally, a summary outlines the differences between FoG and the recursive reference model.

Chapter 4 describes the design of a sample implementation of the FoG architecture. The use cases for the implementation are specified in more detail in Section 4.1. The software architecture supporting them is shown thereafter. Starting with Section 4.3, the algorithms and protocols required for implementing the architecture are described. Most prominent is the new network protocol presented in Section 4.3.2 and the algorithm used to map requirements to functions and functional blocks discussed in Section 4.4.3.

Chapter 5 contains the results from several performance studies executed with the software presented in Chapter 4. The quantitative results complement the qualitative arguments from Chapter 3. Each study takes advantage of a feature of the implementation that is enabled by the architecture. The most important study is described in Section 5.2. It illustrates the flexible state distribution of the FoG relaying and its implications on scalability. In addition, Section 5.3 describes the study reviewing the state distribution of the routing. The robustness of connections in case of link failures is analyzed in Section 5.4.

Finally, Chapter 6 summarizes the results of my thesis. Chapter 7 gives an outlook to future research questions.

Fast readers with a background in computer communication are encouraged to read at least Section 2.1 about the terms used in Chapters 3 to 7 and Section 2.2.4 about the recursive reference model. The former will prevent misunderstandings, for example regarding the term “connection”. The latter supports the understanding of the structure of the proposed solution and of the thesis itself.

2. Background

This chapter introduces the terms and concepts required by this book.

At first, the basic terms used in Chapter 3 to 7 are defined in Section 2.1. This is important since the definition of some terms, such as connection, is not standardized. The description of the state of the art in this chapter uses mainly the terms as they are defined in the references.

Section 2.2 describes the traditional reference models OSI and the IP suite as well as a new recursive model. The historical context of each model is outlined in order to highlight the motivation of the model creators. The traditional reference models are presented to motivate the choice of the new recursive model as reference model. The recursive model provides guidance for the design decisions documented in this book.

In general, the focus of the description is more on the similarities of solutions. The invariants of solutions for similar problems are highlighted by the structure of this chapter. In particular, Section 2.3 does not follow the structure of traditional textbooks derived from the OSI layer model. Instead, protocols are presented from the point of view of the recursive layer model by grouping protocols for similar purposes.

Starting from Section 2.4, this chapter reviews the state of the art of providing functions within networks. This description is close to today's situation in networks or commercial products. Some of the introduced concepts or protocols will later be reused as building block for the new architecture. Others are presented to justify different design decisions in Chapter 3 and 4. Related research work is presented in Chapter 3 after introducing my contributions.

Since this book focuses on flexible networks supporting arbitrary functions, Section 2.5 introduces the state of the art for dynamic protocol stacks. Section 2.5 describes solutions for constructing such stacks and how to select the parts for the construction.

2.1. Terms and definitions

Since this book focuses on architectural issues, terms for ideas and elements of architectures play an important role. However, many terms in networking are not defined clearly or are not used in a homogenous way. Thus, this section defines the terms used in most parts of this book. Only Chapter 2 and sections about related work use the definition given here and the definition used by reference documents. The mixed usage should simplify the reading for readers, who are familiar with the reference documents.

The computer network and protocol community happily uses a lot of abbreviations. This thesis tries to avoid abbreviations in order to improve readability. While common and frequently used terms such Internet Protocol and Forwarding on Gates[1] are still abbreviated, more rarely terms are written out. Thus, "transfer service plane" and "autonomous system" are favored instead of TSP and AS, respectively.

2.1.1. Network, subnetwork, and inter-network

A *network* is a set of systems interconnected with each other via links. Examples for systems are host computers and routers. In not fully meshed networks, a system can act as *end system* or *relay system* for communication. End systems communicate with each other while the relay systems support the communication by relaying exchanged data [Day95]. Both terms refer to a role of a system. For example, a router can be a relay system for others and an end system for its management applications.

Along the same lines, I use the term relay network for networks relaying data for end systems. End networks are the networks in which an end system is located.

The Internet is not a homogenous network under the control of one single operator. It is a network of networks called *inter-network*. Each individual network is owned by an operator. Each operator might have different use and business cases in mind and might have different optimization goals for its network. Furthermore, different countries might have different laws and regulations for network operators, which influence the technique used for service provisioning. Consequently, the inter-network has to support a variety of network techniques, protocols, and policies. Since the term Internet is tightly bound to IP, the term inter-network will be used to refer to a network of networks independent of IP. The OSI term subnetwork is used to refer to a network within an inter-network, if the network aspects are important. Occasionally, the term *autonomous system* is used in the context of management decisions by operators.

The term network has a broader meaning and includes inter-networks as well as subnetworks. On a more abstract level, a network can be seen as graph with the systems as nodes and the links between them as edges. The mathematical terms will also be preferred if the described issue is independent of systems or refers to entities or virtual elements on systems.

In the context of recursive layers (cp. Section 2.2.4), the terms inter-network and subnetwork refer to two layers with different scope. The term network is

[1]This is a common term at least for this thesis.

used to refer to the graph of interconnected entities of a single layer.

The term *topology* refers to the characteristics of a graph. A topology is a set of graphs sharing common properties. For example, the tree topology contains all loop-free graphs. [Day08a]

2.1.2. Layer and its architecture

Communication systems are complex systems with a lot of interacting components. In order to simplify design, management, and extendibility, a reduction of complexity is required. A common approach is to group the components and to limit the relationships between these groups. The most prominent approach in networking is to define groups and arrange them linearly by allowing interactions only between one group and its two neighboring groups in the line. In such an arrangement, a group is called a *layer*. Since this arrangement is mostly drawn with a vertically orientation, each layer is allowed to interact only with its direct "lower" and "higher" neighbor layers. Most commonly, the layer most related to hardware issues is located at the bottom of this stack of layers and the layer most related to the applications on top of it.

The OSI model, which is described in Section 2.2.1, defines a generic set of terms to describe layers of any kind. A slightly modified version of these terms is used in this book and introduced in the following.

Four aspects of a layer can be separated:

- The service definition describes what a layer provides to higher layers. The description is rather abstract. Implementation details are not part of the service definition.
- Service access points with their service primitives describe the interaction between higher layers and the layer itself in order to access the service. The term *interface* is considered to be not as abstract and used in between design and implementation [Day95]. This book uses the term interface as a general term for design and implementation discussions.
- A protocol state machine describes the states of a protocol and the allowed state transitions. Instances of a protocol state machine reflect the current state of a protocol entity on a system.
- The protocol is an internal issue of a service and used to exchange commands and information between instances of protocol state machines.

The base line of this split is comparable to the split between interface and implementation in object-oriented software engineering.

Figure 2.1 shows a layer at the N-th position in a stack. It consists of two instances implementing the layer functions. An instance is called *entity*. The

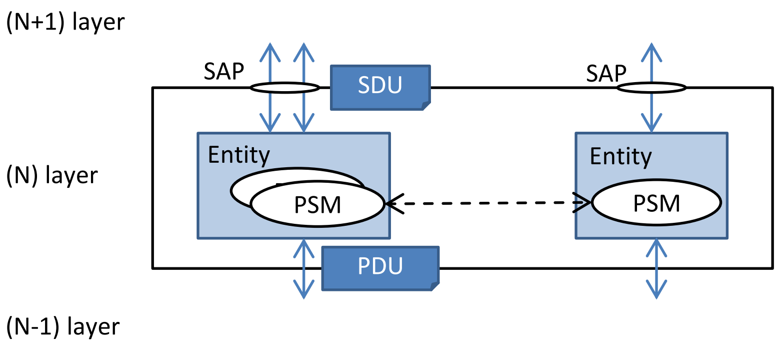

Figure 2.1.: Layer at the N-th position in the stack with its entities, protocol state machines (PSM), and service access points (SAP). Service data units (SDU) are exchanged with higher layers. Protocol data units (PDU) are exchanged with lower layers and between PSMs, respectively.

(N)-layer provides a *Service Access Point* (SAP) to higher layers at the position (N+1). Via this SAP, the (N+1)-layer hands over *Service Data Units* (SDU) for delivery. The (N)-layer forms a *Protocol Data Unit (PDU)* by adding *Protocol Control Information* (PCI) to the SDU. The PDUs are logically transported from the source entity to the destination entity. Since the entities normally do not share memory, the PDU has to be handed over to the lower layer at the position (N-1) in order to perform the delivery. This recurses till the lowest layer is reached and the PDU is physically transmitted over a medium. At the receiver side, the PDU is handed over from the (N-1)-layer to the (N)-layer. The (N)-layer is removing the PCI from the PDU and hands over the SDU to the (N+1)-layer. This recurses till the application receives its SDU.

An architecture defines logical components and their interactions [Cla05]. Consequently, an architecture for a layer defines the components of a layer and their interaction.

Approaches introducing interactions between non-adjacent layers are referred to as cross-layer approaches. An overview about different types of cross-layer approaches is given in [SM05].

2.1.3. Name and address

The terms name and address are overloaded with different semantics in network literature. For this book, these terms are defined based on Saltzer [Sal82] and Day [Day08a].

According to Saltzer, *names* are taken from namespaces. A *namespace* is the set of all allowed names. Names may be human-readable, like URLs, or machine-readable, like labels for the *MultiProtocol Label Switching* (MPLS) [RTF+01]. Since the differentiation according to readability is not important for this book, the term is used for both. More important is the differentiation according to the internal structure of names. Names having an internal structure are called *addresses*. In general, such a structure is defined in order to perform efficient searches in a hierarchical namespace. Names without an internal structure, such as hashes, are called *labels*.

A structure is only useful if it is known to someone using an address. Addresses used by Ethernet actually have an internal structure, which indicates the manufacturer of a network interface card (Organizationally Unique Identifier [IEE13]). Network management applications can use the internal structure in order to resolve the name of the manufacturer and the type of the interface card. However, this structure is of little help for routing protocols since it does not reflect the location of the network interface card. Thus, they treat such addresses as labels. Another example is the interpretation of URLs. For the DNS system, they have an internal structure used for looking up the DNS server responsible for storing the DNS entry. However, from the point of view of IP, DNS names are just labels. Thus, an (N+1)-address might be treated as (N)-labels. Consequently, the definition is extended by the context in which a name is used. [Day08a]

IP addresses are no addresses from the point of view of inter-network IP routing. Although unicast IP addresses are composed of a network and a host part, the network part itself does not have a structure and does not reveal a location in the graph of subnetworks. Other IP addresses, such as multicast addresses, identify groups and are not structured.

Proposals for a *locator/identifier split* [HMH13] used the term identifier for labels and locator for the addresses used by routing. Even though, this book discusses addresses in the context of routing, these new terms seems not to enhance the original definitions by Saltzer. Critical issues of locator/identifier split approaches are discussed in [Day08b].

2.1.4. Connection, quality of service, and requirements

In networking literature, a variety of terms is used for state shared between two protocol state machines handling a sequence of packets. Examples are flow, association, and connection. I require a term for such a construct independent from the amount of shared state. Based on the discussions in the G-Lab SIG "Functional composition", I use the term *connection*. It refers to an instance of a communication between two higher layer entities. A connection is mapped to some layer internal mechanism implementing the requirements associated with a connection. Depending on this mechanism, a connection may be mapped to a UDP-like construction with a small amount of shared state, a TCP-like construction with a large amount of shared state, or anything in between.

Chapter 3 shows how connections are mapped to chains of functional blocks. The recursive layer model described in [Day08a] prefers the term flow. It allows to map flows to EFCP connections (cp. Section 2.2.4). While the terms differ, the idea of splitting the request and its implementation is the same.

QoS is concerned about non-functional requirements and characteristics of communication such as data rate, loss, and delay [Tan03]. An overview about the QoS parameters for the OSI layers is given in [Sta93, p. 44]. The requirements might be "hard" or "soft". The former defines exact limits, which are not allowed to be exceeded. The latter defines thresholds, which have to be met for a specific percentage of packets. If not only the non-functional aspects are considered, the functional requirements have to be specified additionally. Examples for functional requirements are in-order data delivery or delimiting aspects (if the peer should receive the data in the same chunks as they are send). I use the term *requirements* to refer to a set of functional and non-functional requirements. The term QoS is used as short form to refer to non-functional requirements.

2.1.5. States

Protocols are used to exchange information between protocol state machines residing on different systems. Each protocol state machine requires some internal information about its current status called *state*. States represent the knowledge of a protocol state machine and require memory on the system hosting the protocol state machine.

In the context of QoS-provisioning, multiple types of states are known. For example, the authors in [DF99] differentiate between three types of states in the context of the *Resource Reservation Protocol* (RSVP) [BZB+97]:

- *Classification states* identify the connection a packet belongs to. They contain all knowledge a relay system requires to decide which function should handle a packet. RSVP uses a (minimal) classification state of source address, destination address, and protocol field value in order to classify IPv4 packets according to their connection.

- *Scheduling states* define the service for a packet. They contain all information required to define a service such as scheduling parameters.

- *Signaling states* include the control information such as timers. In contrast to the two previous ones, these states are used less frequently since they are not required for each packet.

Comparable to the term connection, I require a more general classification of states. It should be suitable for functions in general and not only for QoS-relaying functions. Thus, the following classification is used in this book:

- *Mapping states* identify the functions, to which a packet should be relayed to, and the parameters required therefore. Fine grain mapping states can specify the functions per connection. For example, they can map packets from a specific connection to a function, which ensures the relaying with 10 msec and with a speed of 20 Mbit/s. More abstract mapping states can map packets from a specific service type (e.g. VoIP) to a function, which ensures the prioritized relaying of such packets. The mapping states include the classification states with some parameters originally classified as scheduling states. Examples for such parameters are given in Section 3.2.1.1.

- *Function states* include all information required by a function, which are independent of a single packet or connection. That includes the signaling states and the remaining scheduling states. In particular, they include the buffers, timers, and authentication information.

2.1.6. Routing, relaying, forwarding

The calculation of routes through a graph is called routing. *Routing* calculates the best route connecting a source and a destination node with respect to constraints such as non-functional requirements. In general, it requires knowledge of the graph. In order to obtain this knowledge in distributed environments, routing protocols are used to exchange information about the graph between routing entities.

Relaying is the process of handling a packet in a relay system without knowing the whole network graph. It consists of determining the outgoing queue of a network interface card for an incoming packet based on information in the packet and a smaller information base.

Forwarding includes routing and relaying and refers to the whole process of transmitting data from a source to a destination.

For IP, routing is done by the Dijkstra algorithm and the graph information is distributed by the *Border Gateway Protocol* (BGP) or the *Open Shortest Path First* (OSPF) protocol. The graph information is called *Routing Information Base* (RIB). The result of this process is the *Forward Information Base* (FIB). It is consulted by the relaying process when a packet arrives at a system. In MPLS networks, both tasks are more different. The relaying is done based on tables containing labels, which are local for each system. Each label represents a *Label Switched Path* (LSP). Routing is only required in order to establish the LSPs.

2.2. Layer models

The debate about layers has a long history in the research community. How many layers a system has, depends strongly on the guidelines for grouping components to a layer. A UMTS system has a very different layer structure [HT11] in contrast to, for example, a DSL system [ML02]. In some cases these guidelines are not only relevant for a special system but for a family of systems. Such guidelines define a layer model or reference model, which can be used to judge the correctness or completeness of new system designs. In the following, the most prominent reference models are outlined. They are used later on to discuss the design decisions of the new architecture presented later on.

John Day states that a standard document is half political and half technical [Day08a]. Since all reference models are rooted in the thinking of the time they had been defined, the history of each reference model is reviewed. This review stresses the influences of the political and economic environment on the technical decisions. As far as they are known, motivations for design choices are outlined.

Instead of reducing complexity with a layered approach, less strict rules of how components may interact are possible. The approaches mainly follow the object-oriented paradigm such as the *Modular Communication Systems* (MCS) described in [Boc97].

2.2.1. ISO/OSI

2.2.1.1. History

In the early 1970s, the increasing need for distributed systems lead to many activities focusing on the development and standardization of communication systems. In particular, the network required to connect computers was one major issue. The *International Telegraph and Telephone Consultative Committee* (CCITT) started its work on computer networks standards in 1969 with the Joint Working Party on "New data networks" (later: Study Group 7). This group was dominated by the telecommunication companies and, thus, more focused on connection-oriented solutions. A meeting in 1970, mentioned an ARPANET presentation from Bob Kahn (see next section about IP suite), and summarizes that "little support was given for packet switching by any delegation" [CCI70]. After a first publication in 1972, the CCITT published its standard for X.25 in the so-called "Orange Book" in 1976. Networks such as Telenet in the US and the public British SERCnet used X.25 from the mid/end 1970 on.

In 1977, the British Standards Institute proposed to standardize a network for distributed processing to the *International Organization for Standardization* (ISO). ISO formed the Subcommittee 16 of the Technical Committee 97 for *OSI*. John Day states in [Day08a, p. 356] that this was a political maneuver. The CCITT was dominated by telecommunication companies and the upcoming computer companies had no influence in this standardization body. In contrast, ISO was a nongovernmental standardization body and gave the computer companies a better chance to push their ideas. The *American National Standards Institute* (ANSI) had to develop a proposal for the first meeting of this subcommittee. Mike Canepa and Charlie Bachman from Honeywell Information Systems participated in the ANSI preparation meeting in 1977 and presented their *Distributed System Architecture* (DSA). DSA resulted from the work of their group in the context of communication architectures for distributed databases, which started in the mid 1970s. It was influenced by the work of the ARPANET and by IBM's *Systems Network Architecture* (SNA), which was published in 1974. DSA was selected by ANSI as the best proposal and submitted to the first ISO meeting in March 1978. The proposal was accepted. After some refinements, the ISO committee finished a first draft in June 1979. Starting from 1981, CCITT and OSI aligned their work. OSI published a draft standard in 1982. CCITT published the compatible standard X.200. [Sta93]

The final standard ISO 7498 was frozen in 1983 and published in 1984 [Day95].

The standardization was mainly driven by governmental telecommunication companies, the new computer industry lead by IBM, and the researches from the DARPA Internet. The result was a technical standard influenced by the individual business goals of the participating companies. Consequently, OSI shares common parts with products the companies wanted to sell. The early Internet pioneers had a hard task to push the ideas of the Internet in this process. First of all, such a process was new for most of them. Second, they did not have the political support the big companies had. The conflicts between IETF and ISO/OSI date back to these times. [Day08a]

In 1988, the ISO committee SC21, which is responsible for OSI, decided to start the work on the first revision of the OSI reference model 7498-1. It was published in 1994 with only limited changes. John Day, working as the Rapporteur of this process and, thus, responsible to arrive at a consensus, stated that the most important problems regarding a cooperation between connectionless and connection mode, revision of the upper layers, and multicast had not been solved. The changes primarily cleaned up the document and aligned it with the new technologies. An example for the latter is the extension of data link layer tasks by routing due to new technologies such as Ethernet. [Day95]

Ethernet was developed at the beginning of 1972 by Xerox. Later, Digital Equipment (DEC), Intel and Xerox continued the development jointly. The IEEE standard family 802 for *Local Area Networks* (LAN) defines two sub layers within the OSI data link layer: *Logical Link Control* (LLC) (802.2) and *Medium Access Control* (MAC) (dependent on the access, e.g., 802.3 for Ethernet). [Hea93]

2.2.1.2. Model

The OSI reference model defines seven layers. The lowest one is highly influenced by the physical media and its limitations. Each subsequent layer increases the functionality. Finally, the application layer can rely on an abstract and highly flexible service provided by the presentation layer. This grouping of functions to layers was inspired by the design of operating systems. Figure 2.2 shows the following stack:

1. Physical: Transmission of an unstructured bit stream over a physical link. Defines electrical, mechanical, functional and procedural characteristics for connecting two devices
2. Data Link: Provides a reliable link with flow control between two devices and manages the link, which includes the activation, maintenance and deactivation of the link. It handles QoS for single links and might be capable to negotiate suitable parameters with the peer. Most prominent is the "high-level data-link control" (HDLC) protocol, which provides such features.
3. Network: Data transfer between end systems, which are not directly adjacent but connected to a communication network. This includes finding an end-to-end path (routing) and to convey the data over that path (relaying). Segments packets for different data link layers.
4. Transport: Providing end-to-end data transport. Depending on the requirements, the transport layer can provide reliable connections with flow control and in-order data delivery or no corrections at all.
5. Session: Controls the dialogue between applications; Token management to prevent mutual access to operations; Synchronization with checkpoints.
6. Presentation: Defines the format of the exchanged data and provides data-transformation capabilities; defines syntax and semantic of data.
7. Application: Logic dedicated to a specific application.

More detailed descriptions can be found in nearly all text books about networking. An implementation view on the model with an outline of its advantages and disadvantages is given by Rose in [Ros90].

Except the physical layer, each layer supports a connection and a connectionless mode. The connection mode includes all signaling required to setup, maintain and release connections between peers. The connectionless mode allows sending data without such a setup. However, Rose remarks that the upper layers use nearly exclusively the connection mode [Ros90]. Theoretically,

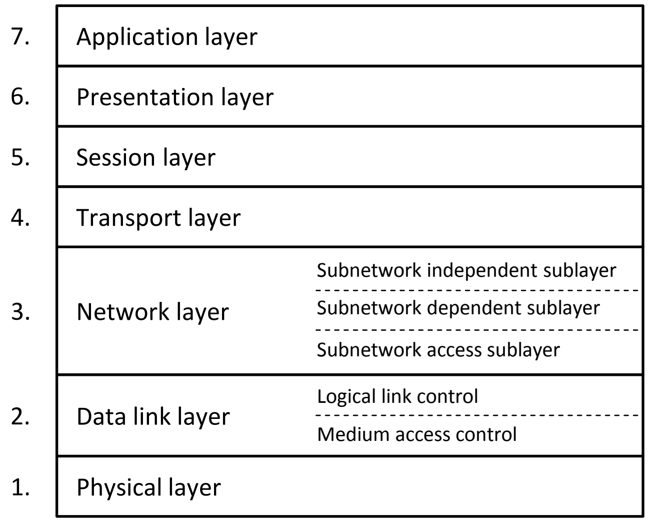

Figure 2.2.: OSI layer stack with the layer most related to hardware at the bottom. Sublayer are separated by dashed lines.

adjacent layers can operate in connection or connectionless mode independently of each other. Transition between both models is possible [Sta93] (e.g. connection to connection with one-to-one, multiplexing (many to one), and splitting (one to many)).

Basically, X.25 corresponds to the first three layers of the OSI model in connection mode. It uses virtual circuits over the *Line Access Protocol Balanced* (LAPB). The *Connectionless Network Protocol* (CLNP), which is similar to IP (including the *Internet Control Message Protocol* (ICMP)), takes over the connectionless mode. Depending on the network protocol, transport protocols of different classes (TP0-4) have to be used.

In order to support cooperation between subnetworks operating in different modes, OSI introduces the *Internal Organization of the Network Layer* (IONL) [ION88]. It splits the network layer in three sublayers:

- The subnetwork access sublayer defines the interworking between a specific subnetwork technology (e.g. Ethernet) and the network layer.
- The subnetwork dependent sublayer adapts the subnetwork technology to the desired network service (e.g. a connectionless subnetwork to a connection mode).
- The subnetwork independent sublayer provides the network service between the end systems.

Stallings [Sta93] and Day [Day08a] provide a detailed description of the sublayers of IONL. Rose points out the technical and political problems and limitations surrounding this issue. In [Ros90], he concludes that due to addressing and other problems resulting from combining X.25, CLNP, and the *Transport Protocolclass 4 (TP4)*, an interworking between different OSI subnetworks is in most cases not possible.

QoS parameters have been defined for each layer (Table 2.10 in [Sta93]), which includes throughput, transit delay, residual error rate, resilience, delay for establishing a connection, and others. Basically the higher layers pass these parameters down to the transport layer.

2.2.1.3. Problems

The OSI model was heavily influenced by the political situation during the time it was created. The telecommunication companies tried to defend their monopolies against the influence of the upcoming computer companies. The computer companies in turn tried to get into the market of communication. Some companies such as IBM had already products (such as SNA) and pushed the standard to match these products. In summary, the standard does not only reflect scientific insights but also the environment of that time. In particular, the complexity of OSI was strongly increased by the "war" between the connection mode and connectionless camps. A more detailed description of the political and market forces and their effects on the standard are given in [Ros90], [Day95] and [Day08a].

Furthermore, the standard was not defined in the best time period between the peak of research and the peak of investments. [TW11]

Together with the bad impression of the first implementations, the perception of the standard in the community was not the best. Thus, a tendency towards the alternative IP solution was created. [Ros90]

Moreover, new inventions do not fit in the reference model. The reference model does not "explain" these solutions and how they might fit in an overall architecture. Ethernet was among the first inventions leading to changes at the OSI model (see previous description of the first revision). "Shim layers" such as MPLS or overlays do not fit in the OSI layer structure as well.

The session and presentation layer are usually omitted as discussed in Section 2.2.3. John Day argues that there are even some indications that both have to be in reverse order [Day08a].

Several functions repeat themselves in multiple layers. For example, the data link and the transport layer handling reliability issues. Routing, originally a management function of the network layer, is now also present in the data link layer. Moreover, peer-to-peer overlay networks operating in the application layer perform their own routing. This repetition of functions raises the idea of recursive layers outlined in Section 2.2.4.

In addition, some of the OSI definitions seem to be unpractical. For example, addresses name SAPs not protocol state machines and connections are defined as association between (N+1)-entities. [Day11, Day95]

2.2.2. IP suite

The IP suite used for today's Internet is a collection of protocols with the Internet Protocol at its center. I use the term IP to refer to the IP suite and the protocol in order to avoid terms such as "IP suite nodes". Only occasionally, both have to be differentiated and the term IP suite is used.

2.2.2.1. History

The following condensed history of the Internet is based on [HL03], which points out the history and the biographies of the people involved in detail. This section focuses on the motivation and the inspiration of design decisions. References for the newer history are given directly in the text.

The RAND Corporation was ordered by the US Air Force to enhance the control system for the nuclear weapons of the US. This system depended on the long-distance communications network, which was highly centralized and, thus, vulnerable for a nuclear attack. Paul Baran, who joint RAND Corporation in 1959, had the idea to implement a distributed network with redundant connections in order to provide alternative routes through the network in case of node failures due to a nuclear war. Furthermore, he proposed to split data in small "message blocks" and to forward them independently through the network. Each node in the network should have a routing table and use a decentralized "hot potato routing" algorithm. The idea was inspired by the human brain, which can survive local defects. While the military liked his ideas and Baran was able to convince his colleagues in RAND Corporation, he did not succeed in convincing AT&T, which should build the system. The approach was too different from the circuit switched networks classical telecommunication companies used. In 1965, after five years, the project was stopped.

In spring 1966, Donald Watts Davies, physicist at the British *National Physical Laboratory* (NPL), gave a presentation with his ideas for a new public communication network more suitable for the bursty characteristics of computer generated traffic. His idea of splitting up data into smaller "*packets*", which are handled independently by the network, matches the one proposed by Baran very closely. Davies idea was born from his work with time-sharing systems in 1965.

In 1958 the *Advanced Research Projects Agency* (ARPA) was founded by President Eisenhower as reaction to the Sputnik shock. By taking over the research funding from multiple military organizations, ARPA had the task to ensure that the USA will win the scientific race against the USSR. In 1962, Joseph Carl

Robnett Licklider took over the “command and control division” of ARPA. Licklider was focusing on time-sharing and interactive systems and shifted the focus of the division to research in time-sharing systems, computer graphics and computer languages. Thus, the name of his division was changed to *Information Processing Techniques Office* (IPTO). In 1964, Licklider left ARPA and his colleague Ivan Sutherland took over. In 1965, Sutherland hired Bob Taylor. Taylor was influenced by Lickliders work on psychoacoustics and had changed over to computing. In 1966, Taylor took over IPTO from Sutherland and approached the “terminal problem”. Each contractor of his office requested money for its own computer in order to conduct time-sharing experiments. Moreover, the computers were so different that the exchange of results between universities and different computers was very costly if possible at all. Taylor proposed to link the computers and form a network. Instead of buying new computers for each research project, the researcher should use the network in order to access a suitable one. He saw the resource-sharing network as a tool for reducing costs of research projects and to overcome the incompatibility of hardware from different vendors. Taylor convinced Larry Roberts, who had prior experience in linking two computers, to act as project officer. Performance (below 500ms response time) and reliability were the primary goals for the network. However, ARPAs contractors were unwilling to burden their computers with the extra load of doing networking. Thus, one of the contractors, Wes Clark, came up with the idea of having small, interconnected specialized computers for the network. The computers of the sites should be attached to the network via these small computers. In the end of 1967, Roberts published an article about this idea in ACM SIGOPS Symposium on Operation System Principles and called such a small computer *Interface Message Processor* (IMP). At the same conference, a paper from Davies team at NPL described Davies idea and the small scale prototype for the NPL campus with a reference to the work of Baran. Roberts planned to build an equivalent system called ARPANET for the US. In December 1968, the consulting company BBN got the contract to build the IMPs. A group led by Frank Heart, which included Bob Kahn and others, designed the store-and-forward IMPs with a special focus on reliability. In particular, the IMPs performed packet retransmission to deal with bit errors due to the faulty telecommunication lines of these days. In December 1969, an experimental network with four nodes was operational. Afterwards, the IMPs were installed at a rate of one per month and the network grew rapidly.

Len Kleinrock at UCLA, a queuing system expert, got the contract for the Network Measurement Center. Kleinrock was leading a group of students including Vint Cerf, Steve Crocker, and Jon Postel.

At the same time, Roberts selected four universities, whose mainframes should be connected to the network at first. Each of these four universities had the task of implementing the connection between their computers and the IMP of their site. The university teams started in summer 1968 to design the "host-to-host" protocol. They called themselves the Network Working Group (NWG). In April 1969, the first note of their meetings, the *Request for Comments* (RFC) number 1, was written by Steve Crocker. The protocol development took more time than the IMP development and in summer 1970, the *Network Control Protocol* (NCP) was finished. Till July 1972, the Telnet and the File Transfer Protocol were designed.

The first big official demonstration of the ARPA network took place at the first International Conference on Computer Communication (ICCC) in October 1972. It consisted of various demos showing the access to resources via the ARPANET.

In parallel to the ARPANET, several other packet networks had been established. For example, a radio network demonstrated the wireless transmission of packets between islands in Hawaii. During the year 1973, Cerf and Kahn developed a protocol, which enables the exchange of packets between these networks. The solution consists of gateways between these different networks and a *Transmission Control Protocol* (TCP) for the host-to-host communication. In contrast to NCP used before, TCP assumed an unreliable network. This design choice was inspired by Louis Pouzin. Pouzin, responsible for the CYCLADES network at IRIA in France, stated that such "datagram" networks could operate without connections and that the function of recovering from packet loss or bit errors could be moved from the IMPs to the hosts [Pou74b]. TCP was demonstrated in October 1977 with the concatenation of three different networks. During 1978, TCP was split in one part required by gateways for relaying packets and another part responsible for dealing with errors and loss. The former was called Internet Protocol (IP). This split was inspired by the PARC Universal Packet (PUP), which had been developed by the Xerox Corporation. On January 1st in 1983, the ARPANET made its transition to TCP/IP.

Over the years, incremental changes had been added to the Internet in order to solve upcoming problems. Around 1980, updating the "host file" containing the mapping from names to IP addresses on each end system got too slow for the growth of the Internet. This problem was solved with introducing the *Domain Name System* (DNS). Starting from October 1986, the Internet faced severe congestion collapses. Jacobsons [Jac88] proposed to include a congestion-avoidance scheme in TCP. [Day08a]

Some more details about this "a history of change" is given in [Han06].

In 1974, a National Science Foundation (NSF) advisory board stated the enormous importance of the network for science in general and not only for computer science. However, access to the ARPANET was very expensive and not affordable by all universities. In 1979/80, a proposal for a Computer Science Network (CSNET) was developed, which should link computer science researchers in academia and industry. After the five year project duration, nearly all computer science departments in the US were connected. This success enabled the creation of an NSFNET backbone in 1985. It connected regional networks and was open for all academic institutions. Similar efforts were launched in other countries all over the world.

By the end of the 1980th, the NSFNET outperformed the ARPANET in terms of performance and number of users. Thus, ARPA decided to shut down the ARPANET by transferring the nodes to other networks (e.g. the NSFNET). The transition ended in 1989.

At the beginning 1990th, the shortage of IPv4 addresses and growing routing table sizes became a mayor concern and the "IP Next Generation" (IPng) activities were launched [BM95]. After rejecting to build upon the OSI protocol CLNP in 1992, new network protocols were designed. The IPng working group considered the Common Architecture for Next Generation Internet Protocol (CATNIP), TCP and UDP with Bigger Addresses (TUBA), and Simple Internet Protocol Plus (SIPP) proposals. Finally, the work on a successor for IP targeted on expanded addressing capabilities, header format simplification, improved support for extensions and options, flow labeling capability, and authentication and privacy capabilities. A first draft of IPv6 was published in 1995 [DH95] and was finalized in 1998 [DH98].

2.2.2.2. Model

Many network researchers believe that there is "no architecture, but only a tradition" [Car96] for IP. This is supported by the time difference between the design decisions and publications describing them. Important design papers had been published by Leiner et al. [LCPM85] in 1985 and by Clark [Cla88] in 1988 from a backward perspective. They describe the design of a connectionless inter-network service with a special focus on reliability. Reliability is ensured by a network specialized on routing without connection states. Simplicity and the "empowerment" of the end hosts are important. Less important are security and accounting considerations.

One of the most important design foundations of IP is provided by the end-to-end argument. The end-to-end argument suggests that if a function "can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system", it has to be implemented at the endpoints. In such a case, it is not "a feature of the communication system itself" [SRC84]. RFC 3724 [KAI04] describes the various versions of the end-to-end principles in their historical order. One consequence of the argument is that states and functions required to implement a loss- and error-free connection are moved from relay to end systems.

RFC 1122 [Bra89] served as guideline for implementing IP. It defines four layers, which are also used by Tanenbaum [TW11] to describe the structure of an IP stack. The layers follow closely the OSI layers 2 to 4 and 7:

- The link layer includes anything below IP. It includes the host to network protocols. However, it is more an interface than a protocol.

- The internet layer corresponds to the OSI network layer in connectionless mode. Its protocol is IP.
- The transport layer provides end-to-end communication services and matches the OSI transport layer. TCP and UDP are its protocols.
- The application layer includes anything above IP.

Starting from 2001, the IP stack was compared with an "hourglass" with IP at its waist [Dee01]. Evolution models inspired by biology try to explain this shape and to derive design hints for Future Internets (cp. EvoArch project [AD11] and references within). However, such models are criticized for being over-simplistic and for underestimating the control aspects [Agu08].

2.2.2.3. Problems

In contrast to the OSI model, the model explaining the design of IP had been developed after the protocols. It is very much tailored to the IP world and not useful for explaining networking in general. Tanenbaum states that in contrast to OSI "the TCP/IP model is not at all general and is poorly suited to describe any protocol stack other than TCP/IP" [TW11, p. 75].

Along the same lines, Day argues that IP is a reference model for subnetworks and not for inter-networks. IP misses a layer for the inter-network and uses addresses naming subnetwork point of attachments. [Day11]

IP addresses name the point of attachment of a node to a subnetwork. If a system has two points of attachments, it gets two IP addresses, which might be completely different. In particular, the network prefix might be different if a system is attached to the Internet via two different ISPs. Such multihoming scenarios lead to large RIB. [BGT04]

Names for nodes are missing. Compared with the theoretical solution proposed by Saltzer [Sal82], node names are required to keep them constant if the point of attachment changes. Practically, the lack of node addresses hampers IP in mobile scenarios, in which a mobile node changes its points of attachment and, thus, has to change its IP address. [Day08a]

IP addresses are visible to applications. The mapping from DNS names to IP addresses was introduced late (cp. Section 2.2.2.1) and was optional to applications. Moreover, the applications had to use interfaces requiring the application to know the IP address of the communication partner. The transition from IPv4 to IPv6 showed the problems of this missing abstraction.

The architecture of IP assigns all communication related functions to end systems. Relay systems are dedicated to routing, only. However, a lot of "middle boxes" have been introduced to the Internet. Firewalls are used to restrict the openness of the Internet and to block undesired or malicious traffic. Multicast requires group management in the network to perform efficiently. Anchor systems provide mobility support for IP. Finally, *Network Address Translation* (NAT) introduces gateways to extend the IP address space. This functionality in the network is not supported by IP and leads to management and connectivity problems and undesired feature interactions.

In [Han06], Handley notes that the Internet seems to fail to adapt to new challenges since 1993 and he fears that it might stagnate in the future.

Although the original design of IP favors simplicity, Perlman argues that today's IP implementations are larger and less reliable than these of ISDN [Per01].

2.2.3. Mixed versions

Due to the limitations of the two reference models, several text book authors introduce alternative reference models. A five layer model is described by Tanenbaum [TW11] and Stallings [Sta93]. This model is a "typical network architecture of the early 1970s" [Day08a]. The model consists of the following five layers:

- Physical layer
- Data link layer
- Network layer
- Transport layer
- Application layer

Compared to the OSI model, the session and presentations layer are omitted. However, the layers themselves follow closely the layer definition of the OSI model.

2.2.4. Recursive layers

2.2.4.1. History

At the beginning of the 2000th, the Internet with its IPv4 protocol had evolved to the most important and critical communication infrastructure for economy, science, and private life. The dramatic growth had broadened the diversity of the Internet users. From a pure research network in the early phases, the Internet had reached the public and nearly everybody's life in the industrial countries. This broad user base comes along with a broad variety of applications and business models. These business models have very heterogeneous requirements on the network. However, the best-effort and open Internet showed several limitations in terms of performance, reliability, security, and privacy. Moreover, the addressing problem – even though IPv6 was long standardized – has been still an open issue due to a very limited deployment of IPv6. This slow convergence from IPv4 to v6 raised questions about the flexibility and evolvability of the Internet. The public pressure caused so-called "Future Internet" research activities in nearly all aspects of the Internet. The US (GENI and FIND), the EU (within its Framework Program 7), and others set up large funding agendas to support and speed-up these research activities. Obviously the first Internet had opened new markets and nobody wanted to miss the opportunities a second new Internet might open.

In this wave of research activities, the reference model for networking got new attention. Multiple "clean-slate" projects invented new ways of networking and ways of structuring the network stack. Starting from 2006[2], the *Recursive Network Architecture* (RNA) project led by Joseph Touch proposed a single "metaprotocol", which recurses in an unbound number of layers. The flexibility of such a recursive layer was inspired by modular protocol systems and configurable protocols. Such an approach moves away from the static layer setup of the OSI or IP reference models and focuses on a single layer, which is repeated as often as required.

John Day, who was unsatisfied by the outcome of the first revision of the OSI model (cp. Section 2.2.1.1) in 1994, started to work on "patterns", which should lead to "major simplifications in our understanding of protocol architecture" [Day95]. In 2008, he published papers and a book [Day08a] about his results. Related documents are collected on the community webpage of the "Pouzin Society" [Pou]. His book proposes a recursive layer model and outlines the reference architecture of such a recursive layer. The model is called "Network IPC model". Instead of defining layers based on functionality, they are defined via their scope. In newer presentations and on webpages [RIN] the name *Recursive INternet Architecture* (RINA) is used.

In 2011, a joint work of Touch's group, Abraham Matta – who worked with John Day – and others proposed the *Dynamic Recursive Unified Internet Design* (DRUID) [TBD$^+$11]. It uses a single recursive block in combination with translation tables and persistent states to create a network. However, the project was not funded by the NSF.[3]

With begin of 2013[4], the EU project *Investigating RINA as an Alternative to TCP/IP* (IRATI) [IRA] started to work on a specification for RINA and to validate it with implementations.

[2] http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0626788
[3] According to John Day [private conversation]
[4] http://cordis.europa.eu/projects/index.cfm?fuseaction=app.details&REF=106030

2.2.4.2. Model

Currently, there is no official standard for networks with recursive layers. In the following, the Network IPC model is summarized, which has been proposed by John Day [Day08a]. From my point of view, it is closer to a new standard model than any other proposal for recursive layering.

Layers in the recursive model are not structured according to functionality as in the OSI model but to their scope. Each layer in the model provides the same functionality such as addressing, routing, relaying, resource management and error and flow control. The functionality comprises roughly the functions known from layer 3 and 4 of the OSI model. Each layer implements these functions for a different scope. For example, there might be four layers:

- Broadcast scope: The lowest layer handles the functions for a single link or broadcast domain.
- Subnetwork scope: The next layer is responsible for transmitting data through a subnetwork of an operator by using a variety of lower layers with broadcast scope.
- Inter-network scope: The next higher layer connects the subnetworks to one big inter-network.
- Virtual-private network scope: The last layer shrinks the scope to a private network that is built on top of the inter-network scope. The layer may restrict the access to some trusted entities.

There might be more or less layers if required by a network setup. The number of layers is not predefined as in the OSI or IP model.

Day calls a layer a *Distributed IPC Facility* to mark the difference to layers in the OSI/IP context. For example, the layer for the virtual-private network scope may also use layers with subnetwork scope as (N-1)-layers [Day08a]. I prefer the term layer to avoid the introduction of too many terms.

Due to the recursive nature, each layer provides the same interface to its higher layers. Such an interface has to encapsulate the layer efficiently in order to enable the different implementations of a layer. In particular, the addresses have to be internal to a layer. They must not be revealed to other layers. Only names are exchanged at the interface. Consequently, a layer is free to choose its mapping from higher layer names to its internal addresses and to choose the name space of its addresses. Furthermore, the interface hides the details of how requirements are satisfied. A higher layer is not able to choose the methods used to ensure, for example, reliability directly. It specifies its requirements, like lossless, and the layer itself has to decide how to achieve them. In terms of the other reference models, the transport protocol used for connections should be hidden.

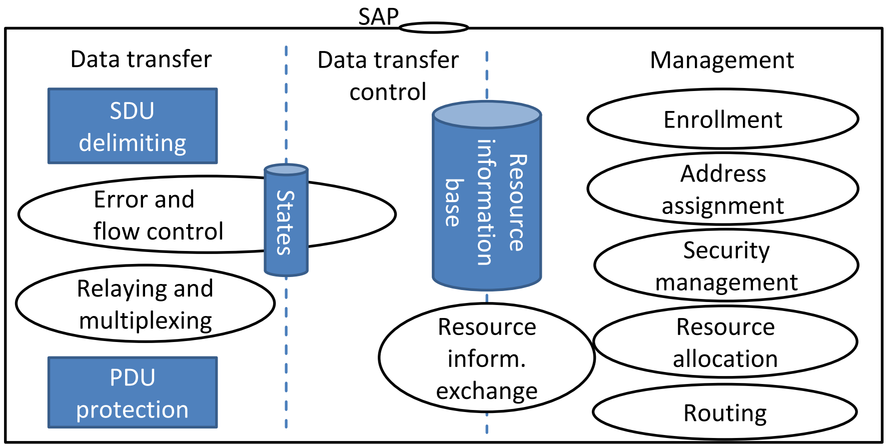

Figure 2.3.: Architecture of a single recursive layer (based on Figure 6-16 in [Day08a] and on the list of "The IPC Management Task[s]" [Day08a, p. 260ff])

The architecture of a single recursive layer is shown in Figure 2.3. It contains three parts that are loosely coupled via state vectors or information bases.

- Data transfer is responsible for relaying packets and handling buffers and queues. It takes over all tightly coupled mechanisms for transferring packets.
  - SDU delimiting function splits SDU into PDUs, which can be delivered by the layer. At the destination the SDUs are reassembled again.
  - Error and flow control is split in two parts. The data transfer part contains the tightly coupled mechanisms and uses only a basic set of PCI such as port identifiers, message identifiers, and checksums.
  - Relaying and multiplexing task relays PDUs between not directly connected entities of a layer.
  - PDU protection function is responsible for calculating checksums and encryption functions.
- Data transfer control contains loosely coupled mechanisms required to fulfill the requirements for packet delivery. Its main task is to create, configure and release the connections required for the data transfer part.
  - The data transfer control part of the error and flow control contains the loosely coupled mechanisms for the initial state synchronization.

Its states are shared with the data transfer part.

- Management includes all mechanisms required to maintain a layer. Some of its functions are listed in the following:
  - Enrollment creates enough shared state between entities of a layer that connection can be established. It contains the mechanisms necessary for an entity to join an existing layer and to share states such as timeout ranges and policies.
  - Address assignment ("directory") provides and sets up mapping information from names to addresses.
  - Security management includes authentication, protection against malicious lower layers and access control of higher layers.
  - Resource allocation reserves resources required to fulfill the requirements of connections.
  - Routing is responsible to extract forwarding tables for the relaying from the resource information base. For different QoS requirements of connections, routing has to support different optimization metrics.

A layer comprises several protocols to implement its service. In the following, these types of protocols are summarized. Each type is explained in Section 2.4.5 in more detail with some example protocols from today's networks.

- An *Error and Flow Control Protocol* (EFCP) implements reliability, order, and flow control. The protocol is used for maintaining shared state between two EFCP protocol peers. The amount of shared state depends on the requirements for the connections and ranges from a low amount of shared state comparable to UDP to much more shared state comparable to TCP. The protocol is split in a data transfer PDU and a control protocol[5] for the loosely coupled ones.

- An access protocol[6] is used to query the contact information about an application process. This includes, e.g., its address and requirements. Furthermore, the protocol is used to figure out if the requesting application is allowed to contact the application process.

- A *Resource Information Exchange Protocol* (RIEP) is used to exchange information between entities of a layer. The information comprises connectivity information required for routing, mapping information, and information about available resources on nodes and links. This protocol is used to update the resource information base.

[5] Originally named "IPC Control Protocol" in [Day08a]

[6] Originally named "IPC Access protocol" in [Day08a]. However, I omit the "IPC" in order to align the name with the names of the other types.

The relaying and multiplexing task requires some relaying PCI attached to EFCP PDUs in order to implement packet forwarding. “The PCI contains the source and destination addresses” [Day08a, p. 255]. It is required for networks not fully meshed.

RINA assumes that all layers use the same protocols. However, the ways they are used differ according to the scope and the conditions of a layer. [TGD+11]

A layer operates with six different identifiers [Day08a, p. 271f]. The three internal ones are:

- A port identifier is internal to the system an entity is located at. It is used by the (N)-layer and the (N+1)-layer to refer to a communication sequence.
- Addresses are used for internal layer operations such as routing and management.
- A connection identifier is used by the EFCP to distinguish between multiple connections. It may be created by the concatenation of source and destination port identifiers.

For the relaying of PDUs between entities of a layer, each layer operates over a graph representing the connectivity between its entities. Entities are nodes in the graph and connections between two entities via (N-1)-layers are represented by links. The latter are established in the enrollment of the management. In the enrollment, an entity joins an existing layer by establishing connections to other (N)-entities via (N-1)-layers. Therefore, the entity has to know the name under that the other entities are known at the (N-1)-layers. This name is called a *distributed application name*. This name is defined as “anycast or multicast name” [Day08a, p. 246] or only as “multicast-application-name” [Day08a, p. 247].

Multihoming is modeled by having connections to other entities of the (N)-layer via more than one (N-1)-layer. Mobility is dynamic multihoming.

2.2.4.3. Problems

There is no standardized recursive layer reference model. The model outlined before is a proposal of a single author and not as mature as the other older models. Furthermore, no larger network designed in a recursive way exists. Thus, there is little experience with such models in comparison to the others.

Even though the model opens new perspectives and provides guidance for the design of protocols and networks, some issues require further analysis. Especially the role of QoS and its relation to the protocol types is not described in detail. On the one hand, the analysis of the general problem of communicating systems indicates that QoS information is transported with access protocols in order to determine “whether the communication can be established” [Day08a, p. 204]. On the other hand, the description of the model does not confirm that. It states that “when an [access protocol] request returns successfully, [. . .] management must determine whether and how to allocate the flow/connection to a new or existing flow” [Day08a, p. 261]. An access protocol “is required to carry [. . .] names to the remote” peer [Day08a, p. 257] in a “simple request/response” [Day08a, p. 201] fashion. Relay systems are required to deliver an access protocol message [Day08a, p. 269f] but seems not to be involved in setting up QoS.

The model leaves open algorithmic questions. In particular, John Day raises the question of how to make addresses topological.

2.3. Protocols

As pointed out in Section 2.2.4.2, a recursive layer requires a set of protocols in order to implement its service. In the following, some existing protocols are classified according to the protocol categories required for recursive layers. All protocols of a category serve the same purpose. Therefore, the similarities between the protocols are dominant. The differences due to the different scopes, for which the protocols had been designed, are highlighted. This way of presenting protocols should highlight the building block character of the protocols. Some of these protocols are useful for the creation of a new recursive layer. The others are required later on to justify different design decisions.

This section is more concerned about putting protocols in the context of recursive layers according to Day [Day08a] and not too much about describing each protocol in detail. For details about individual protocols, the reader is referred to the references given in the text.

In addition to the protocols, the common relaying PCI is outlined in more details since it is a mayor building block for the solution developed later on.

2.3.1. Relaying protocol control information

If not all nodes in a network are fully meshed, the relaying and multiplexing task has to relay packets between nodes and to multiplex them on (N-1)-connections. It requires relaying PCI attached to each packet. This common part of the protocol control information is used to forward the packet through intermediate nodes to its destination. [Day08a, p. 215]

The network protocols of the other reference models provide examples for such relaying PCI with various formats. The diversity of existing *relaying PCI formats* contradicts the definition that only addresses are stored in the relaying PCI (cp. Section 2.2.4.2). I introduce a classification of relaying PCI formats according to two different criteria, criteria structuring the long list of

		QoS	
		External	Orthogonal
Relay	External	Circuit switched netw.	
		GMPLS	
		SDH	
	Destination-based	DDP	CLNP (opt.)
		Ethernet	Ethernet+802.1Q
		IPX	IP (TOS / traffic class)
		PUP	PFRI (knobs & dials)
			SIPP (prio. bits)
	Route-based	Frame Relay	ATM (cong. bit)
		MPLS	PARIS (Prio. bits)
		OvIP	
		PARIS	
		[Pathlet]	
		SIRPENT (Viper)	
		[X.25]	
	Hybrid		PIP

Table 2.1.: Classification of relaying PCI formats according to their fields

developed relaying PCI formats. First, the formats are classified according to their encoding of the relaying direction into external, destination-based, route-based, and hybrid formats. The encoding strongly influences the data structure and size of the required FIB. Second, the formats are classified according to their ability to specify QoS-related parameters into external and orthogonal. In the following, the classification is presented in more detail. The details of the classified protocol headers are shown in Appendix A. Table 2.1 summarizes the classification.

2.3.1.1. External

The first class of formats stores the information about how to relay a packet not within the packet but external to it on systems. The FIB of systems contains a description of some kind of incoming channel and an indication to which outgoing channel the packets have to be relayed to. The packet itself does not influence the decision.

For example, optical networks using *Generalized Multi-Protocol Label Switching* (GMPLS) can define a "channel" based on the wave length. All packets received via such a channel are relayed to a specific outgoing interface and an outgoing wave length. Other systems can use time-division multiplexing for defining such channels.

External relaying PCI is especially useful for transmitting packets without additional overhead caused by headers. Therefore, such a format is preferred by telecommunication backbones such as the *Synchronous Digital Hierarchy* (SDH), which have to transport comparatively small packets with a short delay. In most cases, the channels and the rules for the FIB are rather static. Since packets are not evaluated by relay systems, most networks based on external relaying PCI formats require an out-of-band signaling.

2.3.1.2. Destination-based

Destination-based relaying PCI formats transport destination names, which are used as input for forwarding decisions. For each destination, the FIB contains the direction where to forward the packet to.

The most prominent example is the Internet Protocol (IP) [Pos81, DH98]. A relay system has to forward packets according to the destination IP address stored in each packet. For each IP address, the FIB contains the outgoing port and IP address of the next hop, to which the packet has to be forwarded to. In order to reduce the size of the FIB, IP addresses having the same next hop are aggregated according to their prefixes. If, for example, all IP addresses with the prefix 141.24/16 share the same next hop, the FIB contains only one entry with this prefix. The entry with the longest prefix, which matches a destination IP address, specifies the next hop for a packet. Consequently, the better the aggregation of IP addresses to prefixes the smaller the FIB.

Further examples are *Datagram Delivery Protocol* (DDP) [App94], *Internetwork Packet eXchange* (IPX) [FLSS99], *PARC Universal Protocol* (PUP) [BSTM80], *Simple Internet Protocol Plus* (SIPP) [Hin94], *CLNP* [Int86], and Ethernet [IEE08]. They differ in the size of the destination name. If the name is assigned dynamically, its size is directly related to the scope of a protocol. The bigger the scope the longer the name must be. If names are assigned statically, such as MAC addresses, they do not depend on the scope of an Ethernet network but on the amount of produced hardware elements.

The semantic of the destination name differs. An IP address names a network interface (a subnetwork point of attachment). In contrast, a CLNP address names a node.

The *Postmodern Forwarding and Routing Infrastructure* (PFRI) [BCG$^+$06] uses a list of addresses, which name the links, a packet should travel through. Comparable to the loose source-route of IP, the list might only be a partial one. Since the link addresses are globally valid and a relay system requires a mapping from destination link address to “next hop” link address (at least for large networks, which use aggregation) the protocol is classified as destination-based.

2.3.1.3. Route-based

Route-based relaying PCI formats specify the next hop without the indirection of destination addresses. A packet contains a name (in most cases a label) that refers to a local rule specifying how to handle the packet. This name is only valid locally and must be inserted in the packet before it reaches a relay system requiring the name. In an extreme case, all names required during forwarding are inserted at the source of a packet. This stack of names is called an explicit route. In the extreme case mentioned before, it is a source-route.

The size of a single entry in a route is rather small compared to the names included in destination-based relaying PCI. A single entry is only valid locally and, thus, its size has to reflect the number of "outgoing ports" or number of "context information" of a relay system.

A prominent example is *MPLS* [RTF$^+$01], which is doing forwarding based on labels for outgoing paths. Each packet contains a label, which refers to an entry in the local FIB of a relay system. For each label the FIB contains the next hop information. Since the labels are only valid locally, each hop replaces the label of an incoming packet with a new label, which is valid for the FIB of the next relay system. Additionally, labels can be stacked to reduce the number of labels maintained in the system. The topmost label of the route is treated as described before. A FIB entry can specify the removal of the topmost one or to add more labels.

Other protocols focus more on the route itself and do not replace the topmost label for the next hop. They use the topmost label for querying the FIB entry and remove it from the route afterwards. Each label in a route used by such protocols refers to one forwarding decision. In particular, some use indices of local FIB entries as labels. In the following, such routes are called index-based routes. Examples for protocols using such routes are *Asynchronous Transfer Mode* (ATM) [MP02], *PARIS* [CG88] and *Pathlet* [GGSS09]. Although the Pathlet routing proposal does not specify a relaying PCI format directly, the authors describe a protocol using a stack of "forwarding identifiers", which are valid only locally. Examples using not only indices but more information per hop are *Sirpent* [Che89] and *Overpass IP* (OvIP) [FRM97]. Some (esp. the older) approaches are more related to subnetworks. Pathlet - the newest one - deals specifically with policy issues in inter-network routing.

X.25 is a network access protocol and not used for relaying in a network. However, it uses locally valid labels to differentiate between connections.

2.3.1.4. Hybrid

Hybrid relaying PCI formats combine destination-based and route-based aspects. Thus, they can define the path for a packet based on destination names (e.g. names of relay systems) and explicit routes (e.g. stacks of indices).

The *P Internet Protocol* (PIP) [Tsu92] introduces a variable length header field, which contains a list of "routing hints". Each hint encodes some piece of information required by a relay system to make a relay decision. The proposal mainly encodes each part of a hierarchical address as one hint. Each relay system uses the hint for the current hierarchy level as index for its FIB. That usage pattern would suggest a classification as route-based. However, a hint might be a destination address, which requires a level of indirection as for destination-based relaying PCI formats. Therefore, the protocol is supporting both and is classified as a hybrid protocol.

2.3.1.5. QoS support

Some relaying PCI formats do not support the storage of QoS-related information. If QoS should be supported, the information is either integrated it in the relaying direction or provided external to a packet. Relaying PCI formats supporting the storage of QoS information in a packet provide a dedicated field for them. The QoS information can be set orthogonal to the direction information.

For example, IP specifies the non-functional aspects of each packet in the *Type of Service* (TOS) field (IPv4) or traffic class field (IPv6). The fields are orthogonal to the destination specification.

Ethernet, DDP, PUP or IPX do not support QoS directly and rely on context information located on relay systems. In order to enable a differentiation of multiple connections between two end systems such protocols often include socket numbers. A combination of source and destination addresses and port numbers acts as connection identifier. Examples are PUP and IPX. If socket numbers are known by the relaying system, they can be additionally used for error messages (e.g. for MTU errors or congestion notifications). Ethernet and DDP are examples without socket numbers.

Packets with orthogonal QoS PCI fields can be subject to additional QoS information on relay systems. However, the connection identification might require a more or less deep packet inspection. For example, the connection identification for IPv4 requires more information than provided by the IPv4 header (e.g. port numbers of the transport protocol). IPv6 and SIPP are counter examples, which provide header fields for both, connection identification and QoS aspects. The connection identification is done via a combination of names and a single flow label instead of socket numbers.

In general, QoS or other application requirement aspects are not discussed in detail for protocols with index-based routes. Obviously, parallel edges between two hosts with different indices can be used to represent (virtual) links with different QoS attributes. Thus, most route-based relaying PCI formats rely on QoS context information on relay systems. Their routes link a packet to some local information on relay systems, which specifies the direction as well as the non-functional aspects. Exceptions are ATM and PARIS, which provides short priority fields that are used in case of congestion, and hybrid protocols such as PIP. MPLS might be also an exception, if the experimental bits are used as "TOS field" analog to IP.

PIP is specifying QoS indirectly via a "logical router" field. The field selects the role of a relay system. Such a role might include non-functional aspects and leads to different FIB entries for the same hint. The examples in [Tsu92] use 1 bit of the logical router field for QoS. Thus, the QoS information is integrated in the direction decision.

2.3.2. Error and flow control protocols

As the name suggests, an *error and flow control protocol* handles lost packets or bit errors that occurs during transmission. If requested and if necessary, it might reorder received packets. Furthermore, it controls the data rate of connections in order to avoid congestions in the network and to protect slow receivers from being overloaded by fast senders. Furthermore, the protocol has to delimit the SDUs received from the higher layer. If a SDU is larger than the maximum PDU supported, the SDU has to be split up. Multiple small SDU for the same destination might be concatenated. The question of whether the original SDU has to be re-assembled at the receiver depends on the higher layer. For higher layers doing streaming it might be unnecessary. For datagram-oriented higher layers it is required [Day08a, p. 253ff].

An EFCP uses one data transfer PDU for the tightly coupled mechanisms and a control protocol[7] for the loosely coupled ones.

The most prominent EFCPs for multi-hop scenarios are TCP and UDP. TP4 is an OSI transport protocol. In the scope of a broadcast domain LLC can be used. However, newer protocols such as *Xpress Transfer Protocol* (XTP) [SDW92] separate the protocol from configuration and are capable of handling the whole configuration space.

2.3.3. Access protocols

In [Day08a], these protocols are named "IPC Access Protocols". However, I omit the "IPC" to align the name with the names of the other protocol classes[8].

Access protocols are used to find a higher layer entity and request permission to setup a connection from it [Day08a, p. 201]. Additional, they signal which instances of the EFCP transmit data for a connection [Day08a, p. 207]. It transports the source and destination application names, access control and capability information, QoS requirements, and identifier for the EFCP peers

[7] Named "IPC Control Protocol" in [Day08a]
[8]The abbreviation AP is not used in this book.

("port-ids") used for the connection (Figure 6-6, page 207, and page 269 in [Day08a]). They decouple higher layer request from the EFCP actions by taking over the following tasks (quotations from [Day08a, p. 257]):

1. "Finding the address of the destination" layer instance, which hosts the destination higher layer entity.

2. "Determining whether the requested [higher layer entity] is actually at that destination and the requesting [higher layer entity] has permission to access" it.

3. "Carrying information to the destination [layer instance] on the policies to be used for the requested communication and returning the destination's response to the source".

The information in the resource information base guides the search of an access protocol for the location (meaning: address) of a higher layer entity. In contrast to today, the protocol verifies the information by sending a signaling message to the layer instance of the destination. If the resource information base contains no or wrong information (e.g. due to mobility), access protocol messages spread out to determine the current destination. If the destination is found, it is asked for permission to set up a connection. The decision is influenced by the parameters transported by the access protocol mentioned before. If it is positive, the management can bind the connection to the EFCP instances specified by the access protocol. The management does not necessarily have to create new EFCP instances. The access protocol can be used to assign an existing EFCP "flow" to a new connection between higher layer entities. [Day08a, p. 257f]

In today's Internet, parts of the access protocol functionality are integrated in the control protocol of the EFCP. For example, the control protocol of TCP includes the higher layer instance names in form of port numbers (e.g. destination port 80 for the web server). This integration loses the flexibility to reuse existing TCP "flows" for new connections. The localization of the destination depends on the correctness of the resource information base. If the higher layer name is mapped to a wrong IP address (and port number), no connection can be established.

As stated in Section 2.2.4.3, it is not clear if the access protocols are responsible for signaling non-functional requirements of connections to relay systems as well. If we assume that they are responsible, relay systems would have to react on access protocol message passing through. Their management would have to reserve resources in order to satisfy the non-functional requirements of connections. Thus, typical resource reservation protocols such as RSVP and *Next Step in Signaling* (NSIS) take over this task of the access protocols in today's networks.

2.3.4. Resource information exchange protocols

Resource information exchange protocols are used to inform layer entities about capabilities and states of remote entities. This information comprises connectivity between entities, current state and load of links, and mapping information. [Day08a, p. 259]

Such protocols feed the resource information base, which is used to derive routing and allocation decisions of an entity. Furthermore, the information base acts as cache for all information required locally for operating an entity, such as its name and policy. Due to the delay and cost of exchanging information between instances, information bases reflect only a partial, potentially outdated state of the remote entities. Decision algorithms have to take this unreliability into account.

The BGP and the OSPF protocol are protocols for exchanging connectivity information in today's networks. The former had been designed for the scope of inter-networks and the latter for the scope of subnetworks. Both can (be extended to) carry load information.

Mapping information is typically exchanged with dedicated protocols. Examples are the DNS for the inter-network scope, Bonjour [App] for the subnetwork scope, and the *Address Resolution Protocol* (ARP) for the broadcast scope.

Not all layers require a directory for the mapping. Peer-to-peer overlays, such as Kademlia [MM02], derive the address by calculating a hash value directly from the name.

The *Dynamic Host Configuration Protocol* (DHCP) is used to initialize important information in the resource information base such as the name of the system, DNS server names, and gateways in a subnetwork scope.

Management and load information are typically exchanged with the *Simple Network Management Protocol* (SNMP).

2.4. Connections, states and quality of service

Packet switched networks split communication data in small fractions, called packets. Furthermore, they do not provide exclusive physical connections between end systems but multiplex packets from various communications on physical connections. Such networks are especially useful for connections with varying traffic characteristics, such as data bursts. By multiplexing packets of different connections to the same physical network resources, the resources can be used more efficiently compared to a circuit switched approach. According to [Heap93], packet switched networks can be classified according to their setup procedure:

- Datagram networks do not require any individual setup before a communication can start. The end systems of such *connectionless* networks can send their packets without any connection setup. The relay systems are not aware of these packet sequences and their requirements. Examples are IP and Ethernet networks.

- Logical/virtual circuit networks require a connection setup within the network before a communication can start. Such *connection-oriented* networks inherit the possibility to control the non-functional aspects from the circuit-switched networks by being aware of packet sequences. They are useful for data exchanges requiring constant non-functional characteristics such as a constant data rate. Examples are MPLS, ATM and Frame Relay networks.

A comparison of datagram and virtual-circuit networks is also given in [TW11, p. 379].

One of the most important problems discussed in networking history is the synthesis of connection-oriented and connectionless networks. In more general terms, Day states that the "primary distinction between connectionless and connection transmission is the amount of shared state. Connectionless transmission represents less shared state than connection mode. A connection mode functionality can be built quite simply on a connectionless functionality by adding mechanisms to the connectionless mechanisms. The converse is not true." [Day95]

Connections are often the entities that are subject for resource management. Relay systems can control the non-functional characteristics of connections by appropriate buffering, multiplexing, and scheduling their packets. The more connection-oriented aspects a network supports, the more control does it have over non-functional aspects. This benefit comes along with some disadvantages causing scalability problems:

- Setup delay: Signaling messages have to be exchanged in order to set up a connection. That introduces delay and induces processing load on relay systems. The setup costs are especially problematic for short connections with a very low amount of data.

- States: Relay systems have to store some states about connections in order to manage their resources. These states require memory. The classification states are particularly problematic because they influence the handling of packets and, thus, have to be processed for each packet.

In the following, solutions with different amounts of shared state within networks are presented. The description starts with solutions handling a small amount of shared state and continues with solutions with more shared state. Afterwards, the special aspects of inter-networks are outlined in Section 2.4.4. The technical mechanisms influence the distribution of power between the market actors. The political dimension of QoS-provisioning is described in Section 2.4.5.

2.4.1. Connectionless networks and overprovisioning

Pure connectionless networks do not support any non-functional guarantees since they are not aware of connections and their requirements. Since the connectionless relay service requires only the destination name for each packet, destination-based relaying PCI formats are used.

Overprovisioned networks provide resources well beyond the resources required to satisfy connection requirements. They provide relaying characteristics near the *Nature of Service* (NoS), which is defined by physics (e.g. speed of light). Backbone networks of telecommunication providers tend to be overprovisioned and operate at a maximum of 20 to 30% of its capacity.

A mathematical model for estimating the efficiency of overprovisioning regarding its capability to handle variations of traffic is presented in [HG05]. The authors conclude that overprovisioning has advantages as long as router capacity increases faster than the number of routers in a network grows.

2.4.2. Connectionless networks with quality of service add-ons

If the network resources are limited and significantly utilized, QoS mechanisms become important. They can prevent overload situations or at least secure important traffic from being affected by congestions. Thus, the higher the load in a network tends to be, the higher the impact of QoS mechanisms. Examples for networks with limited capacity are radio access networks and last miles to customers of wired networks.

2.4.2.1. Type of connection

The scalability problems caused by the number of connections can be avoided by introducing a limited set of service classes instead of connections. Each relay system knows about the service classes and how to handle traffic for each class. Since this set is limited, no scalability problem arises. A typical example is to introduce a priority value for each class that defines which packets are dropped in case of congestion. End systems classify their connections and store the identifier of the service class in each packet of a connection. This scheme cannot enforce QoS guaranties on a per-connection base. However, it enables a differentiation between types of connections.

For IP networks, *DiffServ* [BBC+98] uses the header field “type of service” (IPv4) or “class of service” (IPv6) to encode service classes in IP packets. DiffServ is mainly used by subnetworks to provide better QoS for delay sensitive services such as voice and to limit the traffic of data rate-intense Peer-to-Peer overlay networks.

For transit traffic or incoming traffic, the mapping to service classes is mainly done by more or less deep packet inspection at the gateways of a subnetwork. Comparable to the set of service classes, the set of rules for this mapping is rather small. Thus, the amount of classification states is limited as well. Since the DiffServ standard does not define a signaling protocol to create service classes, the setup is mainly done manually. End systems are neither able to influence the mapping nor are they informed if their connections are mapped to some specific classes.

2.4.2.2. Packet states

Instead of encoding only the type of a connection in a packet, the entire requirements of a connection can be stored within each packet. A relay system decides about the relaying based on the connections, from which it has packets in its buffer. Other connections that are routed through a relay system but do not send packets are not known to a relay system and do not influence the decision.

For subnetworks, Stoica proposes a stateless core network called *Scalable Core* (SCORE) [SZ99] that is able to emulate an IntServ solution. Each packet contains "dynamic packet states", which tells the relay system about the QoS requirements of the connection and of the consumed fraction of the individual packet. In particular, packets transport three parameters for the scheduler used by relay systems (Jitter-Virtual Clock) and one for admission control (cp. Section 5.2 of [SZ99]).

2.4.2.3. Volatile connection

Connectionless networks can support QoS for selected connections. For these connections, the problems mentioned on page 40 apply.

A difference between connection-oriented and connectionless networks with a connection-oriented add-on is that the latter do not drop packets not related to a connection but relay them with best-effort quality. This feature relaxes the coupling between resource reservations and routing. The relaxed coupling may cause more routing states on relay systems. Connectionless networks with an add-on have to know the routing for established connections and, additionally, the routing for all other destinations. Thus, the amount of routing states is maximized. That is somehow contradicting the colloquial phrase that connectionless networks are "dumb networks" since they minimize their internal states. However, this minimization is only done for the states about resource usage and not for routing. [Day08a]

The most prominent example for IP networks is the *IntServ* model [BCS94] with its signaling protocol *RSVP* [BZB+97]. Another signaling framework is *Next Step In Signaling (NSIS)* [HKLdB05]. For RSVP, a combination of source address, destination address, and protocol field of IPv4 serves as connection identification, which is included in the classification states. Optionally, the port numbers from the transport protocol can be added. If this is not possible, e.g., because the transport protocol is not known to relay systems, RSVP cannot distinguish between multiple transport connections between two peers. IPv6 introduces a header field called "flow label". In combination with the source and destination IP addresses, it may be used to identify connections on IP level without knowledge about the transport layer [RCCD04].

RSVP uses a *soft-state* approach. Soft-state indicatesthat the reservations on relay systems are only valid for a limited time. In order to extend their duration, a keep-alive signaling message has to be sent. This approach exploits the loose coupling between the reservations and routing. If routing changes, packets are relayed on the new path in a best-effort manner. The next keep-alive message will try to create the reservations on the new path. The reservations on the old path will expire after a while without an explicit release message. RSVP can operate in partial deployment scenarios, where not all relay systems support RSVP. However, the requirements of the connection can only be "fulfilled" if the relay systems not supporting RSVP belong to an overprovisioned part of the network.

The scalability problem regarding the number of connections can be approached by reducing the number of states via aggregation of connections. [BIFD01, DF99]

Overlays can improve the QoS of the Internet as well. An example is the overlay-based architecture for reducing packet loss OverQoS [SSBK03]. It uses "controlled-loss virtual links" between the overlay nodes in order to provide "statistical loss and bandwidth guarantees". It uses a hybrid forward error correction and automatic repeat request method to achieve loss rate limits.

2.4.3. Connection-oriented networks

Connection-oriented networks require a connection setup before data can be exchanged. Such connections include not only the states on the end systems but additional states on all relay systems. The exchange of signaling messages is done either in-band or out-of-band. The former is implemented with specially marked packets and mainly used by approaches inspired by the Internet. Approaches inspired by telecommunication networks prefer out-of-band signaling (e.g. Signaling System No. 7) via special channels or even separate infrastructure. Some approaches support both (e.g. frame relay).

In order to enable relay systems to identify the connection a packet belongs to, a packet has to have some fields that can be used to identify a connection.

It does not necessarily have to include the destination. Since destination names are, in general, longer than connection names with a local scope, relaying PCI formats of such networks tend to use short labels. Example protocols are listed in the Section 2.3.1.3 about route-based relaying PCI formats.

2.4.4. Inter-network issues

The previous description focused on subnetworks. For inter-networks, the situation is more complex since each *subnetwork* of an *inter-network* might operate in a different mode[9]. OSI address this issue by introducing three sublayers for the network layer. As pointed out in Section 2.2.1.3, the IP model includes only a single subnetwork layer. Thus, the model does not include inter-networks with different kinds of subnetworks. The solutions of the Internet community to this problem are called "two tier approaches". Tier one represents the solution for each subnetwork while tier two handles the inter-network aspects and concatenates the tier one solutions to end-to-end connections. A survey about one and two tier solutions for QoS is given in [VPMK04]. A typical solution consists of a connection-oriented add-on on inter-network level and a connectionless network with service classes or SCORE-like approaches on subnetwork level. This combination of IntServ and DiffServ/SCORE has the advantage that connections are known to each subnetwork and that each subnetwork can handle these connections with a less state-intense approach internally. However, the scalability problems of the connection-oriented part remain for the gateways of each subnetwork. They have to map packets to connections and connections to the internal service classes. Thus, they have to store at least a classification state for each connection. In practice, the connection-oriented add-on for the inter-network part is omitted due to interoperability, political, and business case reasons.

From a more general standpoint, an inter-network can either define mappings between each adjacent subnetwork or define a single standard, which has to be supported by all subnetworks (see Section 5.5.2 in [TW11]). For example, the Internet defines the connectionless IP as the least common denominator for data relaying between all its networks[10]. Such a common standard has the advantage of a lower complexity and enables a subnetwork operator to define the mapping from its subnetwork technique to the inter-network approach independently from the techniques used by its neighbors.

However, the inter-network standard defines the least common denominator of the subnetwork techniques. If a subnetwork is providing more features than supported by the inter-network, these features might not be available for transit connections or for connections with other subnetworks of the inter-network.

[9] More differences between networks are listed in Figure 5-38 in [TW11].

[10] Furthermore, supporting BGP is required to participate in routing.

Since IP does not provide a standard notion for a connection, connection-oriented networks are not able to map their connections natively to IP. Two connection-oriented networks willing to communicate with each other have to use the second method of how an inter-network can be implemented: They have to define a mapping between their subnetwork techniques. As discussed before, this can be done as add-on to the Internet. Other solutions use application specific protocols, such as the *Session Initiation Protocol* (SIP), on top of IP.

2.4.5. Political aspects and network neutrality

Due to the enormous importance of large communication networks for business and private life, its aspects are highly political. Thus, the *socio-economic aspects* are important for the design of an inter-network. Due to the enormous complexity of the topic, this section just gives a short introduction. The interested reader is referred to a more detailed analysis in [Zit08] and a survey in [DSW09].

As networking history shows, the design of a communication network has a strong impact on the distribution of power among the participants. An "intelligent" connection-oriented network, which provides services for "dumb" end systems, empowers the owner of the network. In contrast, a "dumb" connectionless network reduces the dependency of the end system on the network and, thus, reduces the influence of the network owner. In combination with cheap equipment owned by private people, an "intelligent" end system empowers private people. The last two decades with countless new services showed the enormous innovation potential. However, the fast growth of the Internet was enabled by the result of the previous innovation cycle. The "intelligent" network, which started its revolution against the telegraph back in 1877 when the Bell Telephone Company had been founded, provided the links for the "dumb" Internet. In summary, history shows that the empowerment of people is an important factor for innovations. However, the network, which interconnects them, is important as well. The key political question is how the power should be distributed in a future inter-network.

Today's discussion about *network neutrality* is about the distribution of power in the communication market. Depending on the political decisions and the technical capabilities the distribution is somewhere in between the two extremes outlined before. In addition to the private people and the network operators, companies, which provide their service via the Internet, appear as third fraction. Companies such as Goolge and Amazon sell their services "over-the-top" of the network operators, which provide just the "dumb" bit pipe.

Network neutrality is not a unified term and there are various definition used in literature. A common one refers to the situation in a "two-sided market", where the operators are in between the companies and the private people [ET12]. On the first side, operators sell Internet access to private people and companies. This side is present today and includes best-effort traffic. On the second side, operators buy services or content from companies and sell it to their customers (in particular private people). Thus, they bundle Internet access with specific services and content. Such bundling is known from the telephone market, where operators sell bit pipes in addition to their telephone service. However, it is rather new in the computer business. Examples in Germany are the bundling of Apples iPhone with a T-Mobile contract and Bundesliga football live streaming with a Telekom DSL [Deu] contract. Further examples are given in [Zit08].

Such a two-side market empowers the operators. They can bundle services and content with their bit pipe in a unique way. If no other operator can offer the same bundle, the bundling can be used to create monopolies. Moreover, it might get very frustrating for users if they have to decide for one bundle out of hundred different ones. The situation is in particularly critical since the bit pipe might be used by similar competing services not included in a bundle. In such cases, the services compete for the bit pipe. Since the bit pipe is under the control of the operator, the operator might favor its own service. Even more severe, the operator might forbid services, which are comparable to its own services (like VoIP), or slow-down undesired services (like peer-to-peer applications).

Consequently, the political questions are twofold. First, the politic has to decide if a two-side market is allowed at all. If operators are allowed to bundle services with Internet access, rules for such bundles are required to prevent monopolies. Second, politics have to decide about the rules how to handle competing services. This question seems to be the network neutrality question in a narrower sense. The answers to this question depend on which kind of neutrality is considered.

One definition of network neutrality refers to the content of packets. All packets and, thus, all services should be treated equally. A best-effort network seems to fit into this ideal, since it handles all packets equally without any traffic engineering methods. In essence, this form of neutrality assumes that the services are fair to each other. However, not even the UDP transport protocol is fair to TCP. More information about fairness is given in [MW00] and the references within. In a best-effort network a single user is able to balance its services by switching them on or off or by applying traffic engineering locally. Balancing the services of multiple users requires a fair distribution of the data rate among users. That seems to be challenging without any traffic engineering[11]. Thus, at least the last miles to customers with limited capacity might require traffic engineering mechanisms. The politic would have to decide about the rules for competing services in more detail.

[11] It seems that traffic engineering – as many other technologies, too – can be used for useful and dangerous things.

Another neutrality definition refers to the neutrality of offerings. According to this definition, a network is neutral if it offers QoS to everyone without discrimination [Cer06, CCR+09]. A proposal for a technical solution is presented in [DWJ09]. The political questions for a *neutral-offering* network concentrate on the price. For example, a free of charge best-effort access might be a suitable political statement to connect the whole society to the Internet. As for the prices for telephone calls or the SMS today, the politic – or an agent in form of a regulator – would have to limit prices for connections with requirements. Moreover, the regulator would have to control the fraction of capacity operators reserved for best-effort. Without such a reservation for best-effort traffic, operators would be able reduce the quality of best-effort by using all their capacity for high-price QoS connections. This reduction would "force" their customers to use non-best-effort requirements, even though, the service does not necessarily require it.

2.5. Dynamic protocol stacks

Traditionally, software engineers design network protocol stacks in a static way in order to optimize them for performance. For very high performance networks, stacks or parts of them are even implemented in hardware rather than software. That is especially true for link layer implementations, where the protocols have tight real time requirements. For example, IEEE 802.11 requires interframe spacing in the order of 10 to 50 µs. If performance is less critical, a software implementation has advantages regarding its flexibility. The flexibility is even greater, if the stack can be composed or changed at runtime. In the 1990th, a research field called "active networks" analyzed programmable network stacks. A survey is given by Tennenhouse et. al in [TSS+97].

In the last years, the relaxation of the layer boundaries led to dynamic network stacks. In contrast to today's monolithic stacks, dynamic stacks are composed by linking atomic functions. Such atoms are called building blocks, functional blocks, or function blocks. They are created at design time and provide predefined functions, which can be inserted in a stack at runtime. The set of functional blocks required for a data exchange depends on the requirements of the communication. Thus, the service access point to the outside world is important for dynamic stacks. It decouples the stack from the other components and provides the degree of freedom to change its implementation. Moreover, it enables the application to specify its requirements. The stack should provide a connection, which satisfies some requirements of the network or/and of the higher layer. Thus, the goal of the dynamic creation of stacks is the provisioning of the most efficient implementation for supporting these requirements. However, this requires a method for deriving the new composition of the stack from the requirements. Finally, runtime environments and tools for conducting the programming are research topics. All these issues are discussed in the following.

2.5.1. Service access points

A flexible stack has to be decoupled from the static software components. From a software engineering perspective, the programmable component has to be hidden behind an interface. This interface must encapsulate the component in order to enable changes of its implementation. The interface protects other components of being affected by different and changing implementations.

On end systems, the stack has to be decoupled from applications. Today's de-facto standard for accessing the local TCP/UDP stack is the Berkeley socket *Application Programming Interface* (API). It provides functions for exchanging stream-based data via TCP and functions for datagram-oriented communication based on UDP. For applications willing to provide a service to others, it offers the function `bind` to link its service to a port of TCP or UDP and a local subnetwork point of attachment. Even though it is possible to bind to any port, most communication partners expect the service to bind to its well-known port [IAN13]. That is caused by the `connect` function, which enables an application to start a communication with a remote one. It requires an IP address specifying the destination point of attachment and the port number of the service.

The *Berkeley socket API* provides only limited abstraction from IP and its transport protocols. A higher layer has to be aware of the different error and flow control protocols and of the addressing. Newer interfaces are more abstract and hide all these details. First, they restrict the `bind` and `connect` functions to names, which can be chosen by higher layers. Furthermore, they allow higher layers to specify their requirements for bound services and connections. An example is the GAPI [LVM+11], which was developed by the SIG "Functional Composition" of the G-Lab project. Current operating systems hide the details of IP and HTTP behind the functions usually used for local files. An example is the Windows API function `open`, which support URLs in addition to local file names. The proposal in [SM12] provides a definition of how to specify requirements for communications.

Relay systems have to support either a protocol, enabling the dynamic programming of its stack, or to provide a local interface for management applications. OpenFlow [MAB+08] is an example for accessing a programmable router. The OpenFlow specification 1.1 [Ope11] is mainly limited to program the router behavior based on Ethernet and IP header fields. The access to arbitrary octets of packets is not supported but might be subject of future versions of the standard.

2.5.2. Stack construction and selection

The ability to construct a stack dynamically requires an algorithm determining the layout of the construction based on optimization goals and construction constraints. Since a communication includes at least two (remote) peers such a construction has to be done on both sides. The peers have to exchange different data depending on the algorithm used to construct the stack layout. If the algorithm is deterministic, it might be sufficient to signal only its input values and not the layout itself. If both peers may calculate different layouts, the whole layout has to be exchanged.

Approaches to these problems differ in the point in time of the construction decision and complexity aspects. The time aspect defines when decisions about the layout are made. In order to reduce the complexity, some approaches such as the *Netlets* [VMEK+09, VMW+09] decide about the layout at design time. At runtime, an algorithm selects one of the already constructed stacks. "Template approaches" [SKM12] define a stack template at design time, which includes a limited set of variability. At runtime, only the variable part is configured to satisfy given requirements. Depending on the degree of freedom of this flexible part, such template approaches can trade flexibility for setup time. The more flexibility a template includes, the more complex is the decision at runtime but the better the adaptability to requirements. Without a fixed template part, the whole complexity of the layout decision has to be handled at runtime. *Service Oriented Node Architecture* (SONATE) [KSRM12, MR08] is a framework for full dynamic and template based stack construction.

2.5.3. Stack runtime environment

In addition, most approaches require some kind of "runtime environment", in which the stacks are constructed and in which these stacks can operate. A prominent example for such a framework is the *Click* modular router [KMC+00]. It provides a framework for developing functional blocks, which can be composed to a stack. Its main focus is on the development environment and the design-time of a stack and not on dynamic composition.

The SILO architecture is an example for a dynamic composition, which focuses on cross-layer issues [DRB+07]. In contrast to SONATE's composition and selection algorithms, it focuses more on the ability to provide a set of functional blocks and their configuration. Its authors do not discuss the method of how to derive the set of blocks from the requirements. Similar, the *Autonomic Network Architecture* (ANA) [KHM+08] focuses more on the grouping of functional blocks in "compartments" and the resolution of addresses within.

3. Forwarding on Gates architecture

FoG enables a flexible and scalable network that supports different locations for arbitrary functions. It aligns functions traditionally located in stacks on end systems and functions representing transmission capabilities within networks. FoG achieves that by operating in a world constructed solely of instances of functions, which are called functional blocks. Examples for functions are encryption, retransmission, video encoding or transcoding, ordering, prioritization of packets, best-effort transmission, transmission with minimal data rate guarantees, packet duplication for multicast, and virus scan. They take packets as input and perform their operations on them.

Each functional block is hosted by a function provider and is used by function users. Both roles can be adopted by end and by relay systems. Moreover, a system may have both roles at the same time. The distinction of these roles is crucial, because each role comes along with different tasks. A *function provider* provides the memory and CPU resources required by a function. A *function user* knows about a functional block and decides which packets should use it. Only the creation of functional blocks requires negotiations. Afterwards, function users can assign packets of any connection to a functional block without having to negotiate with the function provider. A function user implements the assignment by encoding its decision within each packet that is send to the function provider. Figure 3.1 shows an example setup, in which a relay system of a connection between the applications C and S decides about the usage of the functional block X based on the requirements given from the client application C. The relay system acting as function provider is notified about the decision but not about the requirements. The importance of both roles is exemplified with a use case, which is described in Section 3.1. It lists the requirements of function providers and users regarding flexibility and scalability.

The separation of tasks between function users and providers decouples the creation of functional blocks from the decision about their utilization. A system can request the creation of a functional block and adopt the role of a function user independent from the management of connections. This enables a pro-active creation strategy for functional blocks that reduces connection setup delays and allows even the setup of connections without initial signaling.

FoG is flexible regarding the assignment of roles to systems and, thus, flexible regarding the placement of states. Function users and providers have to store different states in order to fulfill their tasks. In particular, the function users have to store mapping states, which define the functions the packets of a

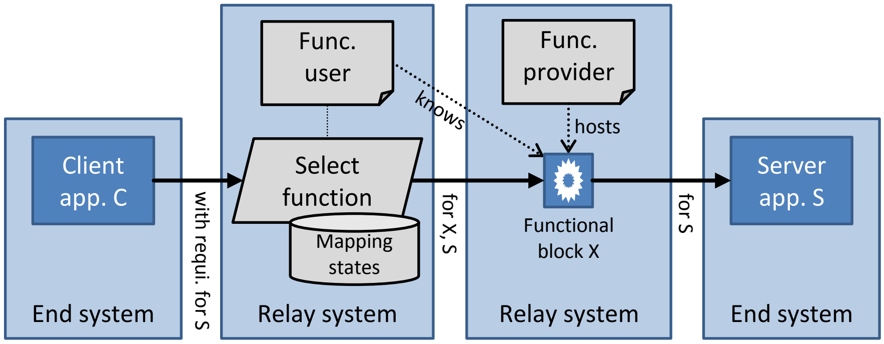

Figure 3.1.: A function user decides to include functional block X to the chain of a connection between the applications C and S. The stream of packets of the connection is shown with thick arrows. Meta-data included in the packets is listed between the systems.

connection should use. FoG can move states by delegating roles from one system to another. In particular, a system can delegate the role of a function user and, thus, delegate states and the power of decision. FoG can adjust the initial placement of states flexibly by network policies. Since each network operator can define its own policies, the autonomy of subnetworks in an inter-network is preserved.

Functional blocks are concatenated to chains in order to combine their functions. Packets of a connection are mapped to chains that satisfy the requirements of the connection with their combinations of functional blocks. A communication model, which is described in Section 3.2, defines the rules for the creation of chains. Its key aspect is the differentiation between two types of functional blocks: Forwarding nodes and gates. The formers take over the multiplexing function and the latters represent all other functions. This differentiation enables the creation of chains by an external entity that does not need to know the details of the implementation of a function.

The FoG layer architecture combines the parts mentioned before to a holistic solution for large inter-networks. It is an architecture for a layer with inter-network scope in a recursive reference model. This reference model provides an enhanced layer encapsulation that enables the implementation of the new ideas within a single layer. FoG's separation between function user and provider and its communication model influence the logical components of a layer such as the transfer and the routing service. Section 3.3 describes these components and their interworking.

The related work is reviewed within the individual sections. Other motivations for a future Internet are outlined in Section 3.1.4. Alternative or similar communication models are discussed in Section 3.2.4. Finally, related architectures or proposals for architecture building blocks are presented in Section 3.3.8.

FoG is not just a combination of existing technologies such as IP and MPLS but supports a real superset of scenarios. In order to support this claim, two aspects are analyzed in Section 3.4. First, it shows the implementation of a scenario that is not supported by traditional architectures with FoG. Therefore, it picks up the use case introduced in Section 3.1.1 and discusses its implementation with FoG. Second, the ability of FoG to emulate the behavior of IP and MPLS networks and, thus, to support the scenarios supported by them is shown.

Due to the economic and political importance of the Internet, an inter-network architecture is not only influenced by technical aspects. The socio-economic consequences of an architecture are important as well. Section 3.5 reviews the political aspects of FoG in the context of the "network neutrality" debate. Business aspects are highlighted in Section 3.6, which takes a closer look at deployment and interoperability issues.

The chapter closes with a discussion of the architecture in the context of the use case and of the reference model. Section 3.7.1 picks up the design questions from the use case and discusses the fitness of FoG for the use case. The advantages of the reference model and the differences between FoG and the reference model are outlined in Section 3.7.2.

Functions in a network? Sounds like an old telco ad.

I use this phrase to highlight that functions are not located on end systems. From the perspective of a recursive layer, however, the functions are all within a FoG layer and located in transfer service entities. From an architectural standpoint, these entities are all the same. There are no special entities for "hosts" and "routers" (but most probably there will be special implementations for both). Thus, the phrase refers to the traditional reference models and should make it easier to understand this text without knowing the recursive ideas.

3.1. Motivation and design

The histories of the reference models show their strong dependency on the environments existing when they were created. Although the recursive model directs the development in a new and promising direction, historical conflicts are still present. Most important is the conflict between connection-oriented and connectionless approaches. FoG proposes a middle ground between both by a small shift of the perspective. The perspective shifts from a "node-" or "route-based" view to an "edge-based" one. A FoG-route is neither defined

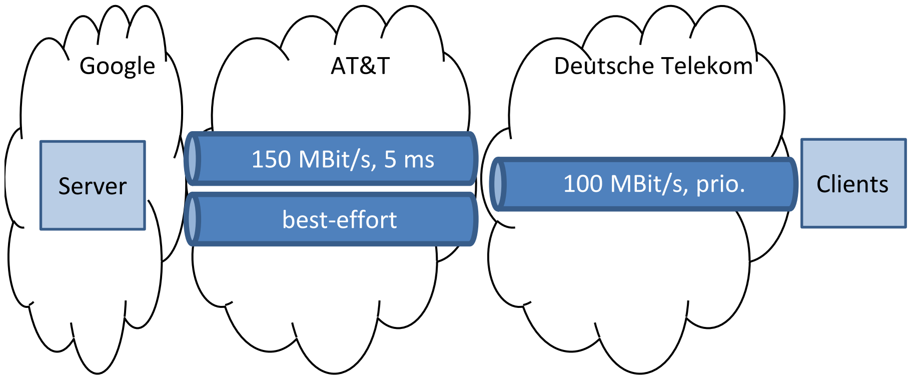

Figure 3.2.: Sketch of motivating use case with the two function providers AT&T and Deutsche Telekom and the function user Gogle.

by a single (destination) node nor a list of nodes as assumed by IP or RINA. Neither is it defined by a complete list of concatenated edges as required by connection-oriented systems. FoG defines a route with at least one node and one edge. However, the node is just a means to an end to direct a packet to an edge. The edge is more important than the node, which is just required to direct the construction and to bridge gaps in the knowledge about edges.

In the following, a use case starts the discussion why such an "edge-based" view is reasonable. Afterwards, Section 3.1.2 points out the problems of state-of-the-art solutions to support the use case. Finally, Section 3.1.3 highlights the main design decisions for FoG.

3.1.1. Use case with Google and Deutsche Telekom

An example with a typical application provider and a network operator may look like the one depicted in Figure 3.2 and described in the following: The application provider Google requests a higher priority for some of "its" connections from a transport provider like Deutsche Telekom. Google plans to improve the QoS, for example, for its YouTube video streaming. The higher priority should ensure a better QoS for the videos on the resource-limited last mile to the end systems in the network of Deutsche Telekom. Deutsche Telekom has to limit the usage of this function, in order to prevent its network from being overwhelmed by high priority traffic. In this example, the throughput of the function is restricted to 100 Mbit/s. In other words: The function user Google requests a function – prioritized packet relaying with a maximum throughput – from the function provider Deutsche Telekom. In addition, Google combines the function provided by Deutsche Telekom with other functions from other providers. For example, Google uses a transit relay function that ensures 5 ms delay for a maximum of 150 Mbit/s from AT&T. Google concatenates both functions in order to direct its connections through the network of AT&T to the assignment of a higher priority in the network of Deutsche Telekom.

The following questions structure an evaluation of solutions for the use case:

- **Arbitrary functions:** Can Google request an arbitrary function from Deutsche Telekom? In the use case, Google requests a different relaying function from each function provider. It should be possible to request not only relaying functions but arbitrary ones such as video transcoding, loss handling, virus scanning, and encryption.
- **Flexibility:**
  - **Combining functions:** Can Google combine several different functions from the provided set? Google has to assign a connection to a set of different functions. The use case includes a combination of a relaying function with guaranteed non-functional properties with a relaying function with rather vague non-functional characteristics. More sophisticated combinations with, for example, application-related functions should be possible.
  - **Location of functions:** Can Google choose a specific location and, thus, a function provider for a function? Google should be enabled to select function providers even if their subnetworks are not directly adjacent. In the example, Google requests a function from Deutsche Telekom without further support or negotiations with the relaying subnetwork of AT&T. In general, the function user should be able to decide which connection uses which function from which provider.
  - **Autonomy of providers:** Can Deutsche Telekom decide about the set of supported functions independently from others? Deutsche Telekom must have the ability to influence the location decision of users by limiting possible locations (e.g. to a subset of its systems) and by limiting the set of supported functions.
- **Scalability:**
  - **States required for relaying:** Does Deutsche Telekom has to store connections-specific states for relaying? The function provisioning must scale with the number of connections using a function and, thus, connection-specific classification states (definition in Section 2.1.4) must be avoided.
  - **Signaling:** Does Google have to notify Deutsche Telekom of each new connection that is using a function (e.g. is assigned a higher priority)? The omitted notification reduces the delay for setting up

new connections because the exchange of signaling messages before a data exchange is avoided. For Deutsche Telekom, the omitted signaling reduces the computational load for processing signaling messages and updating states.

3.1.2. Possible solutions with today's Internet

Today's network operators focus on best-effort relaying and do not offer function user – end users or other operators – mechanisms to request functions as required by the use case. In most cases, peerings between subnetworks include best-effort service level agreements only. However, the extensions to the Internet presented in Section 2.4 offer some possibilities to implement the use case. In the following, the fitness of the extensions for the use case is discussed. The discussion assumes that the extensions are deployed in a network. Table 3.1 shows a summary. After these protocol related solutions, drawbacks of today's interfaces are outlined in the last part of this section.

IntServ and other similar approaches define the mapping of packets (via connections) to functions with classification states, which are established with signaling messages. For RSVP and IPv4, such a classification state consists of at least source and destination address and of the protocol field and sums up to a minimum of 9 octets. Typically, classification states are created for each connection separately, which limits the scalability significantly. The use case requires mechanisms to request functions for relaying with non-functional properties without support from relay subnetworks. This is already supported by most QoS signaling protocols, like RSVP or NSIS. However, the request of arbitrary functions, such as video transcoding or encryption, is not supported. That requires new signaling protocols or extensions to existing ones. Since most signaling protocols are designed to be extensible, it seems to be possible to add this feature. Since signaling protocols such as RSVP are optimized for requesting functions for connections and since IP routing is involved, the user cannot control the location of functions. The providers direct traffic to their function locations on their own.

Aggregated classification states are suitable for IntServ, if connection-specific classification states induce too much overhead. For example, Google could signal an IP prefix to Deutsche Telekom, which identifies the source IP addresses of its video servers. Deutsche Telekom could setup this prefix as classification state in its ingress routers. However, this limits Googles autonomy in addressing its servers. Moreover, it may be problematic to determine such a prefix in environments, where load is distributed dynamically among several systems. An example is a cloud environment, where the server application is distributed transparently over a large set of systems. If multiple services are provided by such a cloud, the prefix does no longer identify a single service. Thus, the classification state in the network of Deutsche Telekom would lead to

Possible solutions	**Arbitrary functions**	**Flexibility**			**Scalability**	
		Combining	Location	Autonomy	States	Signaling
IntServ	(+)	+	−	+	−	−
Aggregated states	(+)	o	−	+	o	o
DiffServ	o	−	−	−	+	(+)
MPLS & IP	−	+	o	+	o	o
OverQoS	o	+	(o)	−	−	−

Table 3.1.: Fitness of possible solutions for the use case from an inter-network perspective. A plus indicates a good fitness, circle an average one, and a minus indicates a non-fiting solution. Brakets indicate that a classification bases on some assumptions mentioned in the text.

a wrong mapping for, e.g., web server traffic. Since YouTube videos as well as web pages would be transmitted from source addresses with the same prefix via TCP and port 80, they are undistinguishable for RSVP. Thus, it might be impossible to combine arbitrary functions.

DiffServ [BBC^{+}98] stores the classification state in a header field, such as the TOS field of IPv4, in order to inform relay systems about it. Today, the TOS field is mainly used within a subnetwork because there is no common agreement for the TOS field values for inter-networks. Since the RFCs 791 [Pos81] and 2474 [NBBB98] are not compatible, a network may have to translate the values at its gateways. If all operators of an inter-network agree on a homogenous interpretation of the values, DiffServ might be used in inter"-networks[1]. However, the alignment of values unifies the set of functions and limits the autonomy of providers. The limited size of the TOS field restricts the number of functions that can be provided by a network. Moreover, the use case combines two different functions (guaranteed relaying in the AT&T network and prioritization in the network of Deutsche Telekom) requiring different TOS field values in different parts of the route. Such a setup is not supported by DiffServ. DiffServ in an inter-network context raises additional authorization problems. If DiffServ is used in a subnetwork, the gateway assigning packet to classes is under the control of the same operator as the other relay systems of the subnetwork. If used in an inter-network, the entity assigning a packet to a

[1] Subnetworks not using DiffServ internally would have to map the DiffServ classes to their internal mechanism implementing the function.

class is not identical to the entity providing a function. For example, Deutsche Telekom would require some more information in addition to the TOS field value in order to figure out that traffic has been classified by Google and not by AT&T or some other subnetwork. In the worst-case, Deutsche Telekom would require information about connections in order to enable an authorization. If this problem is ignored, DiffServ does not require signaling.

Since the TOS field of the relaying PCI of IP seems to limit possible solutions, combinations with other relaying PCI formats may be more suitable. A combination of MPLS and IP provides more flexibility with respect to a separation of routing and relaying. IP takes over the routing and MPLS the QoS-related relaying of packets. Arbitrary functions are not supported. The label stacking of MPLS provides the ability to identify different relaying functions. However, the combination limits the switching between IP and MPLS. With an IP packet encapsulated in MPLS, it is possible to execute a defined sequence of relaying functions as specified in the MPLS label stack before IP routing takes over. The MPLS labels, which should be used after the IP routing, cannot be specified directly. Figure 3.3 illustrates that the classification states on the last IP relay system B are required to map a connection to the next LSP number 3. Google is not able to construct a packet, which is sent via IP to the network of Deutsche Telekom and uses a specific MPLS label for the prioritization function afterwards. In general, the solution specifies the functions to be executed before IP routing is done. The labels for the functions, which should be executed after the IP routing (the subsequent labels for MPLS), have to be stored on the last IP relay system. Consequently, this relay system requires classification states, which specifies which packets should be relayed to which subsequent functions. Again, these states require signaling in order to establish them. However, the first explicit route gives the function user more control over the functions and the selection of function providers.

Other workarounds using source routes with IP have security constraints. [Rei07, ASNN07]

OverQoS is a typical overlay approach that introduces functions not supported by IP. It is capable of executing a limited set of functions (in this case: smoothing losses, prioritization, and bandwidth control)[2]. OverQoS allows the combination of functions from its set, but seems to assume that all overlay nodes support the same set of functions. The routing and, thus, the placement of functions in the overlay are not addressed by the OverQoS proposal directly. In general, an overlay requires the end systems to be aware of the overlay and to address the relay systems of the overlay. This adds addressing and routing functions to the application and, thus, increases the complexity of the application. Frameworks, such as *SpoVNet* [BHMW11, WCBH$^+$08], take over such burdens. The split between routing in the underlay and function

[2] Other overlays are dedicated to other functions such as multicast.

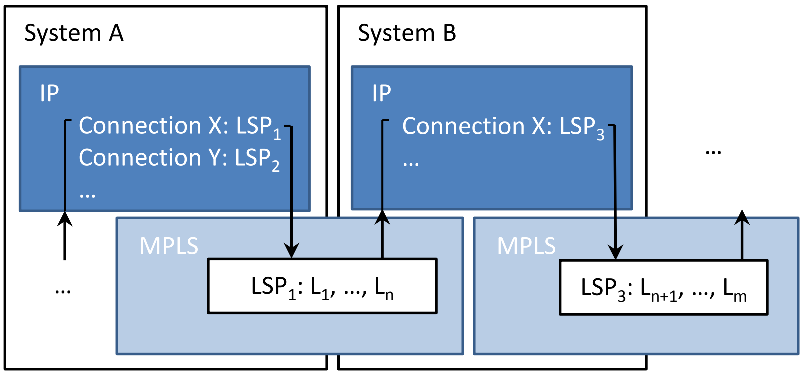

Figure 3.3.: Classification states required by IP to map connection to subsequent MPLS LSPs

placement in the overlay limits the placement of functions since nodes that are not part of the overlay are not available as location. For similar reasons, routing in the overlay might be sub-optimal since it is not aware of the graph of the underlay network. Such problems are known from peer-to-peer overlays as well [RB11]. The stacking of an overlay on top of IP suffers from the same state distribution limitations as the stacking of MPLS and IP mentioned before. Moreover, OverQoS requires a per-flow state setup on relay systems (cp. Figure 4 in [SSBK03]).

Besides the protocol issues, current stack interfaces limit the ability of applications to specify requirements for connections. In most cases, applications have to use an interface tailored for a specific set of functional requirements. For example, the most common interface – the Berkeley socket API – offers different functions for stream- (TCP) and datagram-based (UDP) communications. Without proper information about the requirements, it is difficult (or even impossible) to decide about the assignment of a connection to functions. Even if the requirements are known, a lack of deployed signaling protocols hampers the usage of the requirements on relay systems.

With proper interfaces in place, the stack on end systems can be dynamically adjusted according to the requirements. As discussed in Section 2.5, such solutions mainly focus on the end systems and use the existing IP network to relay packets between both. If relay systems are considered, the usage of IP or MPLS/IP raises the same problems as discussed before.

3.1.3. Conclusions for new design

The key issues for a solution supporting all aspects of the use case are

- the distribution of states between the function user and provider and
- the differentiation between requesting functions and using them.

Figure 3.4 shows three different state distributions including the information exchanges between function users and providers. The upper two distributions A and B are from the previously presented solution possibilities. The distribution C is the one I propose.

A) Concentrated states: The function provider plays a dominate role. It stores all mapping and function states and has to be contacted for all changes and every new connection of a function user. This concentration leads to the scalability problem regarding the number of states and the amount of signaling. The IntServ approach is an example for this distribution.

B) Moved parameters: DiffServ moves the classification states and the priority parameter from the function provider to the user. Packets contain only the function parameter in the TOS field[3]. The function itself – packet type differentiation – is selected implicitly.[4] This single header field is used for all functions. Multiple parameters or different functions in various subnetworks are not supported as stated in the previous section. However, the idea of splitting responsibilities and to transmit the decision of the function user to the function provider encoded in packets is promising.

C) Moved mapping states: The function user should be enabled to decide not only about the parameter but about the functions and their parameters. As shown in the lower part of Figure 3.4, this requires the selection of functions in addition to the parameters at the function user as well as the transmission of this selection to the function provider.

The flexible placement or movement of mapping states is the technical mechanism to implement the architectural delegation of the usage decision from the function provider to the function user. Moreover, it leads to a scalable system since the function user takes over states from the function provider. The function provider can offer functions without having to store the mapping states. Furthermore, the movement enables the function user to decide about the functions and their parameters used by connections. However, not all these decisions are equally important for the function user. For Google, the decision about the usage of the prioritization function in the network of Deutsche Telekom may be much more important than the decision about the

[3] The "differentiated services field" according to RFC 2474 [NBBB98].

[4] If the bits of the TOS field are used as flags, they might select functions explicitly. However, the approach supports only a limited set of pre-defined functions supported by all nodes. Thus, it can be considered as one big function with one parameter.

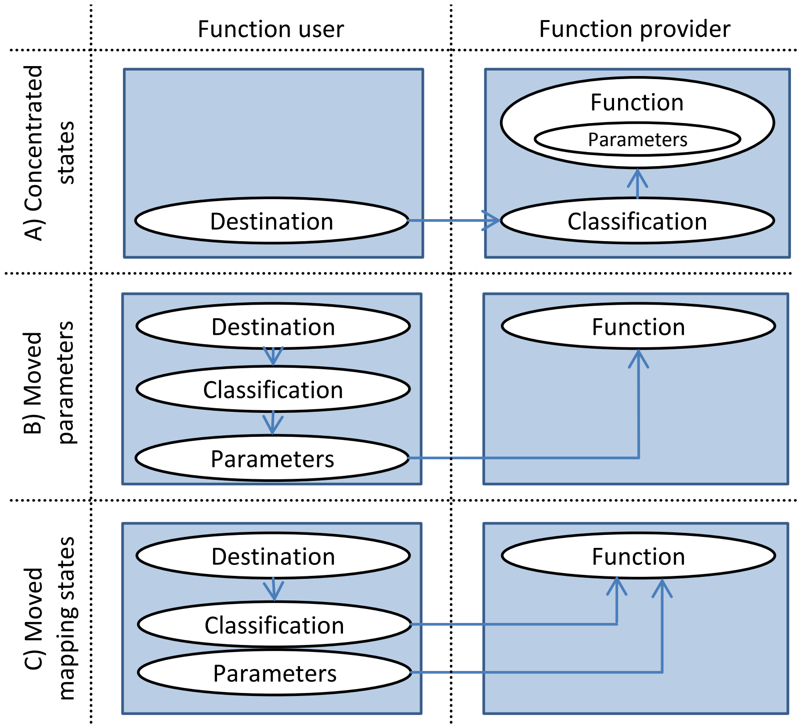

Figure 3.4.: Three state distributions: (A) the concentrated setup with all states at the function provider, (B) the setup with moved parameters and implicit function selection, and (C) the setup with moved mapping states with explicit selection of function and parameters. States and other information (the destination name) are depicted as ellipses in the box of the function user or provider depending on where they are stored. Arrows indicate which information is used to derive which other information.

functions used in the AT&T network. Google can, for example, decide about the usage of the prioritization function on its own, but it can leave the decision about the functions used in the AT&T network to others (e.g. to AT&T itself). Consequently, a single function user is not the only contributor to a connection. Multiple function users decide about the concatenation of functions for a connection. In the example, AT&T as well as Deutsche Telekom may want to influence the decision about functions used for a connection as well. The former may decide about the function implementing the transit relaying and the latter may decide about subsequent functions checking for viruses and relaying packets to end systems.

This flexibility is implemented by FoG with partial routes. A partial route contains the already selected functions as "edges" and bridges "gaps" between them by specifying node names. Multiple function users can decide whether they contribute functions they know about to a route. By filling in more and more edges, gaps are shrinking and, in the end, are filled up. For example, Google can begin the construction of a route by bridging the gap between its network and Deutsche Telekom by the gateway name of Deutsche Telekom, by specifying the prioritization function, and, finally, bridging the gap to an end system by specifying its name. Afterwards, AT&T has to fill the first and Deutsche Telekom has to fill the second gap.

In order to transport these decisions, relaying PCI formats have to include fields for storing decisions of multiple heterogeneous subnetworks. The problems surrounding the TOS field show that a short priority field is not flexible enough to encode such decisions. As the example of a combined IP and MPLS network shows, the key issue is a flexible change-over from route parts not strictly defined and fixed route parts. The former requires a destination address as used by destination-based relaying PCI formats. The latter requires a route as in route-based relaying PCI formats. Consequently, a hybrid relaying PCI format is required.

Routing and route discovery for hybrid relaying PCI formats differ significantly from Internet routing. It requires more information about the network and – at least for connections having complex requirements – different routing algorithms. Consequently, the architecture has to ensure that the information is provided and that the routing mechanism has the flexibility to use different algorithms or strategies depending on the context.

Each subnetwork should be as independent as possible. An operator should be able to adjust the setup of its subnetwork such as policies, functions and addressing without relying on other subnetworks to support or even permit such changes. The architecture has to ensure this autonomy and has to enable a heterogeneous set of subnetworks. Since a totally independent subnetwork might not be able to communicate with others, the architecture has to balance the degree of freedom of subnetworks and the features of an inter-network connecting these subnetworks.

The networking history shows that there is always a new feature not included in the original design. This is especially true, if application related functions are "moving" from end systems to relay systems. Therefore, an architecture should restrict the functions offered by networks as less as possible. Instances of functions called functional blocks are introduced as level of abstraction[5] in order to support arbitrary functions in general. The architecture just operates with them and puts little restrictions on how functional blocks look like internally.

Does the abstraction solve anything?

Or does it move the problems under the rug? First, the abstraction is useful to align the handling (e.g. management) of elements covered by the architecture. Second, by introducing the abstraction, the architecture acknowledges that algorithms for solving problems such as the mapping from requirements to functions are required. Therefore, it provides a "logical place" for these algorithms (e.g. the architecture enables the routing algorithm to access the information required by the algorithm). And third, it does not restrict the implementation of such algorithms. A network offering just a hand full of functions and supporting only a limited set of requirements does not require full featured algorithms. An algorithm could just hard-wire a mapping from requirements to functional blocks. As the implementation in Section 4.4.3 shows, some rules, like "if lossless transmission is required, then use retransmission function" might be sufficient for some. However, the architecture supports more sophisticated approaches handling an unlimited set of dynamic functions.

The research on dynamic stacks, presented in Section 2.5, focuses mainly on flexible functions on end systems. FoG shifts the focus to relay systems, which emphasizes two additional aspects. First, the functions that are needed to satisfy requirements have to be distributed among multiple relay systems. The problem of where to locate which functions arises. Second, the creation and re-usage decision for functional blocks has to respect the policies of relay subnetworks. As discussed before, this raises the problem of balancing the relationship between function user and provider. In summary, the architecture has to make sure that the implementation of algorithms for these problems is feasible.

Function provisioning may become a business. Function users might have to pay for the functions they are using. This seems to be a reasonable assumption especially for the functions that are costly for the provider (e.g. video transcoding), most useful for users, or both. As the telecommunication market itself,

[5] Famous saying in computer science: "All problems can be solved by adding one more level of indirection."

What about virtualization?

Virtualization is a nice tool for exploiting the multiplexing gain for resources provided to multiple users. As such, virtualization might help to deploy FoG implementations in a real network. The virtualization would provide a guarded environment and enable a test of the implementation without conflicting with other protocols running in parallel. However, FoG itself does not use virtualization techniques. Nevertheless, it might include virtualized elements in an (inter-)network by representing them as functional blocks.

the "function business" might be subject to regulation issued by democratic processes. Section 3.5 discusses this issue in more detail. The implications for a design are twofold. First, the design should not force a payment for function users. Second, it should not prevent payment, if it is required. A prerequisite for accounting is authentication. Without authentication, a function provider will not be able to account someone for its usage.

In addition to accounting, authentication is also required for authorization. Authorization is required in two different contexts. First, a function provider can decide to authorize some entities to create new functional blocks. The decision can be influenced by the level of trust the provider associates with an entity and by the amount of resources required to create a functional block. A possible rule is the following: The less resources are requested, the lower the minimal trust threshold can be. Second, the function provider has to decide if an existing functional block is allowed to be changed or removed by an entity. In typical situations, the decision is positive if the entity is the "owner" of a function. The owner is the entity that had requested the creation of the function.

In summary, authentication is the important mechanism that must be supported by the function provider as well as the function user. Thus, it should be supported by the architecture. Authorization and accounting are optional mechanisms that do not need to be part of an inter-network architecture. They belong solely to the domain of a function provider and can be built on top of an authentication.

The reference models, and thus, today's networks, seem to provide little room for real inter-networks. OSI introduces the inter-network layer by introducing three sublayers for the network layer. Due to the late introduction of these sublayers, this had no practical consequences. IP tends to ignore the inter-network layer as well. It re-introduces the inter-network aspects through several "back doors". For example, "back door" QoS solutions are proposing "two-tier" architectures splitting the network layer into a subnetwork and an inter-network layer. NAT splits the addressing in a subnetwork and an inter-network part. The *Locator/ID Separation Protocol* (LISP) [FFML13] does it the other way around and introduces different addresses for the inter-network. Mobility solutions are classified as micro- and macro-mobility. The former handles the subnetwork aspects while the latter the inter-network ones. In summary, recursive layers provide a much better theoretical background for the design of an inter-network layer. In particular, the name- and requirements-based interface of a recursive layer provides a much better encapsulation of a layer. This encapsulation provides a suitable basis for the new model and the flexibility of FoG.

3.1.4. Related motivations

With the advent of the Future Internet research programs, multiple statements about the motivations for a new inter-network have been made. Some of them are relevant for FoG as well.

David Clark et al. [CWSB02] discuss *tussles* between groups with contradicting political or business goals. They argue that an inter-network has to be designed "for variation in outcome, so that the outcome can be different in different places, and the tussle takes place within the design, not by distorting or violating it" [CWSB02]. Some of the tussles concern the distribution of power between users and providers. The differentiation between function user and function provider can be used to balance this tussles as discussed in Section 3.5. Moreover, they identified authenticity as a prerequisite for establishing trust. The authentication service of FoG presented in Section 3.3.4 provides a frame for implementing this.

Trossen summarized the discussion of the EIFFEL think tank in [Tro09]. The think tank opts for a design that supports tussles. It identified problems regarding the "robustness to failures", resource accountability, and inflexible interfaces between applications and network stacks. FoG addresses reliability with three repair algorithms described in Section 4.3.4, accounting with its authentication service, and the interface issues with its recursive interface defined in Section 3.3.1.

Feldmann acknowledges the importance of business aspects as well. She points at scalability problems regarding routing in the Internet, missing mobility support, reliability issues, and states that "it is still unclear how and where to integrate different levels of quality of service into the architecture" [Fel07]. Since FoG can combine multiple different levels of QoS within a single chain of functional blocks as described in Section 3.2, it seems to be prepared for use cases with diverse non-functional requirements.

Some "visions" for future networks and their features are outlined by Clark et al. in [CPB+05]. However, no researcher seems to be aware of a "killer application" for a future inter-network.

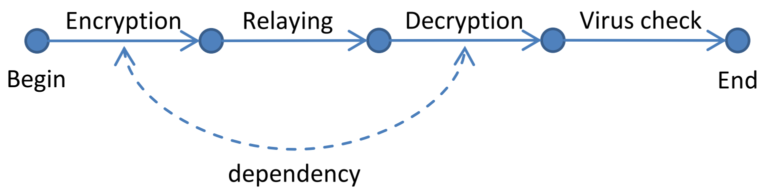

Figure 3.5.: Example chain of functional blocks. Gates are depicted as edges and forwarding nodes as dots. Dependencies are shown with dashed arrows.

3.2. Communication model

FoG builds upon the idea of using arbitrary functions for packet transmission. Instances of functions are called functional blocks or just blocks. From an object-oriented point of view, functions relate to classes and functional blocks to objects. Only functional blocks are accessible for packets. Functions not represented by blocks are not accessible. Section 3.2.1 defines functional blocks in detail.

Functional blocks can be concatenated to chains in order to combine multiple functions. Thus, the requirements of a connection are satisfied by concatenating a suitable set of functional blocks to a *chain* of functions that all packets of a connection have to pass. For example, a uni-directional connection that should be encrypted requires at least three functions. First, a packet has to be encrypted. Second, it has to be transmitted from the source to the destination. Third, it has to be decrypted again. Figure 3.5 shows a small example of such a chain. The order is important, since the encryption block has to be executed before decryption block. The construction process for chains has to take such dependencies between blocks into account.

Two classes of functional blocks called gates and forwarding nodes are introduced. The forwarding nodes are responsible for multiplexing. The gates represent the "real" functions doing the productive work. The classification ensures that functional blocks can be depicted and handled like a graph. Forwarding nodes are the vertices of a graph and the gates are edges linking them. In contrast to other approaches, this gives the edges of the graph – the gates – an important meaning. The forwarding nodes – the vertices – provide their multiplexing function, in order to enable an "outside" decision entity to select a chain of blocks. This is the main differences to other approaches. Further differences between the FoG communication model and related research work is discussed in Section 3.2.4.

Each connection can be mapped to its own exclusive set of chains. However, the model allows the reuse of a block for multiple chains. For example, a block representing the transmission of packets between two systems can be used by all connections between these systems. Blocks can only be reused for multiple chains, if all their dependencies are fulfilled in each chain. Details about the "chaining" and the reuse prerequisites are given in Section 3.2.2.

Afterwards, Section 3.3 introduces the FoG layer architecture, which provides a frame for implementing such a communication model in a distributed fashion.

3.2.1. Functional blocks

A *function* is an action a network supports. Such functions include classical network functions, like sending a packet to a peer or calculate a checksum. In addition to such classical functions, FoG allows arbitrary functions such as video transcoding and encryption. A *functional block* is an instance of a function with a specific location, a set of parameters, and states. A block takes packets as input and performs its function on it. It has zero or more outputs, which can be used to relay a packet to the next functional block attached to it. For example, a block can represents a function that sends FoG packets over Ethernet to a specific peer identified by a MAC address that is a parameter of the block. Another block can encapsulate the encryption of packets according to the *Data Encryption Standard* (DES) and a given key.

FoG classifies functional blocks in two categories according to their ability to decide about the next functional block a packet has to travel through. In other words, they are classified according to their number of outputs:

- *Gates* are functional blocks that do not decide about the next functional block. They have exactly one output and, after processing packets, relay them to it. Gates are expected to perform potentially complex or costly operations.
- *Forwarding nodes* are functional blocks having a variable number of outputs. Each output is assigned a *gate number* that has to be bijective within the scope of a forwarding node. For each packet, a forwarding node has to decide about the output a packet is relayed to and, thus, about the subsequent functional block. This multiplexing function depends neither on the size of the packet nor on the data transported by the packet. Forwarding nodes are limited to the multiplexing function and not allowed to execute arbitrary functions like gates.

3.2.1.1. States

Each functional block stores its function states, which are required to execute and manage the function. Gates representing QoS-enabled links in a network

How long is a gate number?

The length of a gate number in bit depends on the implementation. As shown by the implementation described in Chapter 4, it is reasonable to define a rather small length (e.g. 8 bit) and to cascade forwarding nodes logically, if longer numbers are required. Thus, a FoG entity can use gate numbers with a length of $x * 8$ bit internally. From the outside, these long numbers appear as x gates and $x - 1$ forwarding nodes in between. In summary, the length of a gate number is a minor issue and it is possible to combine as many as required.

have to store, e.g., timers, authentication information, priorities, queues, and token bucket counter. Other gates may store the last I-frame of a video stream, codec parameters, and keys.

Functional blocks neither store the mapping from packets to connections nor the mapping from connections to sequences of functional blocks.These classification states are delegated to a chain. A chain is a stack of gate numbers as defined in more detail in Section 3.2.2. Moreover, a functional block can "export" its parameters to the chain in order to avoid their local storage. The parameters are encoded in the chain as one or more gate numbers. The gate removes these numbers from the chain information when a packet passes by. An example for parameters stored in a chain is given in the interoperability example in Section 3.6.2.

3.2.1.2. Non-functional properties

The performance of functional blocks is described with non-functional properties. For classical functions transmitting packets to neighbors, non-functional properties refer to the QoS provided by the technique used for transmission such as delay and data rate. For other functions, the non-functional properties refer to the performance of a local operation. For example, the delay of a video transcoding function refers to the constant overhead of the transcoding operation while the data rate refers to the computational delay according to the video frame size. The same holds for other functions such as encryption or calculating check sums. If these non-functional properties are not known, best-effort behavior is assumed.

3.2.2. Chaining functional blocks

Functional blocks are concatenated to chains in order to provide a set of functions to packets of connections. The concatenation has to take dependencies between blocks into account. They influence the order of blocks and the type of blocks in a chain. Some functional blocks can be reused for multiple chains in parallel in order to reduce the number of required blocks.

A *chain* is a sequence of functional blocks, like the one shown in Figure 3.5. It is defined by a start forwarding node and a stack of gate numbers. The sequence is defined by the stack and does neither depend on the state of the blocks nor the content of a packet. The stack of gate numbers is sent along with a packet and contains gate numbers for each forwarding node in a chain. Each forwarding node removes the topmost gate number and determines the output based on this gate number. Afterwards, the packet is relayed to the functional block attached to this output. If the stack of gate numbers is empty, the packet reached its destination forwarding node. Thus, only forwarding nodes can represent connection end-points and can be used to interact with applications.

From a more abstract perspective, forwarding nodes provide their multiplexing function to an "outside" decision entity. The decision entity defines a chain of gates by defining a suitable stack of gate numbers. It can specify a chain without the functional blocks knowing about that.

A chain is not equivalent to a connection. Chains provide the functions required by connections; and connections are assigned to one or more chains. There might be several connections mapped to a chain or parts of a chain. Moreover, chains can be reused for subsequent connections, if they are no longer required by the previous connection. The separation between both enables a pro-active establishment of chains.

3.2.2.1. Dependencies

A gate can depend on other gates or functions that are required for its operations. These two types of *dependencies* are described in the following:

- Dependency to a specific gate: A gate requires a specific other gate, if they share states. Depending whether the dependency apply to the same chain or to a reverse chain, the following two cases occur:
  - Same chain: A gate requires a *peer gate* in the same chain. For example, a gate checking for lost packets requires exactly one other gate numbering the packets as a predecessor in a chain. A gate adding a header to the packet requires a peer gate that removes this header again.
  - Reverse chain: A gate requires a *reverse gate* in the *reverse chain*, in which packets travel in the opposite direction. For example, the feedback from the gate checking for lost packets has to reach the gate with the packet buffer in order to request a retransmission. Therefore, the gate with the buffer forces the reverse packets to

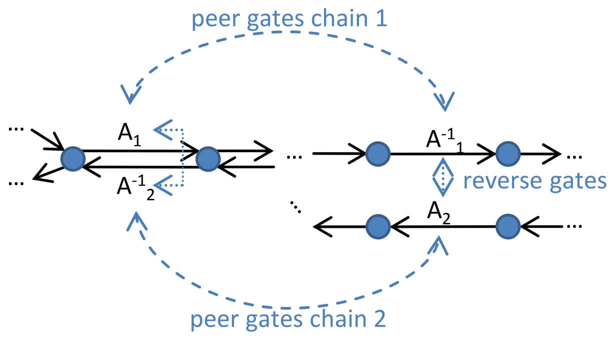

Figure 3.6.: Example with two chains and dependencies within each chain (peer gates; dashed arrows) and dependencies between the chains (reverse gates; dotted arrows) gate. Gates are depicted as continuous arrows and forwarding nodes as dots.

travel through a reverse gate. Typically, such reverse gates have internal "connections" to the gate that depends on them. Such a "connection" is in most cases just a local reference to each other in a shared memory. It enables the reverse gate to relay feedback from the peer gate to the dependent gate.

- Dependency to a function: If a gate does not require a specific gate but an arbitrary instance of a function, it depends on a function. For example, a gate doing video decoding does not necessarily depend on a specific peer gate but on a gate representing a function performing a compatible encoding. In addition to the dependency to a function, a gate can depend on specific parameter values of gates of this function.

Peer and reverse gates can be used to model the dependencies between two protocol state machine instances such as two dependent transport protocol instances (e.g. two TCP instances). Figure 3.6 shows a small example with two gates called A_i and A^{-1}_i per chain *i*. Gate A_1 buffers packets, adds the transport header, updates its states, and sends the packet along to its peer gate A^{-1}_1. Gate A^{-1}_1 removes the header, updates its states as well, and sends the packet to the next functional block. Moreover, it sends a feedback back to the reverse gate of A_1, which is gate A^{-1}_2. Gate A^{-1}_2 informs A_1 about the feedback. The access to the reverse chain is implementation specific. It can, for example, be implemented via information within the block itself or via its own reverse gate (A_2 in the example). A more complex example for chains and dependencies is given in Section 3.2.3.

Forwarding nodes do not have dependencies.

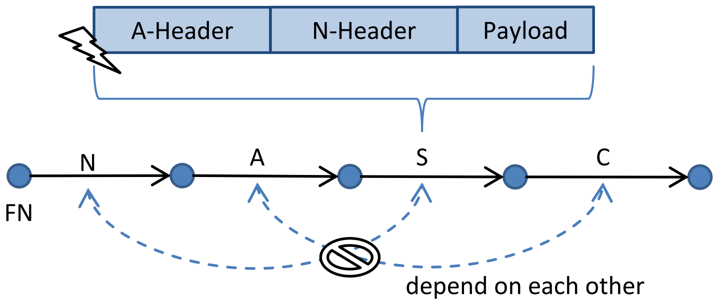

Figure 3.7.: Chain with wrong order of gates (arrows) due to dependencies (dashed lines). Gate numbers at the FNs (dots) are omitted. Packets arriving at A^{-1} have a wrong header at the top of their stack.

3.2.2.2. Order of functional blocks

A dependency might restrict other dependencies by forbidding "interleaving" dependencies. The dependency between two gates of a chain can forbid the blocks in between the two dependent gates to have dependent blocks before or after the two dependent blocks. Such a restriction is useful for blocks adding or removing headers to the payload of a packet. Since the model assumes a stack of headers, the order of adding and removing has to be ensured.

Figure 3.7 shows an example for a wrong gate combination. Gate A adds a header with a packet number to the payload in order to number the packets. This gate depends on gate A^{-1} sorting the packets at the receiver side. The sorting gate expects such a header and removes it. If the chain should also check for bit errors, gate B adds a header with a checksum. Gate B^{-1} removes the header and checks the checksum. A chain "interleaving" the gates in the order A, B, A^{-1}, B^{-1} would result in an error in A^{-1}, since the header of B is the first one in the stack. No gate between A and A^{-1}, and B and B^{-1}, respectively, is allowed to depend on a gate "outside" of the dependency.

3.2.2.3. Reuse

If each connection is assigned to one or more dedicated chains, the number of functional blocks in the model depends on the number of connections, their requirements, and the graph of the network. In order to enable setups with less functional blocks, the model allows "overlapping" chains. Functional blocks of one chain can be reused for another chain. More precisely, two chains being actively used by connections can partly use the same functional blocks. This *reuse* is an optimization and transparent for the connections.

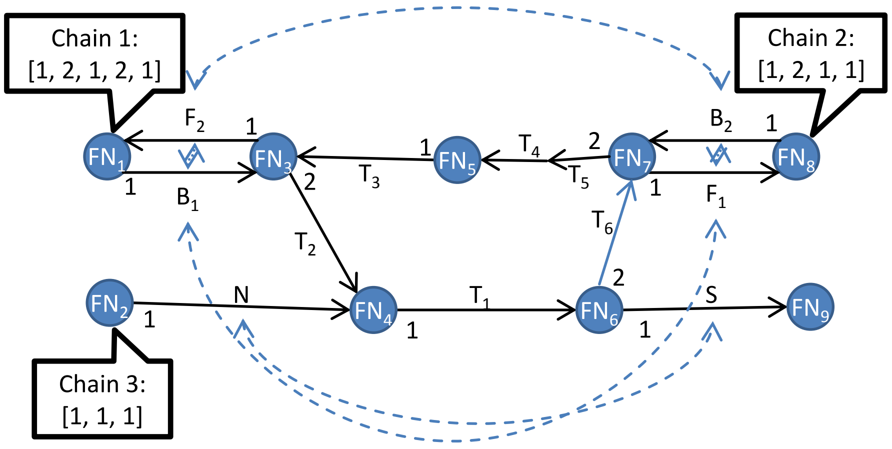

Figure 3.8.: Three chains with forwarding nodes FN_i (vertices), gates (edges), dependencies (dashed and dotted lines), and gate numbers. Starting points of chains are marked with the chain information.

The reuse is limited by dependencies, the reusability of functional blocks, and by the reuse algorithms. First, the dependency rules mentioned above must be fulfilled for all functional blocks in all chains. Second, the implementation of a function has to permit a reuse. If the states of a block depend on the data transmitted by a single connection, it cannot be reused by multiple connections. For example, a video transcoding function may depend on the last key frame of a video and, thus, cannot handle multiple videos at the same time. A forwarding node representing a specific connection end point is also not reusable as connection end point for multiple connections at the same time. Third, the algorithm deciding about the reuse has to balance its delay to find a suitable chain and its probability that a reusable block is actually reused. An example algorithm is outlined in the implementation chapter in Section 4.4.3.

3.2.3. Example

Chains of blocks can be depicted as graphs with forwarding nodes as vertices and gates as unidirectional edges. Figure 3.8 illustrates an example with three chains, by showing several gates concatenated for two connections. Dependencies are drawn as in Figure 3.6 as dashed and dotted arrows. Two gates directly concatenated are drawn as one long edge with two arrows (T_4 and T_5).

Chain 1 starts at forwarding node FN_1 and concatenates the gates B_1, T_2, T_1, T_6, and F_1. Chain 2 represents the reverse direction, which starts at FN_8 with the gates B_2, T_5, T_4, T_3, and F_2. The gates B_i may represent a calculation of a checksum and a buffer for sent packets. The gates F_i check the checksum and provide feedback to B_i. If there is no feedback or a negative one, B_i can repeat the transmission. The gates T_i represent functions for transporting packets between systems.

Chain 3 is used by a uni-directional connection and ensures the order of packets by numbering them in gate N and sorting them in gate S. It starts at FN_2 and concatenates the gates N, T_1, and S.

Each chain is defined by a stack of gate numbers required by the forwarding nodes. For example, FN_3 will remove the topmost gate number in a packet and use it to decide if the packet is relayed to gate T_2 with number 2 or gate F_2 with number 1.

The forwarding nodes FN_3, FN_4, FN_6, and FN_7 are reused for two chains each. In addition, gate T_1 is reused by chain 1 and chain 3. It represents the transportation of packet between relay systems. The gate hides the details of the transportation. Depending on the technique used for data transportation, the implementation might differ. For Ethernet, T_1 may send a packet to a MAC address specified as gate parameter. For an IP network, T_1 may represent a TCP connection between two relay systems.

Chain 1 and 2 show a typical setup for a bidirectional connection requiring functions with feedback mechanisms. Gate B_1 depends on the feedback of F_1, which is send indirectly via F_2. The "connection" between B_1 and F_2 is implementation specific. In most cases, both gates reside in the same process and share memory, which enables a fast communication between both.

3.2.4. Related work

Standard communication models include nodes and links connecting them. In general, these models focus on the network and routing in particular (e.g. Pathlet). The FoG communication model includes the layer specific parts of the network stacks from end and relay systems as well. Its functional blocks represent the traditional elements, such as links between nodes, in addition to stack functions. Therefore, the new model provides a handle for the dynamic placement of functions and for the location-routing problem attached to it.

The main difference to other models using functional blocks is the definition of chains and how they are implemented. In models such as used by SelNet [TG01], ANA [KHM+08], and SONATE [KSRM12], arbitrary functional blocks contain multiplexing functionality resulting in more than one output of a block. The multiplexing decisions of such blocks can depend on any value in a packet (e.g. Ethertype and Protocol field value) or even on the state of blocks. Thus, the entity defining a chain has to know the internal details of each functional block doing multiplexing in order to define a suitable chain. This is shown in the left part of Figure 3.9. The right part shows FoG's split between "real" functionality and its common multiplexing mechanism.

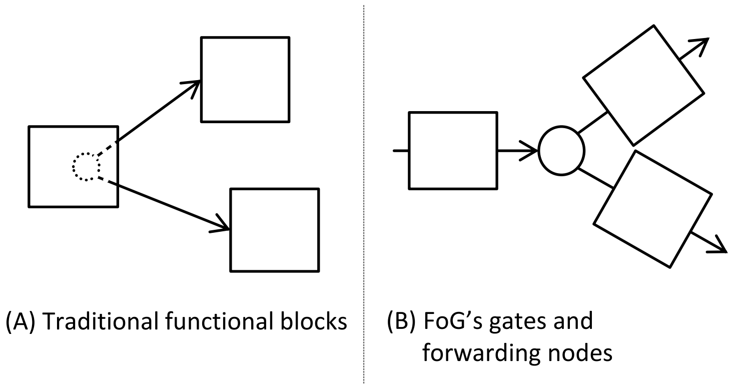
(A) Traditional functional blocks
(B) FoG's gates and forwarding nodes

Figure 3.9.: Differences in models using functional blocks. Functional blocks and gates are depicted as box. The forwarding node is depicted as circle. Arrows indicate potential subsequent functional blocks.

The common multiplexing mechanism introduces a standard way of defining chains, which, in turn, enables the construction of chains without knowing the internal multiplexing algorithm of each function. This advantage is achieved by restricting the possible interconnections of functional blocks in a chain to a sequence. Conditional branches and parallel structures known from other approaches are not possible. For example, a chain cannot define a sequence of blocks for the case that the load in the network is high and another sequence in case the load is low. The chain cannot define to use an encryption gate if the packet is not encrypted and to use another gates if it is. Although the model does not allow such setups, they are implementable with workarounds. For example, the decision entity can change the mapping from connections to chains in order to react on changing load conditions. A gate can internally switch between modes of operations depending on a packet. Thus, the decision if a packet has to be encrypted can be done inside of a gate.

The model is close to the operations in today's stacks. Each layer in today's stack combines the functions of at least one forwarding node and one gate. For example, IP is performing its operations such as TTL counter modifications and adding trace route information, which could be modeled as gate. Furthermore, it decides about the next functional block based on its Protocol field. If this field has the value 6, the packet is forwarded to TCP.[6] This is equivalent to the operation of a forwarding node.

The dependency model assumes that functions operate with a stack of header information. The impact of using heap structures for header information as proposed by the Role-based Architecture [BFH03] has not been analyzed. The

[6] http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml

"roles" of this architecture are comparable to functional blocks. A common aspect of both models is the "reflective role" (e.g. encrypt, decrypt), which is similar to a peer gate. However, RoleIDs (and, thus, role addresses) are not comparable to gate numbers because a RoleID contains an identifier for the type of function.

The model does not specify the granularity of the functional blocks. Even though the variety of functions implemented by forwarding nodes are strongly limited, the functions implemented by gates are not. Thus, the granularity of gates ranges from simple counter modifications via lower layer interactions to a complex video transcoding. However, the state of the art shows the impact of the granularity on the runtime complexity of the decision algorithm that sets up chains. The implementation of FoG presented in Chapter 4 uses rather large blocks such as EFCP blocks, encryption/decryption blocks, and transcoding blocks.

The model shows certain similarities to models used by route-based relaying PCI formats discussed in Section 2.3.1. If only functional blocks representing packet transmission between nodes are considered, a chain is equivalent to an index-based route. For example, a forwarding node is equivalent to a vnode in Pathlet, a gate can emulate a pathlet, and a gate number is equivalent to a pathlet forwarding identifier. As routes in Pathlet and all other approaches with index-based routes, chains are constructed by concatenating numbers. Such chains are also comparable to label stacks in MPLS. The non-functional aspects can be integrated in MPLS by using a QoS-aware routing and resource management, as provided by RSVP-TE. Some equivalent extensions are imaginable for Pathlet as well. However, the FoG communication model goes beyond the features of both by integrating arbitrary functions and not just relaying functions. This integration requires the consideration of dependencies in more detail. Traditional index-based routes ensure only dependencies that are caused by the graph structure of the functions. There should be no "gaps" in a route and relaying functions have to be executed in the right order. The new model extends this by taking dependencies between gates and functions into account.

3.3. Layer architecture

The communication model defines functional blocks, their interconnections, and how they exchange packets. The distributed nature of networks leads to a graph of functional blocks distributed over multiple FoG nodes. Implementations have to manage such a distributed graph, find and create suitable chains for connections in it, and relay packets through these chains. The *FoG layer architecture* is the blueprint for all implementations of this communication model and, thus, summarizes the invariances of all implementations. Its design ideas for accomplishing flexibility and scalability for arbitrary functions are

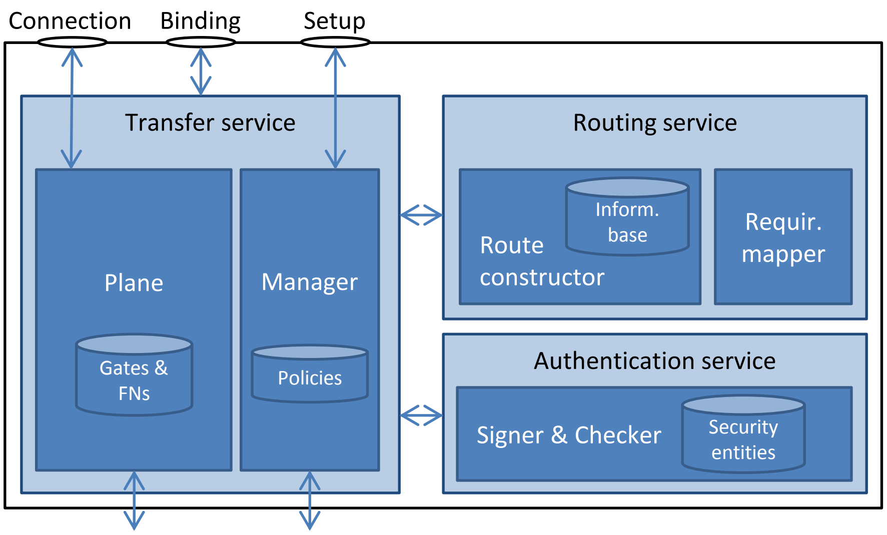

Figure 3.10.: FoG layer architecture with its logical components drawn as rectangles and interfaces as ellipses. Arrows indicate important data and control flows.

described in Section 3.1.3.

The three logical components transfer service, routing service, and authentication service of the FoG layer architecture are depicted in Figure 3.10.

- The transfer service is the "runtime environment" for the functional blocks. It contains and manages them and is responsible for relaying packets between them. Its details are presented in Section 3.3.2.
- The routing service determines chains through graphs defined by functional blocks. It is the "external" logic mentioned in the model description, which decides about how to combine functional blocks to chains. It is described in Section 3.3.3.
- The authentication service secures the management operations and establishes a base for authorization and accounting. It provides the possibility to sign packets and to check such signatures in order to verify the authenticity of packets. Relay systems can sign packets additionally in order to create "chains of trust", which enable a system to rank the trustworthiness of a packet based on multiple signatures. Section 3.3.4 outlines the authentication service in detail.

The separation of the packet relaying in the transfer service from the route computation in the routing service is an important feature of FoG. First, it enables the independent development of routing and transfer service. The relaying PCI format used by the transfer service is in particular no longer limited to a specific address format used by the routing service. Second, the separation leads to a concentration of connectivity, capability, and QoS information in the routing service. This enables routing services to take full advantage of the holistic information base in order to optimize system performance.

Transfer and routing service have to cooperate in order to create a chain in a distributed graph of functional blocks. The computation of a complete chain at a single end system seems to be the obvious solution along the lines of the model. However, this strategy causes scalability problems and may not even be possible at all for large inter-networks because this node would require detailed information from all nodes of a network. The FoG layer architecture introduces an incremental computation strategy. This so-called *incremental routing process* uses partial chains. A partial chain is only a part of a complete end-to-end chain of the communication model. Multiple partial chains are subsequently concatenated by the incremental routing process in order to constructs a complete chain. The process addresses the scalability by enabling chain computations based on partial knowledge. A *partial chain* reflects partial knowledge of the graph of functional blocks. Not a single entity with global knowledge has to construct the complete chain but multiple entities having partial knowledge. The process is flexible regarding the number of entities involved and regarding their contribution. A contribution depends on the requirements of a chain and can differ due to policies and capabilities of each entity. The process respects the autonomy of subnetworks by integrating different operators in the computation of a chain. Thus, an operator can contribute a partial chain during the incremental routing process in order to enforce its policies. More details about the process are given in Section 3.3.5.

The FoG layer architecture represents (partial) chains as routes. A *route* is a stack of route *segments*. There are two types of segments:

- An *explicit segment* defines a (partial) chain. As defined in the model, it contains a stack of gate numbers.
- A *destination segment* indicates a gap in the chain and enables the incremental routing process. It contains information required to resolve subsequent partial chains, which can fill the gap. The information comprises a destination name and the requirements for the chain to this destination.

Usually, the incremental routing process transforms destination segments subsequently to explicit segments. When and where the transformation takes place depends on the configuration of the network. This increases the flexibility compared to IP, which defines the location of the transformation statically by the deployment of network components.

Moreover, this process influences the location of functional blocks. Depending on the policies of the incremental routing process, functional blocks can be placed either on end or on relay systems. The communication model enables this flexibility by representing the transmission of packets between FoG nodes and all other functions as functional blocks.

The FoG layer is accessed via an API hiding all its details from higher layers. It provides the possibility for applications to specify their requirements for connections explicitly. Its details are outlined in Section 3.3.1. Since FoG is designed for a recursive layer stack, it uses the very same interface for accessing lower layers. The interaction with these layers is described in Section 3.3.7.

Section 3.3.8 compares the FoG layer architecture with related research proposals. A review of FoG in the context of the evaluation questions from Section 3.1.1 can be found in Section 3.7.1. A comparison between FoG and its recursive reference model is shown in Section 3.7.2.

3.3.1. Interface

Applications and higher layers access FoG's service via an interface that focuses on requirements, names, and events. The requirements influence the way FoG is providing connections between higher layer entities. In order to enable a flexible reaction on requirements, the networking details are hidden from higher layers. For example, addressing details are hidden behind names chosen by the higher layers themselves. Furthermore, only a single set of functions for stream, datagram-stream[7] and datagram communication is offered. Higher layers are informed about asynchronous happenings within the layer, like failures, via events. While the interface is designed especially for recursive layers as described in [LHSS13], it adopts the naming and requirement aspects of the GAPI.

The functions of the interface can be clustered in three logical subinterfaces, which are shown as classes in the UML class diagram in Figure 3.11. The Layer interface provides functions for registering for incoming connections and for setting up connections to such registrations. The Binding and the Connection interfaces provide control over a registration and a connection, respectively. All subinterfaces shares the possibility to query for events. Therefore, they share a common base interface called EventSource, which provides access to events. The most important functions are outlined in the following. The complete interface is described in Appendix B.

The Layer interface provides the function `bind` for higher layer instances to register themselves. They bind themselves to a name chosen by them in

[7] Analog to TP4: Stream, which preserves the delimiting of datagrams defined by higher layers

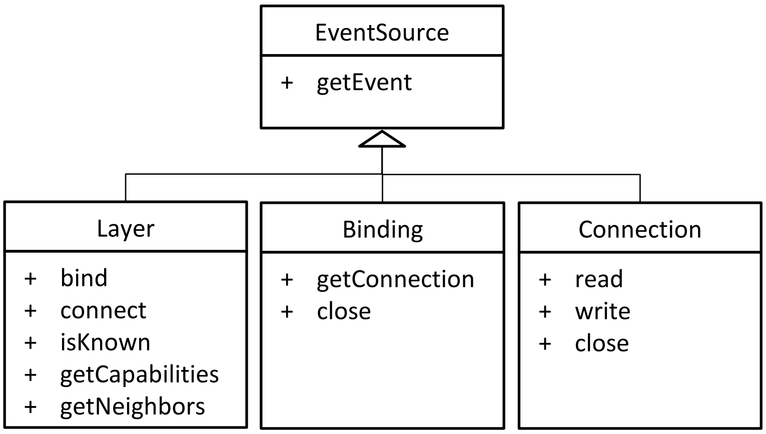

Figure 3.11.: Simplified layer interface of FoG as UML class diagram (arrow points to base class)

order to announce their accessibility for others. In addition to the name, the higher layer can specify requirements that should be fulfilled for all incoming connections established later on. The result of the bind function is a reference to an object supporting the Binding interface. The function `connect` establishes a connection with previously created Bindings. It requires the name of the Binding that should be contacted and the requirements for the connection. Both sets of requirements – the one given by the `connect` function and by the `bind` function – have to be fulfilled in order to establish the connection. If they cannot be satisfied, the `connect` function will return an error. If they can be satisfied, two objects supporting the Connection interface are created. One is returned to the caller of `connect` and the other is given to the caller of `bind` via the Binding interface. Both can now exchange data via the Connection interface. Failures causing a violation of the requirements during the communication are signaled by the end-points via events. In order to reduce the probablity for such errors, a higher layer can get an approximation or guided guess of what may be possible via the `getCapability` function. This function proved useful for a layer to estimate the opportunities provided by lower layers. Since higher layers may register with identical names, the function `getNeighbors` is useful to get a handle for multiple neighbors with the same name. The function does not necessarily return all higher layer registrations matching some filter. Instead, it may return only a subset known by a FoG node. Analog to the information returned by the `getCapability` function, this information helps a layer entity to get an overview about its situation and possible neighbors.

Higher layers operate only with names of their own name spaces. They are neither aware of any addresses nor of functional blocks. In general, FoG allows multiple registrations with the same name. In case of `connect` requests for such names, FoG treads them as any-cast names and selects one of them as destination for the request. However, an implementation can reject such registrations due to policy reasons. A FoG instance and a higher layer share a label for a connection and a binding. These labels are only valid locally for a single FoG instance. They are chosen by the FoG instance freely, like the socket handles for Berkeley sockets.

Requirements are specified according to the proposal in [SM12], which is also used by the GAPI. Each *requirement* is a triple with an effect, an operator, and an attribute. The *effect* describes something observable that is limited by the *attribute*. The type of limit is specified by the *operator*. Examples are:

- delay <= 120 ms: Limits the delay to a maximum of 120 ms.
- in-order = true: Requests the delivery of data in the order it has been sent.
- video quality = 90%: Sets the quality parameter for video transmissions.

FoG adds the possibility to specify the direction of a connection the requirement should apply to.

The interface provides a single set of functions for a connection-oriented service. Although the service is connection-oriented, the implementation does not necessarily need to be connection-oriented as well. The implementation depends on the requirements for a connection and supports stream-, datagram stream-, and datagram-oriented communication. If a connection requires only datagrams, the implementation can use a connectionless protocol. This design is enabled by the recursive model, which hides the choice of the EFCP (comparable to transport or logical link control protocol in OSI) from higher layers.

3.3.2. Transfer service

The *transfer service* is responsible for relaying packets between functional blocks according to the FoG communication model. Furthermore, the transfer service creates, configures, links and deletes the blocks. It comprises the transfer service plane and the transfer service manager. The former implements the data transfer of the recursive layer model with its tightly coupled mechanisms. The latter contains the loosely coupled mechanisms of the data transfer controller.

3.3.2.1. Logical view

The transfer service plane contains the functional blocks of the FoG communication model. It provides the execution environment for the functional blocks and relays the packets between them. As long as a (partial) chain is defined in a packet, the transfer service plane relays the packet between functional blocks as defined by the chain. If no chain is given, the transfer service plane requests it from the manager. The manager decides if the packet is allowed to be forwarded. If the decision is positive, the manager contacts the routing service in order to request a route for the packet.

The transfer service informs the routing service about its functional blocks, their interconnections and supported functions. The routing service calculates routes based on this information. It tries to reuses already available (reusable) functional blocks for multiple routes. If the routing and transfer service are operated by different operators and the transfer service does not fully trust the routing service, the transfer service can *hide* (meaning: do not report) functional blocks or gate numbers from the routing service. When the routing service requests new gates, the transfer service can decide about the reuse on its own. Moreover, the transfer service can hide non-reusable functional blocks in order to reduce the interactions with the routing service. Examples are blocks dedicated to a single chain, such as forwarding nodes representing connection end point, and blocks that cannot be reused.

Bindings are represented by forwarding nodes, which are labeled with the name of the binding. The transfer service reports the higher layer name to the routing service in addition to the forwarding node. Thus, the routing service knows the forwarding node that is responsible for the access to a higher layer entity.

The transfer service plane delegates the decisions related to the setup of blocks and chains to the transfer service manager. The manager is responsible for the management decisions regarding the creation, linking, and deletion of functional blocks. Moreover, it assigns a connection to one or more[8] chains. The decision, which blocks or functions are required for a chain, is taken by the routing service.

The creation of new functional blocks can be requested from the transfer service plane in multiple ways. First, they can be requested manually by an operator. Second, a policy can cause the creation of gates in certain situations. For example, a FoG node can automatically set up best-effort gates to all detected neighbors. Third, signaling protocols can be used to trigger the creation (or deletion) of gates from remote. In order to authenticate such signaling messages, the authentication service is used.

The transfer service manager is influenced by policies defined by network operators. Such a policy defines the guidelines for the decisions of a manager. For example, it specifies which entity is allowed to set up which gates. Furthermore, it can limit the amount of reserved resources by functional blocks or the set of functions available on a FoG node.

[8] Typically two chains for a bidirectional connection

3.3.2.2. Distributed view

The overall transfer service of a FoG network is distributed over multiple transfer service entities. Each transfer service entity provides only a part of the overall graph of functional blocks and stores only the states required for its blocks. A transfer service entity resides in a *FoG entity*, which, in turn, is located on a FoG end or relay system. The term *FoG node* refers to a FoG system and assumes that there is only a single FoG entity on it. In most cases the terms FoG node and FoG entity are used synonymously.

Without loss of generality, each transfer service entity has an associated routing service entity. If no routing should be done on a node, the routing service entity acts as a proxy for another remote routing service entity. Multiple routing service entities can be hidden behind a single entity, which forwards the announcements of the transfer service entity to all other routing service entities.

As with today's IP, the transfer service entities communicate via lower layers (e.g. an Ethernet-based recursive layer; cp. Section 3.3.7), which enable a packet exchange between them. The function for sending FoG packets to other transfer service entities is encapsulated in gates. However, a new relaying PCI format that supports the forwarding model of FoG is required for the exchange. This format is described in Section 4.3.2.

3.3.3. Routing service

The *routing service* calculates routes, which are used by the transfer service to relay packets. The calculation is influenced by various requirements of the network as well as of the connection. Therefore, the routing service has to derive the set of required functions from the requirements via its requirements mapper component. Missing functions are requested from the transfer service. The route calculation requires an information base that contains the information reported from the transfer service. In general, it contains the graph of functional blocks and the capabilities of the transfer service entities. According to the recursive layer model, the routing service is independent of connections and, thus, belongs to the layer management.

3.3.3.1. Logical view

The transfer service informs the routing service about functional blocks and their interconnections. Furthermore, it reports its capabilities in terms of available functions and non-functional parameters. Thus, the routing service does not require any mechanism for discovery. It combines the information from the transfer service to a graph of functional blocks available for reuse by chains. The functional blocks not available for reuse are omitted. Such blocks cannot be integrated in chains but would enlarge the graph the routing service has to manage.

Based on a source forwarding node and a destination name, the routing service calculates a route through the graph of gates and forwarding nodes with respect to various requirements. The destination name identifies the destination forwarding node representing the binding with the same name. The requirements introduce two different kind of constraints. First, the non-functional requirements act as filter for the properties of existing blocks. A block can only be included if it supports the non-functional requirements. Second, the functional requirements force the routing service to include specific functions in a chain. The *requirements mapper* component of the routing service translates requirements to a list of required functions. Based on such a list, the routing service constructs a chain, which includes functional blocks implementing the listed functions. The construction has to take the dependencies between blocks into account. It typically tries to reuse existing blocks in order to minimize the amount of blocks and, thus, to reduce the states the whole network has to handle. If the graph does not provide suitable functional blocks to satisfy the requirements, the routing service determines missing functions and requests them from the transfer service.

The requirements for a route calculation are defined by several sources. First, the higher layer specifies its requirements for a connection via the `connect` function of the Layer interface. Second, the binding of the higher layer that the connection should connect to specifies its requirements for incoming connections as well. Third, a routing service policy defined by the operator can introduce requirements. The routing service has to combine and to satisfy all of these requirements. If this is not possible, an error is returned. An example is given in the following: A higher layer requests an encrypted connection with a throughput of at least 10 Mbit/s. The higher layer binding requires all connections to deliver data in order. The policy enforces a logging function for accounting purposes of non-best-effort connections. In total, the routing service has to construct a chain with functional blocks for encryption, decryption, numbering, ordering, and logging. In addition, some relaying functions are required to reach the remote binding. All these blocks have to support at least the desired throughput. If there are no suitable blocks for reuse in the graph, the routing service has to request new ones from the transfer service.

Typically, the routing service does not operate with the higher layer names directly. In order to enable a more efficient specification of a location, it introduces addresses. The mapping between the higher layer names and the addresses is stored in the routing service information base. The format of such addresses depends on the routing service and is not limited by the transfer service. The transfer service provides variable length fields for the cases it has to transport addresses in signaling messages. Depending on the scenario, the routing service can shorten destination segments and, thus, routes by introducing addresses. This holds in particular for scenarios with long higher layer names and short addresses.

3.3.3.2. Distributed view

The overall routing service of a FoG network is composed by multiple routing service entities. Routing service entities exchange their knowledge about transfer service entities with a routing protocol. The exchange establishes a broader overview over the graph of the transfer service plane and enables the calculation of chains to remote bindings and the reuse of remote functional blocks.

A routing service has to balance two contrary aspects of the information exchange. On the one hand, it requires the detailed information about the transfer service in order to integrate existing functional blocks in chains. On the other hand, this information cannot be distributed among all routing service entities due to scalability. In order to reduce the amount of information, the architecture allows routing service entities to aggregate their knowledge or hide parts of it from others. One consequence is that one routing service entity may not know all gates on the way to a binding and, thus, is not able to calculate a complete chain for a connection. Therefore, routes returned from a routing service entity are allowed to be partial routes. A *partial route* defines only a part of the whole chain to the destination. Without loss of generality, such a route ends at the border of the known area of a routing service entity. At this border, another routing service entity knowing the parts beyond this border has to be contacted. This "neighbor" routing service entity calculates the next partial route towards the destination. The so-called incremental routing process, which is described in Section 3.3.5, concatenates the partial routes incrementally to a complete chain.

The incremental routing process assumes that several routing service entities – each knowing only a part of the transfer service – have to cooperate in order to create a complete chain. Thus, a routing service entity requires detailed knowledge of only a part of the transfer service. In common scenarios, the detailed graph is a connected graph of functional blocks representing "the scope" of a routing service entity and its surrounding. However, the incremental routing process assumes implicitly that a routing service entity knows the direction towards a destination. More specific, it has to know a forwarding node at the border of its detailed knowledge that is closer to the destination than the source forwarding node. Consequently, the detailed knowledge about a part of the transfer service is not sufficient. It has to be expanded with more abstract information about the remaining parts of the transfer service. The representation of the abstract direction information is implementation dependent. In most cases, routing services introduce addresses, which enable an efficient specification and aggregation of locations.

The following steps are required to calculate a partial route at the routing service entity RS_a with its abstract information base I_a and its detailed information base I_d. Forwarding node S is the starting point of the chain and forwarding node D is the destination.

1. RS_a checks if S is in I_d. This should be the case for normal configurations, because the route request comes from a transfer service entity reporting to S. However, if S is not known, no route can be calculated.

2. If D is not known by I_d, RS_a determines a forwarding node Z via I_a. By definition, Z is closer to D and known in I_d. If no Z for D is known, the route calculation failed. If D is known by I_d, Z is equal to D.

3. RS_a calculates route R from S to Z. If Z is different from D, a partial route segment with the information about D and the remaining requirements is appended to R.

4. RS_a returns R.

Another important cause for not distributing information about functional blocks to others are reuse considerations. A routing service entity can decide to distribute the knowledge about a gate to a set of other routing service entities. Each of these entities can reuse the gate for routes. If the gate number is distributed, they can decide about the reuse on their own without asking the providers of the gate. If the gate number is hidden, they can at least plan to reuse it and delegate the final decision to the routing service entity knowing about the number. A routing service entity can monopolize the reuse decision by hiding gate numbers. It can delegate the decision to other entities by announcing them.

The logic of the hiding decision is similar to the logic the transfer service uses to decide whether it should report a functional block to the routing service.

3.3.4. Authentication service

The management of FoG involves many automatic decisions such as the creation of gates for connections. Most critical are decisions about gates representing resource allocations since they reduce the remaining system capacity for others and might be subject to accounting. Therefore, a security mechanism is required to protect a FoG system from unauthorized resource usage. The *authentication service* is the base for such a mechanism. It enables a sender to sign messages and a receiver to check if received messages are authentic. Based on the authentication, a receiver can perform authorization and accounting.

The main user of the authentication service is the transfer manager. It uses the authentication service to validate signaling messages exchanged between transfer manager entities. In particular, requests to open gates with special QoS requirements are signed in order to enable the receiving manager entity to validate the identity of the sender.

3.3.4.1. Logical view

The authentication service manages security entities and their credentials. For public-private-key systems, the authentication service stores the keys of the entities. The entity names are not necessarily related to the names used in the routing service or used by higher layers. The authentication service can use a different name space for the security entities. A separate name space is especially useful for scenarios, in which names change but security entities should stay constant. For example, the address of a FoG node may change in a mobile environment. However, a constant security entity simplifies the authentication of signaling messages involved in handovers.

The authentication service provides the possibility to create signatures for messages in the name of an entity. It uses the credentials of the entity in combination with a checksum of the message to create a signature. Moreover, it can check if a signature is valid for an entity and a given message.

3.3.4.2. Distributed view

A single authentication service entity typically does not know all security entities of a large inter-network. The following cases occur:

- For its own "local" security entities, it can generate and check signatures. In a public-private-key system, an authentication service entity knows both keys for the security entities of its scope (e.g. for a subnetwork).
- Some remote security entities are known and their signatures can be checked. In the example of a public-private key system, the authentication service entity has exchanged public keys with other authentication service entities or public keys have been installed manually.
- Some security entities are not known at all. A public-private key system could try to resolve the public key of such entities in a reactive way.

In general, a request for resources is signed by the higher layer entity requesting a connection. Its security entity typically has a rather limited scope and is known only within its subnetwork. If a destination end system outside of the subnetwork communicates regularly with the source, the entity might be known to the destination end system as well. However, the entity will most probably not be known to most relay systems. Therefore, a signed packet can be signed again by a relay system in order to create a *"chain of* *trust*". For example, a gateway system can add signatures in the name of the subnetwork to outgoing packets. The gateway can most probably check the signatures of outgoing packets, since they are signed by the users of its own subnetwork. If they are valid, it adds a signature from a more general entity, e.g., a security entity representing the subnetwork. The next relay system has to know only the security entity of its neighbor subnetwork. If it can validate the subnetwork signature and if it trusts its neighbor, it may also trust the users of the subnetwork. If it trusts the chain of signature, it can add its own signature before relaying the packet to the next relay system. This process continues along the route of a packet. The destination end system receives a packet with a list of signatures, which can be used to decide about the trustworthiness of the packet content.

The "chain of trust" is used to reduce the number of security entities an authentication service entity of a relay system has to know. It trades the overhead in a packet for the memory required by an authentication service entity and vice versa. In order to support the "chain of trust", a relay system has to add a signature if it knows that the next neighbor does not known the last signing security entity. Consequently, a relay network can omit to sign packets, if all its neighbor subnetworks are known to each other. This is especially interesting for high-speed relay networks ("backbone"), which are not capable to sign all packets due to the low relay times for individual packets.

Are you again balancing memory and packet overhead?

Yes, the "chain of trust" is comparable to a chain of gate numbers in this context. It balances data rate required to transport overhead information in packets and memory required to store states on relay systems.

Since subnetworks are autonomous, they can decide not to support a "chain of trust". Thus, an end or relay system can receive interrupted signature sequences with the last signer unknown to the system. In such cases, the authentication service entity can try to gather information about the last signer from other authentication service entities. If such an information exchange is possible or allowed, depends on the implementation of the authentication service and its policy. If the authentication service can not verify the sequence of signatures, it reposts an error to the caller that requested the check. Depending on the content of the packet (e.g. a signaling message) and the required level of trust to proceed, the caller can decide about its further actions.

3.3.5. Incremental routing process

The interworking between the transfer service and the routing service is the most important aspect of FoG. The *incremental routing process* for determining chains is the key interaction between both and distinguishes it from other layer architectures. In contrast to the creation of individual gates via dedicated signaling messages, it provides a mechanism to create multiple gates along a path. While the former is mainly used in a proactive way, the incremental routing progress is mainly used (but not limited to) to create gates for connections in a reactive way. The process is executed while sending a packet towards its destination. This packet can contain a connection setup request of FoG's access protocol or the first higher layer data. The latter is especially useful for implementing the exchange of short request/response datagrams. The key point of the process is its flexibility when to switch between both services. The switching defines the placement of states in a network. As long as transfer service entities have sufficient information to relay a packet, their associated routing service entities are not relevant. At the switching point, the states in the transfer service are not sufficient any longer and the information of the routing service entity at this point becomes important. The changeovers between both services have performance and policy implications.

The changeovers influence the performance, because both services come along with different cost. The routing service complexity is higher than the complexity of the transfer service, since it decides how the transfer service has to relay data. Its higher complexity leads to algorithms that are more costly in terms of latency and resource usage. Changeovers allow balancing complex decisions in the routing service with efficient and simple relaying in the transfer service[9]. From a pure performance perspective, an average number of routing service requests per relaying action near zero is the best case. It indicates that a small number of routing service responses can be used for a large number of relay actions. However, a high number of routing service requests does not necessarily cause a slower network. As in IP networks, the hardware can be optimized to handle such requests for most cases in line speed. Nevertheless, the network equipment could be cheaper, if a different trade-off is selected.

[9] The performance implication relates to the process of finding the next gate for a packet. The complexity of the function of a gate and, thus, its performance requirements may be much higher. However, the complexity of functions is not an issue of the FoG layer architecture.

But prices in the real world...

Prices for hardware in the real world are not only a function of the number of transistors. The name of the vendor and the number of produced pieces are also important – or even more important – aspects. However, the hardware itself would require fewer transistors and would be smaller. If all other factors stay constant, this would enable the vendor to sell it to a cheaper price. Moreover, the operational costs would be lower due to less energy consumption.

The switch between transfer and routing service does not only have performance implications. Its state distribution aspect is more related to policies. The more gates a routing service entity knows, the higher the probability that is can reuse one of these gates. More reuse leads to "longer" explicit routes, which, in turn, reduce the number of route requests by the transfer service. Whether or not a routing service entity knows about gates depends on the policies of the transfer service and of the routing service. Both have to allow the routing service entity to reuse it. Performance considerations are one influence factor of such decisions. Especially, the announcement of gates representing best-effort connections between two FoG entities can reduce the routing service load. For resource intense gates, however, the authorization aspects can dominate the decision. If a routing service entity requests the creation of an expensive gate requiring a lot of resources, it may not announce the gate to others in order to monopolize the usage decisions for this gate. Thus, the system may accept a high number of routing service requests in order to implement its policies.

3.3.5.1. Workflow

FoG forwards packets according to the route given in each packet. More specific, forwarding nodes chose the next gate for packets according to the topmost gate number of the packet's route. If there are no more gate numbers in a route, the forwarding node currently processing the packet has to take further actions. If the route is empty, the packet reached its destination. If not, the next route segment has to be a destination segment. It specifies the destination the packet should be forwarded to. The forwarding node informs the transfer manager about the packet without valid chain information. The transfer manager requests the next partial route from the routing service, if permitted by its policy. It uses the forwarding node as starting point and the information given in the destination segment of the route as destination. If the destination segment contains requirements, they are used as requirements for the route request. If no requirements are given, best-effort is assumed. The routing service calculates the route and returns it to the transfer manager. The transfer manager replaces the destination segment of the original route with the segments returned by the routing service. Afterwards, the manager injects the packet to the transfer plane by handing it over to the forwarding node again. The forwarding node will restart its forwarding process with the new topmost segment of the route. Details about the process are given in the context of the implementation in Section 4.3.2.3.

There have to be precautions that this process will terminate and not loop forever. Theoretically, the transfer manager has to check whether there is no loop in the sequence of destination information. In practice, a limitation of the number of subsequent requests for the same packet can be suitable as well.

The route the routing service returns to the transfer manager defines when the transfer service has to contact the routing service again. Thus, the switch-over from the transfer to the routing service can be adjusted flexibly by the routing service. The explicit part of the route (meaning the gate numbers) specifies a path to the border of the area known by the routing service entity. The transfer service delivers the packet to the border by relaying the packet along the path. The process assumes that another routing service entity at the border takes over and that this other routing service entity knows the next part of the route to the destination.

Precautions are required in case of lost and retransmitted packets that are subject to the incremental routing process. Without them, each retransmitted packet with a partial route may cause the creation of gates. If the gates are required for a connection, the incremental routing process can resort to the information transported by the access protocol. An example implementation using an "at-most-once" sematic is described in Section 4.3.3.

3.3.5.2. Example

Figure 3.12 shows an example with two FoG nodes with their routing service entities R_i and transfer service entities T_i. Gates and forwarding nodes are depicted as in Figure 3.8. Black dotted lines represent abstract connectivity information of a routing service entity exchanged with other routing service entities.

The process of finding a route starts in the transfer service entity T_1 at FN_1. In step 1, a packet with the destination address of FN_4 is handed over to FN_1. Without loss of generality, the example assumes that the transfer service manager has inserted the packet after getting a partial route from R_1 before. Since a destination segment is the topmost segment, FN_1 informs its manager. The manager resolves a route via the routing service entity R_1 associated with its transfer service entity T_1. R_1 returns a partial route in step 2, which is used by FN_1 to proceed. FN_1 looks up the next gate according to the topmost gate number and relays the packet to gate a. In this example, gate a transports the packet to another nodes transfer service entity F_2 by a lower layer in step 3. The manager of FN_2 of T_2 has to request the next route due to the topmost destination segment. R_2 calculates a route to the forwarding node FN_3 specified

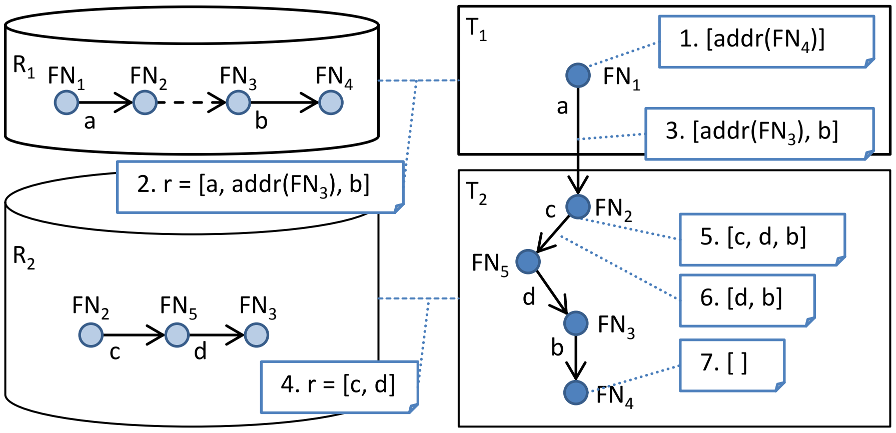

Figure 3.12.: Incremental routing example with two FoG nodes. Gates are depicted as arrows with gate numbers as small letters. The annotations show the route of a packet during the steps of the example.

in the destination segment in step 4. The manager of FN_2 adds the result to the remaining route of the packet and hands it over to FN_2 in step 5. FN_2 continues by using the topmost gate number and relays the packet to gate c. The process continues until the route is empty and the packet reached its destination in step 7.

The example assumes that the existing gates satisfy the requirements given in the destination segments. If this is not the case, R_i has to request new gates from T_i.

The destination segment is not necessarily the last segment in a route. As in step 2 of the example, it can be used to specify intermediate nodes of the route. Due to security considerations [Rei07], policies can limit the number of destination segments.

3.3.5.3. Abstract perspective

From an abstract point of view, gate numbers represent decisions of the routing service that are executed in the transfer service. Interpreting them as indices in a gate vector of a forwarding node is just a basic example. In general, gate numbers can be used to shift information from relay systems (e.g. gate parameters) to end systems or relay systems hosting previous parts of a chain without telling them about that shift.

For example, a FoG node has two alternatives to encode the address of the next hop for an Ethernet broadcast domain. First, the node can create one gate that stores a MAC address per neighbor. Second, it may create only one gate for the Ethernet interface and store the MAC addresses of the destination node in routes. The gate would use the MAC address given in the route in order to determine the next hop for a packet. In general, moving states enables stateless gateways, which do not have to store an address mapping (e.g. between IP and MAC addresses). The shift is transparent, since others just operate with lists of gate numbers and do not need to know the meaning of each number. In Section 3.6.2, interoperability solutions for IP exploit this shift to implement stateless gateways.

If a transfer service entity would like to make it hard for others to guess MAC addresses, the routing service can encrypt the address with a key known only by the transfer service. The transfer service can now check, whether a MAC address given in a route has really been provided by its routing service. The scheme used to encode the decision of the routing service can be adapted to the level of security, which is required by the network policy. Similarly, a node can increase the cost for an attack that guesses gate numbers by avoiding "simple" gate numbers such as 1 and 2 and by using random or encrypted gate numbers with more bits (as much as the policy requires).

Knowing the representation of decisions and how they are secured, introduces dependencies between the routing and transfer service. For example, the scheme itself and its parameters, like keys, must be known to both. In an inter-network scenario that requires both entities being operated by the same administrative domain. Fortunately, the incremental routing process ensures exactly this. By not announcing the gate c and d in the previous example, R_1 is forced to involve R_2 in the end-to-end route calculation. R_2 is therefore getting the chance to insert self-generated route segments into the route. How R_2 is encoding its decision for T_2, is not known by R_1.

3.3.6. Report and request functional blocks

The transfer service has to report its capabilities and can report reusable functional blocks to the routing service. These reports are not related to nor triggered by connections. A transfer service entity can publish its capabilities and blocks without waiting for a request. Requests for functional blocks are also independent of connections in order to allow a proactive setup.

The base for reports and requests are bijective names (labels in most cases) of the functional blocks of a transfer plane entity. They are assigned by the transfer service and used to describe the transfer service plane to the routing service. The routing service uses these names to describe where to add newly requested gates. These names are only exchanged at the interface between the routing service and the transfer service. In general, they are neither used between two transfer service entities nor between two routing service entities. Typically, the routing service assigns addresses to the forwarding nodes in order to hint at their location in the network graph. These addresses – not the functional block names – are exchanged between routing service entities.

The creation of individual functional blocks on remote systems can be implemented with a simple request/response signaling scheme. The creation of multiple dependent functional blocks in several systems can be coordinated with protocols comparable to RSVP. Moreover, the incremental routing process can trigger the creation of multiple functional blocks on relay systems. It efficiently combines the delivery of the first packet (e.g. some signaling message or the first (and maybe only) data packet) with the creation of blocks. The requirements that should be fulfilled by these blocks are stored in the destination segment of a route.

3.3.7. Interaction with lower layers

Due to the recursive layer model, the lower layers are accessed via the same interface FoG provides to higher layers. Analog to the encapsulation of the FoG details, the interface hides the implementation of the lower layers from FoG.

The transfer manger uses the Layer interface functions `bind` and `connect` to establish (N-1)-connections between transfer service entities. Since these functions require names as parameters, the transfer manager uses names of transfer service entities that have to be bijective in the scope of a lower layer. Comparable to the identifiers for functional blocks required for the interface between the transfer and the routing service, these names are only valid for the interface between the transfer service and the lower layers. The names are required to identify multiple FoG transfer service entities attached to the same lower layer.

Gates represent (N-1)-connections in the FoG world. They use the `read/write` functions provided by the Connection interface to send and receive FoG packets via these connections. Thus, fragmentation of FoG packets has to be done by the lower layer if required. The function `getCapabilities` is used to request information about the capabilities of the lower layer. The capability information can be reported by the transfer service to the routing service directly. Alternatively, gates can be enriched with this information. If such a gate is reported to the routing service, the routing service learns about the capabilities implicitly. In particular, gates representing best-effort (N-1)-connections are good candidates for the implicit announcement of non-functional "link" capabilities.

In order to attach to a lower layer, a FoG node has to perform the following steps:

1. The transfer manager is informed about a new lower layer by the operating system.

2. The transfer manager requests a list of neighbor transfer service entities reachable through the lower layer via the `getNeighbors` function. Such instances are identified by a common name prefix, like “fog://”.

3. The transfer manager sets up (N-1)-connections to all neighbors on the list via the `connect` function. In general, such connections will be bidirectional. The contacted node will perform step 8.

4. These connections are represented as best-effort gates, which are reported to the routing service.

5. In order to deal with dynamic changing networks, the transfer manager can register for events, such as joining or leaving nodes. As reaction to the events, the transfer manager can close old connections or create new once.

6. The transfer manager determines a name for its point of attachment to the lower layer. As mentioned above, the name has to be bijective in the scope of the lower layer. Such names can be constructed by concatenating the prefix mentioned before and a random suffix. If the generated name is already used by another entity (test with function `isKnown`), the transfer manager can retry to create another one. Since the length of these names is not limited[10], the namespace is sufficiently large that this algorithm terminates.

7. The transfer manager calls `bind` with the name generated in step 6. The binding announces the presence of the FoG entity in the lower layer.

8. Nodes that have not been contacted in step 3 may now contact the entity by themselves. The transfer manager is informed about incoming connections via events for its binding. It can accept such incoming connection requests via the function `getConnection` and integrate these connection analog to step 4.

Other orders of these steps are possible. In particular, steps 6 to 8 can be performed at the start or parallel to the others. Systems acting purely as end system can omit the steps 6 to 8, if the previous steps have been successful. Different policies (e.g. connections/gates only to a specific subset of neighbors) or manual system configuration are possible as well.

[10] In practice it is of course limited. However, the memory limitations are typically not so severe that they are relevant for names.

3.3.8. Related work

An earlier version of the FoG architecture that was not designed for a recursive reference model and that was focusing on gates representing lower layer connections only is described in [LVMT12].

FoG shares its separation of transfer and routing service with the PFRI [CGP07], which uses labeled "channels" and anonymous nodes to model the available connectivity of lower layers. The channel labels are globally bijective. Routes are loose source routes that contain the labels of the channels a packet should traverse. Channels are comparable to FoG gates representing lower layer connections. However, the PFRI channel labels are not comparable to gate numbers, since gate numbers in FoG are only bijective in the scope of a forwarding node. FoG prefers to name forwarding node and not gates. The assumption is that a network graph has more edges than vertices, and thus, less addresses are required if nodes are addressed. The typical Internet-like graphs used in the performance studies support this assumption (cp. Section 5.1).

DRUID [TBD+11] is a proposal for a combined research project of eight universities in the US. Since Joe Touch's and Abraham Matta are part of the project, their recursive approaches RNA [TP08] and RINA [Day08a, DMM08], respectively, are the base of the recursive approach. DRUID used the three components recursive block, translation table, and persistent state. The recursive block represent protocol functions and is reused to implement layers of different scopes. One single recursive block exists, which can adapt its functionality (e.g., flow control, sequence control, compression, encryption, and fragmentation) to support various requirements. The translation tables link the recursive layer by mapping names between them and perform routing. The persistent state is used by the recursive block in the context of the translation tables. The project plan [TBD+11] gives an abstract overview without too many details about the protocols or how functions within the network are supported. DRUID is also supporting "the dynamic composition of services", uses a "recursion interface", and satisfies requirements with the capabilities provided by the system [TBD+11]. However, DRUID is proposing a single recursive block, which is repeated to implement layers of different scopes. Reusing aspects and the interworking between transfer and routing service are not explained. FoG is a solution for layers with intra- and inter-network scope and not for arbitrary scopes.

Pathlet [GGSS09] is a routing protocol, which operates over vnodes, pathlets, and forwarding identifiers. As stated in Section 3.2.4, the communication model is similar to FoG's model. However, Pathlet focuses on the routing aspect and provides source routes for packets. It does not include a special relaying PCI format or an interworking between transfer and routing service. Reuse aspects are implicitly included since it reuses pathlets for multiple connections.

Again: Isn't it just an MPLS plus IP re-mix?

A FoG-similar behavior might be achievable by stacking IP and MPLS packets recursively. Let's assume a FoG packet with a route containing a destination segment followed by an explicit segment followed by a second destination segment is mimicked with IP and MPLS. This requires the creation of an IP packet containing an MPLS packet that, in turn, contains a second IP packet. The stacking is required to avoid states on the destination system of the "outer" IP packet. The combination causes a lot of overhead for the repeated IP header fields (e.g. source address). Moreover, mechanisms to determine the stacking and the information required for it (addresses and LSP labels) would be required. Furthermore, all of the surrounding architectural questions, such as how to get the requirements of a connection, would have to be resolved. Basically, FoG is the solution for these problems. It builds upon the ideas of IP and MPLS (and others); but it is a little bit more than just the sum of its parts.

However, non-functional aspects of the reuse decision, state distribution of the transfer service, and dependencies between pathlets are not part of Pathlet. In particular, it is difficult to enforce policies restricting the set of customers, which are allowed to use such links. For example, the authors of [GGSS09] state that Pathlet would require a lot more states and cooperation between autonomous systems about how to set up their forwarding structure in order to enforce rules based on the upstream autonomous system/hosts. FoG uses the incremental routing process to deal with such issues.

SILO is an architecture that allows the dynamic composition of building blocks, called "services", based on application requirements. A "method" is an implementation of a service. A silo is a set of methods, which are constructed by a "service agent" for a connection. Each method provides service interfaces, called knobs, which allow the configuration of the methods by the service agent. A method seems to be comparable to a FoG gate, a silo to a chain, and the service agent to the transfer service manager. However, SILO and FoG differ mainly in two aspects. First, FoG supports explicitly functional blocks within networks by using a special hybrid relaying PCI format and an incremental routing process. The SILO architecture description does not mention such issues. Moreover, the authors use examples local to one system (example in Section II.b in [DRB+07]). Second, the reuse of methods seems to be not supported by SILO. Consequently, there is no class of functional blocks comparable to the FoG forwarding nodes.

The ANA framework defines *Information Dispatch Points* (IDP), which are abstract placeholders for functional blocks and information channels [KHM^{+}08]. An information channel represents lower layer connectivity between two ANA nodes. A functional block sends packets to an IDP and the IDP relays the packet to either another functional block or an information channel. The network can re-configure the linking between IDP and functional block and information channel, respectively, at runtime. FoG represents lower layer connectivity as gates and information channels are not required. Moreover, the communication model differs in the way functional blocks are connected. While IDP "identifiers" and gate numbers seems to be similar, ANA integrates the functionality of forwarding nodes in its functional blocks. Thus, a chain can only be constructed by knowing the implementation of the multiplexing functionality within an ANA functional block. Consequently, the separation of relaying and routing is difficult and not as easy as in FoG. Since each functional block in ANA has to decide about the next hop for each packet, the state information about connections have to be available at each functional block. In contrast, FoG combines such information in a route.

The solutions for dynamic stacks mentioned in Section 2.5, NetLets and SONATE, are mainly focusing on the selection and composition of stacks on end systems. Both are proposing algorithmic solutions for deriving stack "workflows" from application requirements. Thus, the protocol issues discussed in this book are orthogonal aspects. The FoG layer architecture provides a "place" for such algorithms in the routing service and collects input parameters for them such as available functions, connection requirements, and lower layer capabilities. However, FoG does not specify an implementation. The algorithm defined in Section 4.4.3 supports the claim that FoG is implementable but do not cover all aspects of the solutions in the state of the art. As discussed in the outlook, a combination of FoG and, e.g., SONATE, seems to be very beneficial in order to combine their advantages.

The *Role-based Architecture* (RBA) "provides a model for packet header processing, not a mechanism for routing packets" [BFH03]. While the algorithm is not defined by FoG as well, FoG defines the interaction between relaying and routing. A FoG routing service knows about existing functional blocks and system capabilities and can build its routing decision upon that information. It is also responsible for enforcing "sequencing rules" that define the order of functional blocks ("roles" in RBA). RBA does not define these issues required for a complete layer architecture. It may be possible to use the FoG layer architecture and modify the transfer service according to the RBA model.

The FoG layer architecture does not limit the mechanisms and protocols used for setting up gates. In particular, it may be possible to adapt existing QoS-signaling protocols such as RSVP and NSIS for this task. Moreover, protocols specifically designed for QoS and SLA handling between ISPs such as the *Automated Bandwidth Allocation across Heterogeneous Networks* (AutoBAHN) system [GÉA] may be useful to implement the gate setup.

A FoG route requires at least one node. It is either present explicitly as destination segment or implicitly as source node, at which the route starts. The route description of FoG degenerates to a list of concatenated edges represented by one or more explicit segments. It degenerates also to IP since even IP includes a single edge in form of the Protocol field. Forwarding a packet to a destination node is a means to an end for IP to get the data to the UDP/TCP connection end point.

3.4. Examples

The following three examples serve two different purposes. On the one hand, they demonstrate the rather abstract issues discussed before in a practical context. On the other hand, they emphasize the broad variety of supported network configuration with FoG. Therefore, the examples illustrate the differences and similarities with current networks. The first example shows a setup not possible with today's networks. It takes up the motivating use case from Section 3.1.1. The other examples show how FoG emulates an IP and a MPLS network, respectively. IP and MPLS have been chosen for comparison because they represent a connection-oriented and a connectionless network and are both in practical use.

One of the most important features of FoG is its flexibility. FoG can be configured in a way that it emulates current systems, such as IP and MPLS. If special configurations are deployed on all nodes in a network, the network will act like an IP and MPLS network, respectively. Without providing a mathematical proof, these reductions to homogenous configurations show that IP and MPLS are special cases of FoG. Moreover, the first example demonstrates that FoG supports a configuration neither supported by IP nor MPLS. Consequently, FoG's set of supported configurations is a real superset of the configurations from IP and MPLS.

Furthermore, the reduced FoG networks highlight possibilities for interoperability solutions, migration strategies, and implementation possibilities for FoG itself. While the former two issues are discussed in Section 3.6, the latter is the base for parts of the implementation presented in Chapter 4.

3.4.1. Motivating use case

The use case comprises an end system or an end subnetwork using functions within another subnetwork. In this example, Google requested prioritized relaying from Deutsche Telekom and relaying with guaranteed non-functional properties from AT&T. Google wants to decide which traffic to map to these transport capability. For example, live videos streams should be prioritized in

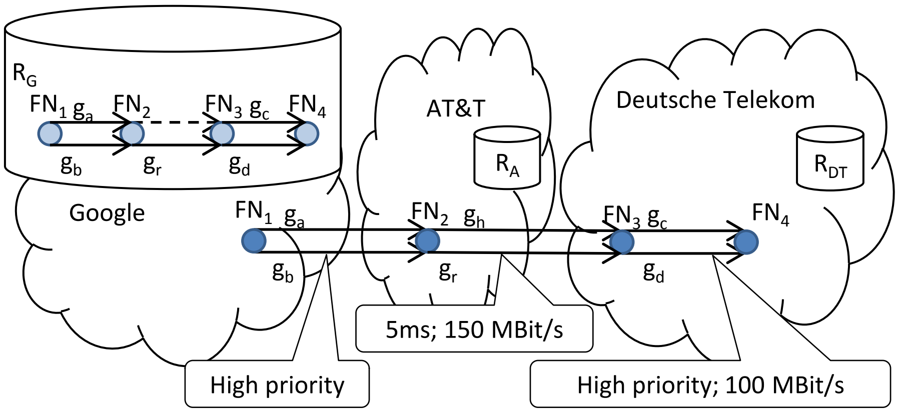

Figure 3.13.: FoG setup for use case. The available gates g_i in the transfer service are depicted as edges. Labels indicate the non-functional properties of a gate, if they differ from best-effort. Dashed lines indicate connectivity, for which the gate numbers are not known. Forwarding nodes FN_i are shown as vertices. The boxes R_i attached to each network represent the routing service entity of a subnetwork. The graph within the box represents the knowledge of the routing service entity about the scenario. To enhance the readability of the figure, the labels for the gate are not repeated, even though they are known to R_G.

order to improve the delay in networks with high load. Furthermore, all operators do not want to process signaling messages for each video transmission. Such messages would delay the start of a video stream and would increase the load of the relay systems in the networks of AT&T and Deutsche Telekom. More details are given in Section 3.1.1.

Figure 3.13 depicts the scenario as an example FoG inter-network with three subnetworks. The Google subnetwork uses functions provided by the subnetworks of AT&T and Deutsche Telekom. For simplicity reasons, the figure shows only the elements required for transmissions in one direction. The other direction can be implemented by adding more gates. The routing assigns addresses addr(FN_i) to forwarding nodes if required.

The routing service entity R_G of Google has been informed about gate g_a and g_b by its own transfer service. Previous signaling with Deutsche Telekom caused the creation of gates g_c and g_d. They are known to R_G either because R_G initiated the creation or R_G got informed by R_{DT}. For the same reasons, R_G knows about the gate g_r through the AT&T network.

The routing service entities of the function provider networks R_A and R_{DT}

are not shown in detail. They delegated the usage decision about g_r and g_d to R_G. In addition, Deutsche Telekom announces a best-effort relaying function g_c, which can be used by others as alternative for g_d. Gate g_c may be visible not only for Google but for other subnetworks as well. AT&T does not announce the gate number of g_h, which is its best-effort alternative for g_r. Thus, Google is forced to include the routing service of AT&T in best-effort routing decisions between FN_2 and FN_3. The example assumes that the routing of AT&T has enough information to calculate such routes.

3.4.1.1. Route opportunities

Without loss of generality, route requests from FN_1 to FN_4 are considered in the following. Other forwarding nodes FN_x beyond FN_4 can be reached by adding a destination segment with addr(FN_x) to the routes. FN_x may reside within the subnetwork of Deutsche Telekom or outside. In the latter case, FN_4 is a "border" forwarding node and gate g_c and g_d represent lower layer connections through the subnetwork.

R_G knows various routes from FN_1 to FN_4. At least four routes can be generated by combining g_a or g_b with g_r and g_c or g_d. Even more routes are possible, if partial routes are considered. Depending on the requirements of a connection, the routing can choose one. Some alternatives are discussed in the following:

1. [[g_b, g_r, g_d]]: This explicit route (with a single explicit route segment) provides the best QoS and could be used for live videos. Neither the achieved delay nor the data rate is guaranteed.

2. [[g_a], addr(FN_3), [g_d]] or [[addr(FN_3), [g_d]]: These are partial routes using best-effort relaying for transporting the data to Deutsche Telekom. Thereafter, the prioritized function in the network of Deutsche Telekom is used.

3. [[g_b], addr(FN_3), [g_d]]: This partial route uses prioritization from Google and Deutsche Telekom and relies on overprovisioning in the AT&T network. It might be an alternative for route 1 if the AT&T network is not highly loaded.

4. [[g_a], addr(FN_4)]: This is a partial best-effort route that requires Deutsche Telekom to decide about the usage of gate g_c on its own.

5. [[g_b], addr(FN_4)+20 Mbit/s]: This partial route prompts AT&T and Deutsche Telekom to setup additional gates supporting the non-functional requirement of 20 Mbit/s. Since both function providers delegated the usage decision of g_r and g_d, they are not allowed to map this new connection to these gates.

Neither AT&T nor Deutsche Telekom tracks the connections that Google maps to their gates. They just enforce the maximum throughput. Whether Google does the bookkeeping depends on Google's goals. Google could tread the gates as virtual links and map connections to them without checking the remaining capacity. Some kind of dynamic video encoding like *Scalable Video Codec* (SVC) might take over the responsibility to deal with the limitations. If the QoS for a connection is important, Google can decide to track the usage of these gates by mapping the connections using them to gates local to its transfer service entity. The local transfer service entity reports the current throughput of these local gates back to the routing service entity, which can use these measurements as approximation of the traffic traversing the remote gates. If, for example, Google decides to use route 3 for all connections using gate g_d, gate g_b can provide the approximation of the throughput. Exact throughput measurements might be reported to R_G by R_{DT}, e.g., periodically. They differ from the local approximation by the different packet sizes due to the variable packet header size (e.g. added authentication information as discussed in Section 4.3.2.1). If the throughput of single connections should be tracked, Google can create a local gate per connection, which is neither reused nor announced by R_G to any other R_i.

3.4.1.2. Discussion

The technique used to implement a gate is hidden by the gate number and the abstract gate description used by the routing service. For example, Google does not know – and does not need to know – the subnetwork protocols and algorithms used to enforce the non-functional properties for the gates in the network of Deutsche Telekom. This is equivalent to IP that hides the link layer technique. However, FoG additionally hides the non-functional aspects. In contrast to IntServ and DiffServ for IP, there is only one mechanism to use such QoS enhancements and not a combination of TOS field and classification states on routers.

As function user, Google takes over the responsibility to map its connections (and the connections for which Google participate in the incremental routing process) to gates known by its routing service entity R_G. No signaling is required as long as the parameters of the functions (e.g. maximum throughput) remain constant. The delegation ensures that Google can choose its rules for the mapping without notifying the function providers. Moreover, Google can decide whether to do bookkeeping on its own or to use the "virtual links" in a best-effort fashion. This flexibility of the function user is a central benefit of FoG.

The mapping states are separated from the function states. Google does not signal any mapping states to AT&T and Deutsche Telekom. The two function providers execute the function of their gates without knowing the connections

Not even a small mapping state on FoG relay systems?

Well, a relay node must know that a gate number x refers to functional block y. However, this mapping is omitted due to two reasons. First, it does not require any memory, if it is implicitly given. In particular, implementations using arrays of functional blocks and gate numbers as indices for such arrays store this relationship implicitly. Second, even if it is not so simple, a trivial mapping such as "for gate number x use functional block y at position z of an array/list" is expected to resolve this issue. The memory required for expressing this knowledge is limited by the number of gate numbers a functional block has. If functional blocks have a fixed maximum number of aliases, the number of such rules is limited by $O(\#\text{functional blocks})$. Thus, it has the same complexity as the function states themselves.

mapped to them. As intended by the design, they profit by not having to store mapping states. The movement of mapping states is even possible if a route is not explicitly defined for all hops. As shown by the partial route 3, addresses can be integrated in routes.

Partial routes such as route 2 and 3 are not implementable with today's protocols as discussed for the stacking of MPLS and IP in Section 3.1.2. While MPLS can take over the role of the first explicit route segment $[g_b]/[g_a]$ and IP the role of the destination segment addr(FN_3), the last explicit route segment $[g_d]$ cannot be stored in a packet directly. The transition from IP back to MPLS requires mapping states at the IP router doing this transition. In this example, it requires a mapping state on the IP node emulating FN_3. In contrast, a FoG route can include the functions after an intermediate address directly and, thus, enables the placement of mapping states on systems at the edge of an inter-network – at Google – or even on end systems. Since FoG operates over abstract functions, the flexible placement is possible not only for relaying but also for application-related functions.

In most cases, setting up gates requires signaling and takes time. If done in a reactive way the delay for the setup is similar to the delay introduced by today's resource management protocols, like RSVP. However, if done proactively, no signaling delay occurs and only the minimal delay for the route calculation is introduced. The proactive setup is especially useful for connections having best-effort requirements, since the routing can excessively reuse existing best-effort gates as shown in the example.

Section 3.7.1 picks up the use case once again and reviews the fitness of the FoG layer architecture for it in general.

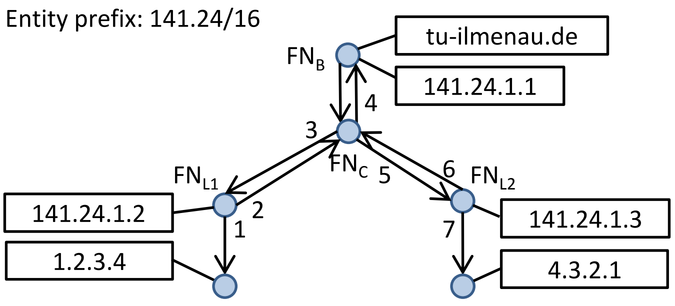

Figure 3.14.: Example routing service entity emulating IP forwarding

3.4.2. Emulation of IP

Only a subset of FoG's features is required for an emulation of IP packet relaying.[11] Especially non-functional aspects and most functions on relay systems can be omitted. Functional blocks on end systems are either representing TCP- or UDP-functionality. Best-effort connectivity between two FoG entities is represented as usual with gates representing (N-1)-connections. These gates are reported to the routing service, which can use OSPF or BGP (or both) to distribute the connectivity information among its entities.

The routing service uses IP addresses (either IPv4 or v6) as names for the forwarding nodes. Since IP names the interfaces, FoG has to create one forwarding node per lower layer attached to a transfer service entity. Figure 3.14 shows the routing service entity of a FoG node, which is attached to two lower layer (in other words: have two network interfaces). The node has one central forwarding node FN$_C$, which connects the two "network interface forwarding nodes" FN$_{Li}$. The names of these "network interface forwarding nodes" can also be used as binding names of the FoG entity at the corresponding lower layer (cp. step 6 in Section 3.3.7). The forwarding node FN$_B$ represents a higher layer binding with the name "tu-ilmenau.de".

ARP and MAC-addresses are part of a lower layer encapsulating an Ethernet. Both are not visible to FoG and – depending on the lower layer – may not be present at all.

[11]The text focuses on the most important aspect of IP, which is the packet relaying. Other issues, like fragmentation and ICMP, are not emulated. Maybe it is not possible to emulate all aspects of IP with FoG. However, I wonder if it is a good sign that not all problematic design issues can be emulated by FoG.

3.4.2.1. Forwarding

If a higher layer entity requests a connection to a remote higher layer entity, it specifies the name the remote entity used for its binding. The IP-like routing service translates such names to IP addresses obtained from its information base. Alternatively, it can access a DNS-like server. Depending on the available IP addresses, two alternative FoG-enabled DNS implementations are possible:

1. Only the "network interface forwarding nodes" have IP addresses. These addresses can be used to forward packets to the node. An extra name is required to identify the forwarding node representing the binding on the node. While the higher layer name would be suitable as extra name, a shorter version analog to the port numbers of the IP bindings would shorten the route. Even more efficient for FoG is the usage of the gate numbers required to relay a packet from the "network interface forwarding node" to the forwarding node representing the binding. In the example shown in Figure 3.14, a FoG-enabled DNS answers as follows: "tu-ilmenau.de" = [141.24.1.2, [2, 4]]

2. If the forwarding node representing the binding has its own IP address, a FoG-enabled DNS answers as follows: "tu-ilmenau.de" = [141.24.1.1]

Independent of the two alternative implementations, relay systems have to handle partial routes with at least one destination segment containing an IP address. Since IP is doing hop-by-hop forwarding, each transfer service entity has to contact its routing service entity, which just responses with the explicit route through the node itself followed by the original destination segment with the destination address. Thus, the routing service entity in Figure 3.14 might have the following FIB entry for FN_{L1}: prefix(4.3/16) = [2, 5, 7]. For each destination address 4.3.x.y with this prefix, the destination segment is replaced by [[2, 5, 7], 4.3.x.y]. Consequently, an incoming packet at FN_{L1} traverses the gates 2 and 5 on the node itself and leaves it through gate 7. The next node receives a packet with a destination segment as topmost route segment.

The reverse route is set on the source system and is not traced on relay systems. Analog to two alternative implementations of the FoG-enabled DNS, the reverse route can contain either

1. the address of the "network interface forwarding node" and an explicit segment with the route from this forwarding node to the forwarding node representing the connection end point or

2. the address of the forwarding node representing the connection end point.

3.4.2.2. Extensions

The basic emulation described before can be extended to model the IP behavior more closely. For example, a route can contain multiple destination segments with IP addresses of relay systems in order to emulate loose-source-routing. In order to emulate strict-source-routing, the requirements fields of the destination segments can be used to indicate that the destination segment specifies exactly the next hop address. However, strict routes are expressed more efficiently with explicit route segments.

The differentiation of traffic according to a priority field can be emulated with requirements specified in the destination segment. In contrast to IP, this even allows to set the priority for each part of the route individually.

3.4.3. Emulation of MPLS

In contrast to the IP emulation, the emulation of MPLS packet relaying is focusing more on the explicit route segment, which contains the label stack of MPLS. Gates represent one hop of LSPs. Despite the last gate of a LSP, they put the gate number of the next gate of the LSP on top of the stack of gate numbers[12]. Afterwards, they relay the packet through a lower layer connection to the next FoG node. Thus, these gates are comparable to the best-effort gates used in the emulation of IP with the extension regarding the route modification. A packet that "entered" one gate of the LSP travels through the gates until the end of the LSP without any routing service requests.

Not all these gates are reported to the routing service. The transfer service reports one gate per LSP. It starts at the forwarding node before the first gate and ends at the output forwarding node of the last gate.

3.4.3.1. Forwarding

Figure 3.15 shows an example network with one LSP consisting of three gates g_i. Gate g_i adds the gate number of g_{i+1}. Gate g_3 is the last one, which does not add a gate number. The routing service is informed about a single gate representing the whole LSP.

As described for the examples before, a higher layer requests a connection based on a destination binding name. MPLS itself does neither provide destination name nor a routing. It depends on a second protocol such as IP with a destination-based relaying PCI format that provides the information required to calculate an end-to-end path. As shown for the previous examples, such a relaying PCI format is possible. Therefore, this example assumes that the

[12] If the topmost segment of a route is a destination segment, the gates add an explicit route segment in order to enable the storage of the gate number.

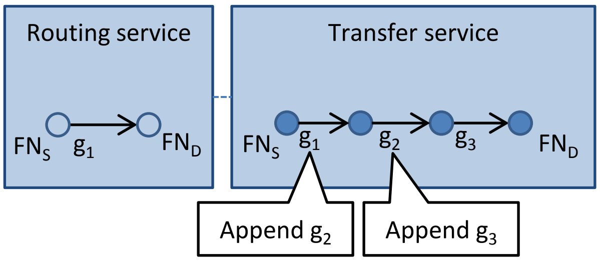

Figure 3.15.: Gate setup emulating MPLS relaying

routing decides to route to a binding name "tu-ilmenau.de" by using the partial route [[3], "tu-ilmenau.de"].

The packet starts at forwarding node FN_s, which relays the packet to gate g_1 by using the gate number from the explicit route segment. Gate g_1 adds the gate number of g_2 to the route and relays the packet to the next forwarding node. The process continues until g_3, which do not add an additional gate number. FN_D receives a packet with the partial route ["tu-ilmenau.de"] and has to request the next partial route from the routing service via its manager.

3.4.3.2. Extensions

Label stacks are emulated naturally by multiple gate number in an explicit route segment. If the route from the previous example should not stop after the LSP with the starting label of g_1 but go on with a LSP with the starting label x, the route has to be modified as follows: [[g_1, x], "tu-ilmenau.de"].

3.5. Political aspects

An inter-network architecture such as FoG is only subject to network neutrality in a narrower sense as defined in Section 2.4.5. The bundling issues are non-technical.

The features of an inter-network define "how much" traffic engineering it supports. Supporters of a neutral best-effort network favor an inter-network architecture, which does not support any traffic engineering at all. Since FoG includes mechanisms for handling functional and non-functional requirements, it may not be acceptable for them. However, the connection-oriented add-ons for the Internet presented in Section 2.4.2.2 and the classification of relaying PCI formats in Section 2.3.1 illustrate that it is very hard or even impossible to design a network which does not support any traffic engineering. Even today's Internet can support QoS.

FoG seems to be more acceptable for a neutral-offering network. It is in particular more suitable to implement a neutral-offering network than IP. This claim builds upon on two reasons. First, FoG supports QoS in an integrated and scalable way. A large number of QoS-enabled connections can be supported without the overhead required in an IP network. This lowers the costs of providing QoS, which, in turn, might lower the deployment hurdles discussed in the next section. Second, FoG enforces the neutral offering of QoS by covering its usage up. With FoG, a function user (e.g. an end user or a network operator) can request a gate with non-functional properties without specifying for which connection and, thus, service it is used. If the usage decision is delegated to the routing service entity of the requester, the architecture does not offer the provider of a gate a way to influence the usage of the gate any longer. Nevertheless, an operator can try to gain some knowledge about the connections based on the routes of packets traveling through a gate. Obviously, different connections have different routes. However, the reverse is not true. First, a route can contain variable gate parameters. Second, a function user can mask its own multiplexing decisions by generating different routes for all packets of one connection. This can be done, for example, by inserting random gate numbers that are ignored in the transfer service.

Traffic classification by more or less deep packet inspection is also more complicated as in IP. First, FoG does not use well-known port numbers. Thus, an efficient way to classify traffic is eliminated. Second, the names of the source and destination end systems can be hidden from relay systems easily by using a specific routes structure. Such routes include explicit route segments for defining the forwarding from the source to a relay gateway and from another gateway to the destination. The route in-between both gateways is specified by a destination segment. Since the routing service entities of relay networks normally do not known the details of the end networks, the relay networks are not able to determine the source or destination end systems based on explicit routes. The relay systems are only aware of destination names used for the part of the route between two gateways.

Besides the non-functional aspects of relaying, FoG enables a function user (which may be the end user) to have better control over the path its packets take. As outlined in [CWSB02, Sec. 3.1.4] and supported by the results of the *New Inter-Domain Routing Architecture* (NIRA) [YCB07], this enhanced control by users foster competition among network operators.

In summary, FoG masks natively the mapping between gates and services using them. This puts a high burden on those who try to violate the "neutrality". However, FoG does not solve the political issues. Although it supports or even suggests a "neutral offering" of gates, it does not enforce it. If the "neutral offering" idea should be enforced, the regulator has to ensure that an operator offers the same gate usage price to everyone. Furthermore, the prices for gates usage and the fraction reserved for best-effort traffic has to be subject to political decisions.

3.6. Deployment and interoperability

The deployment of new protocols and migration strategies from existing protocols to new ones are important commercial issues. If legacy network or applications should not be migrated, interoperability solutions become important as well. A combination of all these aspects can slow down or even prevent the introduction of new protocols. An example is the slow deployment of IPv6 [DFV07, CGKR10].

In this book, I am discussing a research proposal for a new inter-network architecture. Support for legacy applications or migration strategies from today's Internet to FoG are not my focus. This book defines a goal and not the "way" to this goal. If the goal fits the business and society needs, migration strategies will be found. As the history of IP shows, even the migration strategy of re-implementing applications for a new inter-network might be suitable in such cases. The history of OSI shows the great impact of business and political constraints. Technical aspects seem to play a minor role in the question whether an inter-network approach is adopted in practice. Thus, FoG's features that support business and political goals seems to be very important.

By its connection-oriented aspects, FoG favors business cases such as the motivating example introduced in Section 3.1.1. Its build-in authentication service enables a secure, dynamic signaling. Since the authentication question can be decided by the network automatically, the network can react to requests much faster. Such fast reactions may even be required by some use cases. For example, the ability to request QoS is much more useful for a user if it does not have to wait a week until multiple subnetwork operators agree on how to support its request. Furthermore, FoG supports the trend of providing arbitrary functions in the network. By distributing small functional blocks all over multiple subnetworks, it can be a base for a "distributed cloud". With a suitable configuration, FoG can enhance privacy since no source addresses are required. Even explicit routes are not hampering privacy. To alter explicit routes, a gateway can insert random gate numbers, which are ignored later on in the transfer service. As discussed in Section 3.5, FoG also helps to create a market for functions, which is neutral with respect to applications.

Whether all these aspects are sufficiently strong to open an opportunity for FoG is not clear. According to Handley "fear or greed" [Han06] are the main drivers for network operators to adopt new solutions. The question is whether they will be great enough in the future. The diverse und uncertain forecasts for the requirements of future applications (cp. Section 3.1.4) prevent an answer. However, the slow adoption of innovations in the last decades (IPv6, MobileIP, multicast, and more) indicate that the market forces are not as strong as in the creation time of IP. Connectionless networks have been pushed by the upcoming (personal) computers and by the bursty traffic pattern of their applications. Today, there seems to be no radical new traffic pattern that requires a different kind of network[13]. However, the trend towards (video) streaming applications seems to favor connection-oriented networks. FoG provides a suitable middle ground, which can combine the best of the connection-oriented and connectionless worlds.

Nevertheless, deployment and interoperability possibilities are important for the acceptance of a new inter-network architecture. Thus, the following discusses the influence factors for a real-world deployment of FoG. More details are given in the report of the "G-Lab_FoG" project [LVMT13]. A production-ready solution is not given and is subject for product development. Due to the dominant position of IP, a deployment in an IP world and interoperability with IP are considered, only.

3.6.1. Deployment

According to the general design of inter-networks (cp. Section 2.4.4), a FoG inter-network is deployed by making all gateways between its subnetworks FoG-aware. Internally, the subnetworks can use different network technologies, like IP. A subnetwork has to deploy FoG on its end systems that communicate via the inter-network as well, if interoperability scenarios should be avoided.

Gateways and end systems are updated regularly. Updates enhance either hardware or software of operators and users and can include the *deployment* of FoG. Operators have to buy new hardware regularly in order to provide sufficient capacity to satisfy the increasing demand. Thus, FoG-enabled hardware can be deployed in the long run via the regular update cycles for hardware. The same applies for consumer hardware such as notebooks, tablets, and mobile phones. For such devices, a pure software solution can be sufficient as well because the computational overhead for processing FoG packets at end systems is low. Updates for software are even faster than updates for hardware. For example, operating systems support update mechanisms with regular intervals of weeks or month. Special hardware is required only if the generation of signatures done by the authentication service should be speed up.

Due to the size of the Internet, there will be no "flag day" at which the whole Internet switches from IP to FoG. At least in the beginning, IP and FoG have to coexist and the FoG deployment will be partial. A partial deployment has to link FoG "islands" of various sizes through an IP-based inter-network. For example, two FoG subnetworks can be linked via gates representing tunnels

[13] The applications on mobile nodes show basically the same traffic patterns as the applications on fixed nodes. Thus, mobility requires a more dynamic handling of mapping information but does not influence the question whether it requires connections or not. Both are possible in mobile networks as shown by GSM and UMTS networks.

through the IP networks. This tunneling technique is used in the emulator implementation to represent tunnels through Ethernet as well (cp. Section 4.6). Moreover, it can be used by FoG end systems in order to connect to a FoG subnetwork if its own subnetwork does not support FoG.

In general, IP is just another lower layer for FoG, which is used to connect FoG entities. In other words, FoG operates as "overlay" over IP. However, the term "overlay" refers to the layering in the OSI or IP reference model and does not really apply to recursive layers. In recursive layer systems, every layer above the lowest one is an "overlay". FoG is not able to offer gates with all kinds of non-functional properties for these IP tunnels. The variety of gates is limited by the capabilities a best-effort IP provides.

The more FoG "islands" the more fine grain the "overlay" network. If two FoG-enabled subnetworks are directly adjacent, the IP layer can be omitted in favor of the lower layer encapsulating the link layer connecting both (e.g. carrier-grade Ethernet). Thus, a FoG inter-network reduces its dependencies to the IP layer gradually. If IP is deactivated or provided in parallel, depends on the use case. Maybe FoG, IPv4, and IPv6 have to be provided in parallel.

3.6.2. Interoperability

The previous discussion assumes no interaction between FoG and IP end systems. If both should interoperate, e.g., a FoG client requests data from an IP server, additional mechanisms are required. The mechanisms implement the interoperability either directly on end systems via an adaptation of interfaces or on gateways within the network.

3.6.2.1. Interfaces

A FoG-enabled application does not depend on FoG itself but on the interface for recursive layers. It can operate on an IP end system, if the IP suite is providing the same interface. The standard IP interface is similar to the Connection interface as outlined in Section 3.3.1. Its adaptation is a software engineering task and not related to protocols. The other subinterfaces, however, require additional protocols. The `bind` function requires a dynamic DNS-like protocol [VTRB97] to update the mapping from names to addresses. The neighbor-request operation requires a service discovery protocol (e.g. Multicast DNS [CK13]) to inform nodes about joining or leaving nodes.

An IP application can operate on a FoG end system, if the end system provides an adapter from the standard IP interface to the FoG layer interface supported by FoG. The adapter must generate names and requirements for the FoG layer interface. It can combine IP address, protocol field value, and port numbers to names and derive requirements from the type of transport protocol used.

3.6.2.2. Gateways

If the transition between FoG and IP is done in the network, a gateway between the FoG transfer service and the IP suite is required. Various implementations with or without a termination of the transport connection are possible.

Thomas Volkert analyzed the *interoperability* for the IP suite (TCP/UDP/ICMP) in detail [LV11]. His solution was used for the study with real network equipment that is described in Appendix D. In the FoG network, it requires a routing service that is aware of the gateway and that converts names of the Internet to routes leading to the gateway. For example, the routing service might translate the name "www.tu-ilmenau.de" to the route [addr(FN_g), 141, 24, 4, 226, 6, 80]]. FN_g is a forwarding node on the gateway and the remaining parts are commands for the gateway to connect via TCP[14] to port 80 of the IP node with the interface 141.24.4.226. In the other direction, from the IP network to FoG, the gateway has to announce the FoG bindings to IP. If the gateway is the destination for all IP traffic with the prefix 141.24/16, it can, for example, assign IP addresses with this prefix to the bindings available in FoG and report them to the DNS used in the IP world.

3.7. Discussion

The definition of a route is a unique aspect of FoG. Traditionally, relay protocols are designed for either a "route-based view" with a full source route or a "node-based view" with just a destination name. They are combined by stacking them. For example, a subnetwork uses MPLS and supports IP on top of it. FoG does not stack protocols but integrates the ideas of both in a single route. Instead of defining a whole route or defining a sequence of nodes, it picks some edges. Gaps between these edges are filled with information (e.g. a node name) required to forward a packet "over" the gap to the next known edge. Thus, FoG is following an "edge-based view".

This view includes the other two as extreme cases. If there are no gaps in a FoG route, it is similar to a "route-based view". If there are no explicit part and "only gaps", it reduces to a "node-based view". Depending on the number of edges specified, FoG supports both extreme cases and variations in between. This is not just a combination of IP and MPLS but provides a real superset of configurations. Thus, it can emulate IP- as well as MPLS-based forwarding (cp. Section 3.4) and provides additional configurations neither supported by one of these approaches nor by a combination of both (e.g. use case described in Section 3.1.1).

[14] Protocol field value 6 according to `http://www.iana.org/assignments/protocol-numbers/protocol-numbers.txt`

Possible solutions	Arbitrary functions	Flexibility			Scalability	
		Combining	Location	Autonomy	States	Signaling
FoG	+	+	+	+	− to +	− to +

Table 3.2.: Extension for Table 3.1 with the fitness of FoG for the use case

In the following, the FoG layer architecture is evaluated from two different perspectives. First, FoG is reviewed in the context of the motivating use case from Section 3.1. While an example solution was already outlined in Section 3.4.1, this section summarizes FoG's fitness for the use case based on the questions stated in Section 3.1.1. Second, FoG is discussed in the context of the reference model defined in Section 2.2.4.

3.7.1. Review of evaluation questions

FoG fulfills the evaluation questions for the motivating use case listed in Section 3.1.1 as discussed in the following. Table 3.2 summarizes the discussion.

- **Arbitrary functions:** FoG supports arbitrary functions explicitly by using functional blocks that take packets as input. The access to such functional blocks is enabled by gate numbers, which are independent of the function of a functional block.
- **Flexibility:**
  - **Combining functions:** FoG allows the combination of different functions by chaining functional blocks. Chains are implemented with gate number stacks stored in explicit route segments. A FoG packet indicates each functional block with a gate number in an explicit route segment. The numbers are neither predefined like "well known ports" nor do they require any negotiation with systems other than the system hosting a functional block. Thus, the functions represented by the functional blocks those gate numbers are listed in an explicit route segment are transparent for relay systems. A relay system cannot even derive the number of functional blocks chained by an explicit route segment (e.g. there might be parameters in as shown in Section 3.6.2).
  - **Location of functions:** FoG provides function users the opportunity to decide about the usage of gates known by their routing service.

They are free to map any connection to any gates as long as the dependencies of the communication model are fulfilled (cp. Section 3.2.2). The functions of the function provider handle only aggregated traffic from several undistinguiable connections. Thus, function providers have to give up some control. For example, Deutsche Telekom can no longer decide which (e.g. web traffic instead of video streams) and how many connections Google maps to the priority function. The limitation of control is beneficial to restrict the control of network operators due to regulatory issues (cp. Section 3.5). Moreover, the management protocol allows a function user to contact function provider of its choice in order to request a function. Partial routes allow a function user to direct its traffic to such a requested function even if relay subnetworks do not support the function.

  - **Autonomy of providers:** A function provider can define the set of functions supported by its systems autonomously. The capabilities are announced to the routing service, which is responsible to satisfy connection requirements with the capabilities of the systems. A function provider can influence the decision of a function user via its routing service policies. If it does not announce a gate or its gate number, function users are forced to negotiate with the provider about a function or to use a different provider.

- **Scalability:** FoG can provide scalable solution in case functions can be reused. If a connection requires dedicated functional blocks on relay systems, FoG performs only as good as an IntServ solution.
  - **States required for relaying:** Depending on the use case, FoG can move mapping states from function providers to function users. It achieves the flexible placement by setting up gates independently from connections and by encoding the states in routes, which are sent along with each packet. In particular, FoG avoids states on nodes those addresses are used in routes (as seen for the stacking of MPLS and IP) by its flexible route definition based on segments. It scales even better than an IntServ solution with aggregated states, because FoG has a more fine-grained control over the reuse of blocks in various chains.
  - **Signaling:** FoG supports requests for functions[15] independent from connection establishment. Signaling between function user and provider is required only for the setup of functional blocks. If the routing service of a function user knows about a block and its

[15] And, thus, requests for resources

gate number, it can be incorporated in a route without any further signaling. In the best case, no signaling is required and a connection is created at a peer implicitly via the first data packet. Similar to the situation for IntServ with aggregated states, however, the overall performance depends on the update/creation frequency of functional blocks.

3.7.2. Comparison with reference model

The recursive reference model proved to be helpful for the design of a new layer architecture. Most important is the enhanced isolation of a recursive layer, which enables most of the features of FoG. The most important features are the following:

- Selecting protocols and functions: The interface provides a higher layer a single service access point that hides the implementation of a layer. A layer is free to choose its implementation of bindings and connections. FoG chooses its communication model based on functional blocks and chains.
- Names and addresses: Since a higher layer operates on its own names and is not aware of the names used in a layer below, a layer is free to choose its addresses and the address format. FoG delegates this freedom to its routing service. The transfer service is independent of the addresses and provides a variable length field for transporting any address to come. Thus, addresses are not only private to a FoG layer, but private to the routing service of FoG.
- Data transfer and routing: The separation of both parts separates the frequently used tightly coupled mechanisms from the less often used loosely coupled ones. FoG follows the separation with its transfer and routing service. The separation increases the flexibility of the route computation and of the state distribution. The incremental routing process and the route definition exploit this flexibility.
- Access protocol: The protocol separates the EFCP setup from the connection establishment and enables the reuse of EFCP entities. FoG uses this concept in order to reuse already existing gates for chains. It also enables the parallel usage of a gate for multiple connections. The separation presents itself in the separation of the creation of gates and the usage decision for such gates. The setup of new functions on relay systems is implemented by the incremental routing process.
- Enrollment: The enrollment required for the routing and authentication service seems to be implementation dependent (e.g. address assignment).

The authentication service supports the enrollment of the transfer service by ensuring the authenticity and correctness of signaling messages. The authorization aspect of enrollment is part of the FoG gate setup. A node willing to join a layer has to request the creation of gates towards itself from other members. It will not receive any packets without such gates. A lower layer connection does not automatically create such a gate. A member of a layer has to create a gate representing the lower layer connection in order to make it available for packet exchange. The gate creation involves authorization.

FoG differs from the recursive model in the following points:

- Structure of management: The recursive layer architecture envisions a resource information base used by various management tasks. The information base of the FoG routing service is close to this. Since FoG combines the task of resource allocation and routing, most management tasks use it. However, it is not the only information base. The security aspects addressed by the authentication service seems to use an orthogonal second information base storing authentication entities. Since routing and authentication service operate independently they may use different RIEPs. Moreover, a third information base seems to be required for the policy rules. Transfer service entities (in particular those under the control of a single operator) might require another RIEP to exchange such rules.
- Routes: The reference model expects only single destination addresses or "sequences of (N)-addresses" [Day08a, Figure 7-19 (p. 276)]. It favors a destination-based relaying PCI format for implementing a recursive layer. However, FoG approaches routes in a different way by allowing explicit routes or explicit route parts as well. This is the key issue that enables a separation of states between the function provider and the function user. Explicit gate sequences do not necessarily limit load balancing, because a gate is not limited to represent a single (N-1)-connection. If a layer entity has several (N-1)-connections to the same peer (maybe even through different lower layers), a single gate can represent all these connections and balance the load among them. A prerequisite is that the non-functional capabilities of the (N-1)-connections can be aggregated in a useful way (e.g. all best-effort connection to a single gate, two 10 Mbit/s connections to a 20 Mbit/s gate).
- EFCP: The reference model favors a single EFCP for a layer. The EFCP is adjusted to the requirements by a policy. In FoG, however, the logic of the EFCP is implemented by gates. Depending on the set of gates supported by end systems, various different EFCPs might be in use in a FoG inter-network in parallel. The downside of this flexibility seems

to be that a special EFCP for the access protocol is required. The access protocol of FoG requires a single homogenous EFCP in order to secure its operations. In contrast to implementations of the reference architecture, FoG implementations seem to end up with at least two EFCPs: One supporting the access protocol and at least one encapsulated in gates.

- Functional blocks: The support for arbitrary functions increases the complexity of FoG. Examples are the potentially duplicated EFCP and the complex algorithm mapping requirements to functions. If application-related functions are moved to their own layer, the complexity of FoG could be reduced again. If this is a suitable solution for all functions (e.g. multicast packet duplication, virus check), seems to be an open question. In the worst-case, a function provided by all systems would require a second layer that has to duplicate all routing, resource management, and addressing aspects of the "original" FoG layer. However, even if application-related functions are extracted, the relaying function will remain in a layer. This reduces gates to "virtual links" that are combined for connections (cp. Section 5.5.3 in [Tan03]). The ideas of FoG can at least be used for the relaying functions and the QoS aspects. Most studies in Chapter 5 meet these concerns and favor relaying functions to demonstrate FoGs features.
- Distributed application name: FoG entities use a name with a common prefix (cp. Section 3.3.7) for their bindings at (N-1)-layers instead of a single distributed application name. FoG can find its peers via the `getNeighbors` function of the Layer interface. Afterwards, a FoG entity can decide which and how many neighbor entities it contacts in order to join the FoG layer. It does not depend on a multicast semantic of the names. The `connect` function of FoG's recursive layer interface interpret equal names as anycast name. A call of `connect` with such a name establishes a connection to one of these bindings.

4. Implementation of FoG

The *FoG Simulator/Emulator* (FoGSiEm) is an implementation of the FoG architecture that proves the feasibility of the architecture. Its core is a full-featured protocol simulator for FoG supporting large-scale simulations. Two important features enhance the simulator for two use cases. First, a graphical user interface can be attached to the simulator. It shows all details of a FoG network graphically and allows the user to interact with the network. The user interface is used for demonstration purposes. Second, the simulator can be switched to a real-time mode and can be attached to real network interfaces in order to exchange packets between computers physically. Thus, it can emulate a FoG network directly on top of existing link layers such as Ethernet. This feature is used to demonstrate FoG's ability to replace IP in real networks. FoGSiEm is a research-oriented implementation and not applicable for business. All its use cases are outlined in Section 4.1 in more detail.

The flexibility to add extensions to the core FoG simulator is achieved by a plug-in-based software architecture. FoGSiEm adopts the plug-in approach of the *Open Service Gateway Initiative* (OSGi) as its base. The *Eclipse Rich Client Platform* (RCP), which is also based on OSGi, is used as framework for the user interface part. More details about the software architecture and the available plug-ins are given in Section 4.2.

The Sections 4.3 to 4.5 present the implementations of the FoG services. Section 4.3 focuses on the transfer service with its relaying PCI format, functional block types, its access protocol, and error recovery algorithms. Moreover, a mechanism for mobility support based on tunnels is introduced. Section 4.4 describes two routing service implementations and an example algorithm for the requirement mapper component. The first routing service implementation simulates a real routing service without using a routing protocol. It is suitable for large-scale experiments. The second one is derived from the BGP implementation of the *Scalable Simulator Framework* (SSFnet) [SSF]. It provides a full-featured routing service for best-effort requirements. Both routing serivces utilize one algorithm that maps requirements to functional blocks by using a modified context-free grammar. Section 4.5 outlines two basic authentication service implementations. While the first one is just for simulation purposes, the second one uses encrypted hash sums and operates with public and private keys.

Section 4.6 describes the implementation of a second recursive layer, which encapsulates the access to real network interfaces. It is independent of the implementation described before and adapts Ethernet to a recursive layer. It is the base for emulating a FoG network directly over Ethernet without IP in between.

The implementation was created in the context of the project "G-Lab_FoG" and has been used for various demonstrations on workshops [VLBA12, VL12, LVMT11, LVMT10] and from 2010 to 2012 at the trade-fair CeBIT [Pre11, Weg10]. The software is available as open-source on the community platform GitHub under the terms of either the Eclipse Public License or the GNU Public License 2.0 [FoG]. Several colleague and students contributed to it. The acknowledgments list all participants in detail. The software architecture, the core FoG parts, the mapping algorithm, the simulated routing service, the adaptation of the BGP implementation, the authentication services, and the Eclipse integration have been created mainly by myself.

4.1. Use cases and design requirements

The implementation of FoG serves multiple purposes. First, it proves that FoG is feasible and can be implemented. Second, the implementation enables small-scale demonstrations of FoG. Such demonstrations have been required to present the status of the research project "G-Lab_FoG" and to visually show the features of FoG. Third, the implementation enables large-scale simulations required to analyze FoGs performance in inter-network scenarios. The results of the simulations are presented in Chapter 5.

These use cases are quite diverse. The first two use cases require an implementation considering all details of the protocols. In contrast, the third use case favors a more abstract implementation with fewer details in order to enable a larger network graph. Moreover, the requirements regarding a graphical user interface differ. The third and maybe the first use case do not require (or even exclude) a *graphical user interface* (GUI). Some statistical output is sufficient. However, the second use case requires some graphical user interface to literately show FoGs internals. In addition to a graphical user interface, a convincing demonstration has to include multiple computers transmitting real data over real links. Therefore, a pure protocol simulation is not sufficient. The implementation has to interoperate with the environment in order to exchange data physically between multiple FoG nodes.

The project "G-Lab_FoG" funded two persons for approximately three years to work out the theory and to come up with an implementation for all three use cases. The amount of features, the limited time, and the research characteristic of the project lead to an implementation of a rapid prototype. A rapid prototype trades runtime-performance for implementation time. Its features are implemented as proof-of-concept and not with a focus on performance.

However, the performance has to be good enough for the demonstration use case.

The programming language used for the rapid prototype has to enable fast implementations. First, it has to provide high level build-in features to avoid common implementation problems, like memory management. Moreover, libraries supporting the implementation have to be available for this programming language. Libraries for graphical user interfaces, graph handling, statistical output, and sending and receiving real packet directly via a link layer are required in particular. In addition, libraries for those protocols that can be reused to implement services and gates (e.g. BGP and TCP) are required.

4.2. Software architecture

FoGSiEm has a modular architecture based on components. The three use cases are supported by constructing FoGSiEm for each use case individually by selecting a suitable set of components. The component runtime system is provided by the Eclipse RPC framework [Ecl], which extends the OSGi framework [WHKL08]. An RPC application consists of a set of components, called plug-ins. Depending on the set of installed plug-ins, a RPC application provides different features. The composition of a set is limited by dependencies between plug-ins. All plug-ins listed as dependencies of a plug-in have to be included in a set in order to enable the dependent plug-in.

The plug-in structure of FoGSiEm follows a *model-view architecture*. The core simulation of FoG is the model. It is located in the plug-in called *fog*, which is described in Section 4.2.2. There are two views for this model. The first view is a graphical user interface for demonstration and debugging purposes. The second view is a pure command line one with textual status messages and statistics. FoGSiEm can operate with none, one, or more than one view depending on the installed plug-ins.

4.2.1. Plug-ins and extension points

The Eclipse RCP calls an enhanced OSGi *bundle plug-in*. While the difference between plug-ins and bundles is important for licensing[1], it is minor important for the remaining parts of this book. Thus, the term plug-in is used although some of them might actually be bundles.

Figure 4.1 shows the simplified dependencies of the FoGSiEm plug-ins that are most important for this text. Plug-ins are depicted as rectangles, and their names are shortened by the prefix “de.tuilmenau.ics” for better readability. A plug-in depends on the plug-ins drawn below it. For example, the *fog.bus.view*

[1]This issue is described in [FoG]: `https://github.com/ICS-TU-Ilmenau/fog/wiki/Licenses`

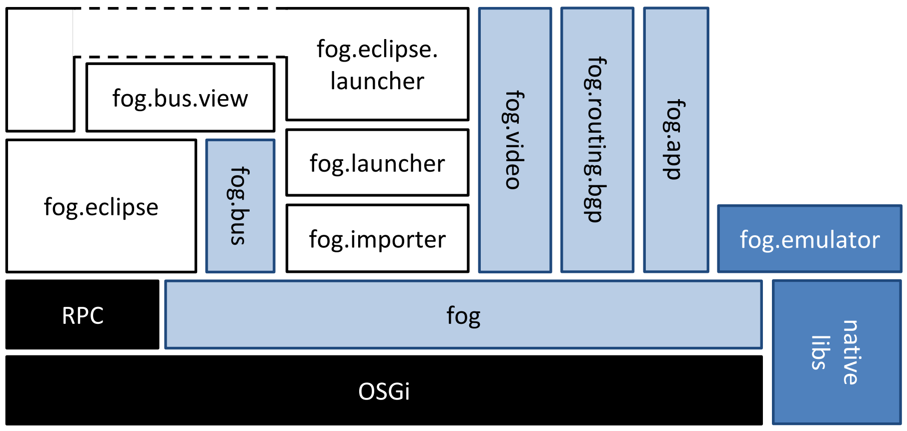

Figure 4.1.: Simplified plug-in dependencies of important FoGSiEm plug-ins. Black rectangles represent unmodified external dependencies. The medium light rectangles represent the emulator, the light rectangles the simulator, and the white ones the GUI parts. Rectangle sizes have no meaning.

plug-in depends on the *fog.bus* plug-in and the *fog.eclipse* plug-in. The plug-ins have the following purposes:

- *fog*: Core simulation of FoG entities as described in Sections 4.2.2. Moreover, the plug-in contains the FoG layer interface, a basic set of gates, all forwarding nodes, requirements and capability descriptions, the mapping algorithm, and basic implementations for all FoG services. The service implementations are described in Sections 4.3 to 4.5.
- *fog.app*: Applications using the FoG layer interface. An example is the stream client, which implements a constant bit rate data source.
- *fog.bus*: Abstract simulation of a lower layer. It simulates a broadcast medium, which allows direct communication between all attached FoG entities. The layer does not provide any functionality (like retransmission) and packets are subject to delay (constant or according to data rate and packet size), loss, and bit errors.
- *fog.bus.view*: Integrates graphical user interface extensions for the *fog.bus* plug-in into Eclipse.
- *fog.eclipse*: Various extensions to the RCP environment in order to provide views on a FoG simulation.

- *fog.eclipse.launcher*: Integrates the starter for simulations from the plug-in *fog.launcher* in the Eclipse launch framework.
- *fog.emulator*: Lower layer implementation for the scope of a broadcast domain that builds on Ethernet. It encapsulates FoG packets in Ethernet frames directly without an IP header. It depends on native libraries, which access the link layer of network interfaces via the operating system. Details about the emulator part are given in Section 4.6.
- *fog.importer*: Provides functions for importing scenarios from various file formats used by simulations. For example, it includes three importers for the graphs listed in Section 5.1.
- *fog.launcher*: Starts a FoG simulation. It configures a scenario with the help of an importer from the plug-in *fog.importer*.
- *fog.routing.bgp*: Routing service implementation for FoG based on BGP. The BGP simulation builds on the implementation from the SSFnet simulation framework. It has been adapted to FoG and, thus, uses the FoG layer interface to open connections between BGP entities. Furthermore, it reacts on gate reports from the transfer service. More details about the adaptation can be found in Section 4.4.2.
- *fog.video*: Provides the gates that handle video streaming.

More details and a list of all plug-ins are given in the documentation on the FoGSiEm homepage on GitHub [FoG].

A plug-in can provide *extension points* to other plug-ins. They enable other plug-ins to provide an *extension*. An extension enhances some internal mechanism of a plug-in. A common example is a list shown by the plug-in that provides the extension point. The list contains entries not from the plug-in itself but provided by extensions from other plug-ins. [GB04]

FoGSiEm provides, for example, an extension point called "fog.gateFactory", which enables plug-ins to contribute new gates. A complete list of all extension points of the FoGSiEm plug-ins can be found on the FoGSiEm homepage [FoG] as well.

4.2.2. Event simulation

The *fog* plug-in contains a discrete event simulation of the FoG core logic. It provides a detailed simulation of FoG, which follows closely the theoretical description in Section 3.3. Performance is not its primary focus. On normal computers, however, it is sufficiently fast for demonstration purposes as shown in Appendix D. The emulator parts, the user interface extensions, and some FoG applications add multithreading aspects to the event processing. Real-time critical aspects, such as receiving packets on real network interfaces, are done in parallel. Simulation-time critical aspects are executed in the event thread. The events can be executed in three modes:

- Stepwise: The events are handled stepwise. This mode is useful for debugging purposes.
- Real-time: The events are handled close to real-time if the execution duration of events allows it. It cannot guarantee real-time since the execution duration of an event can neither be predicted nor be limited. This mode is useful for interactions via the graphical user interface (e.g. for demos), distributed scenarios, and emulation.
- Fast: The events are executed as fast as possible. This mode is typically used for simulations without a graphical user frontend.

The simulator supports large scenarios by splitting a scenario to several smaller, interconnected scenario parts. These parts are distributed over multiple simulators running on different computers, which allow the usage of more CPUs and – more important – more memory than provided by a single computer. These advantages come along with slower communication between simulation elements because of inter-process communication. Such a distributed simulation does not require any software from the emulator part. It sticks to the normal simulation objects as provided by the plug-in *fog.bus* and links them via the *Java Remote Method Invocation* (RMI) middleware *Apache River* (formally known as JINI) [Apa]. The simulation times of the scenario parts are not synchronized and depend on the real-time mode of the event handling. If one simulation host is overloaded and not able to execute the events close to real-time for a long time, other parts of the simulation detect a timeout and FoG will act accordingly. If, for example, a gate is timed out since its creator, which resides in the delayed part of the simulation, does not send updates, the gate will be removed. If a packet would like to use this gate later on, the normal FoG error handling is executed. However, the scenario importer tries to avoid such long overload situations by distributing the FoG nodes of a simulation scenario equally[2] among the computers.

4.3. Transfer service

The *transfer service* is mainly implemented as described in the architecture chapter. It supports dynamically instantiated gates and error recovery mechanisms. It uses a simple request/response access protocol for prompting the creation

[2] It assumes that the number of nodes is an indicator for the amount of memory and CPU performance required for simulation.

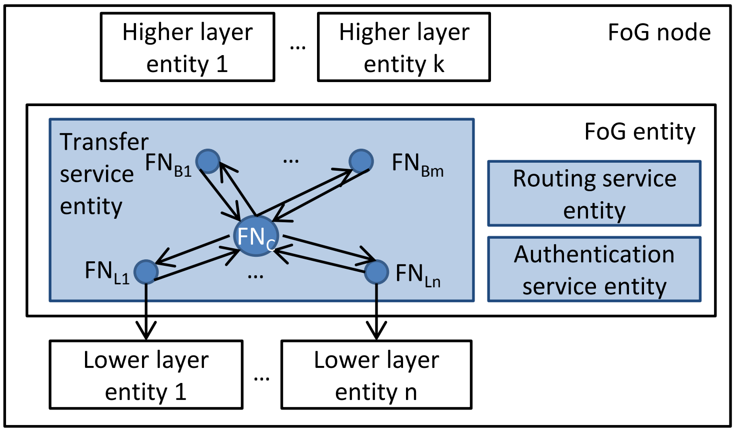

Figure 4.2.: Setup of a FoG node in FoGSiEm. Edges represent gates and vertices forwarding nodes.

of gates and connection end points on remote nodes. The management operations are secured by the authentication service. The transfer service supports mobility with the help of gates implementing tunnels.

Each *FoG node* in a scenario has a single *FoG entity*, which includes one entity of each service. The setup is depicted in Figure 4.2. Lower layer entities can be attached to the FoG entity. For each of these lower layer entities, the transfer service manager creates one forwarding node FN_{Lx}, at which all gates representing lower layer connections are attached. This structure simplifies the implementation but it is not required by FoG itself. The transfer service manager creates gates representing best-effort connections to all neighbor FoG entities reachable via a lower layer. These gates represent the capabilities of the lower layers and are reported to the routing service. They are reused for signaling messages and for best-effort connections. Each binding of one of the higher layer entities is represented with one forwarding node FN_{Bx}. A central forwarding node FN_C connects all other forwarding nodes with a star topology in order to enable transit traffic.

The basic operations of the transfer service are similar to the operations of a label switching router of MPLS that can create LSPs to all neighbors discovered via “hello” messages of the *Label Distribution Protocol* (LDP) [AMT07]. Such “hello” messages are also part of the emulator implementation as described in Section 4.6.

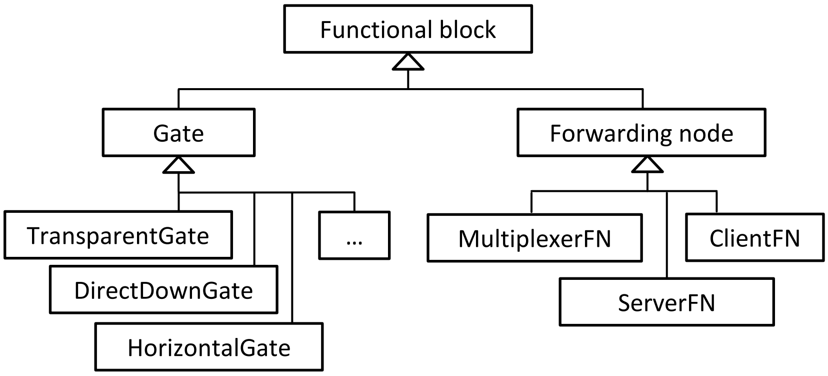

Figure 4.3.: Important functional blocks and their class hierarchy as UML class diagram

4.3.1. Functional blocks

Figure 4.3 shows a *Unified Modeling Language* (UML) class diagram with selected functional blocks available in FoGSiEm. While all forwarding nodes are listed, only the gates important for our discussion are included.

4.3.1.1. Forwarding nodes

In contrast to the FoG communication model, the implementation assumes that forwarding nodes relay a packet to the next gate without delay. Thus, the delay for processing the algorithm shown in Figure 4.6 and for requesting routes from the routing service is neither reflected in the routing service nor in the delay measurements of the simulation. The assumption is based on the observation that the delay caused by gates dominates the overall forwarding delay. The delay caused by forwarding nodes handling explicit routes can be neglected. High-speed IP networks show that even a routing service lookup may not cause high delays. Only if the routing service has to exchange information between its entities in order to decide about the route, the delay is significant. In such cases, the simulation delay measurements reflect the delay since the exchange of packets between routing service entities is simulated[3].

The following forwarding node implementations are available:

- MultiplexerFN: Normal forwarding node, which handles a set of outgoing gates. It performs the algorithm as shown later on in Figure 4.6 and reacts on signaling messages requesting the creation of gates.

[3] If a suitable routing service such as the FoG-BGP routing service is used.

- ServerFN: Forwarding node representing a binding of a higher layer entity. This forwarding node is labeled with the name given by the higher layer. In addition to the MultiplexerFN, it reacts on signaling messages requesting the creation of a connection. If permitted, it can trigger the setup of connections implicitly.

- ClientFN: Forwarding node representing a connection end point. It encapsulates SDUs from a higher layer in FoG packets and injects them in the chain of functional blocks. On receiver side, the SDUs are extracted from the packet and handed over to a higher layer. It reacts on signaling messages regarding the termination of the connection and errors in the transfer service. The ClientFN is not reported to the routing service since it is dedicated to a single connection and cannot be reused for multiple connections in parallel.

4.3.1.2. Gates

Gates contain a description that includes their characteristics and guarantees. In other words, the nature of service and the QoS are both included in the description. For example, a gate might support a data rate between 3 and 10 Mbit/s. The former is the guaranteed minimum data rate while the latter is the maximum. Both may come along with probabilities indicating their reliability. For example, the 3 Mbit/s can be guaranteed for only 99% of the time ("soft guarantees").

The functional description of a gate is given by its role to fulfill functional requirements. The functional requirements fulfilled by a gate are listed in its description.

The following subset of gate types is important for this book:

- DirectDownGate: DirectDownGates represent connections to other FoG entities via a lower layer ("downwards" in the stack). Depending on the lower layer, it can represent a connection through an Ethernet broadcast domain, an IP tunnel interconnecting FoG islands in a partial deployment scenario, and other transmission techniques.

- EncryptionGate/DecryptionGate: Both gate types implement an encryption of packets. They are an example for a dependent gate pair that modifies the content of a packet without requiring reverse gates.

- HorizontalGate: A HorizontalGate represents a tunnel through multiple other gates ("horizontal" within the same layer). It adds a (partial) route stored in the gate to the routes of all packets passing by. Such gates are used at the end systems of a connection to store the current route for the connection. Moreover, mobility solutions can use such gates to setup tunnels between mobile nodes and anchor points in the network. The tunnel semantic is also useful for hiding multiple intermediate functional blocks and to shorten routes.

- NumberingGate/OrderAndCheckGate: Both gate types implement a loss-free and in-order packet exchange. They are an example for functions that can be placed on end or relay systems or on both and that requires reverse gates. A NumberingGate adds a header that indicates the number of a packet in a sequence of packets to the payload of a packet. Moreover, it buffers the sent packets. An OrderAndCheckGate orders the packets based on the number in the header. It acknowledges receives packets and requests the retransmission of lost packets from its dependent NumberingGate. The feedback is implemented via its own reverse gate and the reverse route in packets.
- RerouteGate: The RerouteGate is used as replacement for a failed gate. It is derived from the HorizontalGate and acts as tunnel for a detour as long as the original gate is failed. If the original gate is no longer in FAILED state, the original gate replaces the RerouteGate again. Details are given in Section 4.3.4.
- TransparentGate: TransparentGates are helper gates used to connect two forwarding nodes within a transfer service entity. They hand over a packet to the next functional block without modifying it. Their descriptions indicate zero delay and infinity data rate.
- VideoDecodingGate: The VideoDecodingGate uses a C library in order to decode a video stream. It supports several codes such as H.261 and outputs uncompressed RGB video frames.
- VirusScanGate: Dummy gates, which actually does nothing. However, it is used as an example function with a single gate without dependencies.

A gate is in one of the following states:

- START: The gate was created and is waiting for the start of the initialization.
- INIT: Gate is in the process of initialization. It may communicate with its dependent gates in order to establish shared state.
- OPERATE: The gate is initialized and can provide its service.
- PAUSED: A temporary error occurred and the gate is not able to fulfill its service. The gate will switch to the OPERATE state, if the error disappears or is repaired.

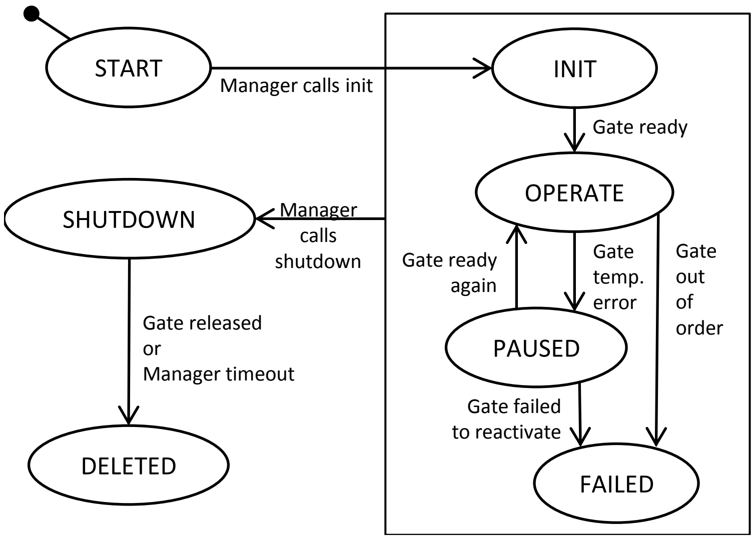

Figure 4.4.: State diagram for gates. The first word of a transition label indicates, who is initiating the transition.

- FAILED: A gate-internal error occurred and the gate it not able to fulfill its service any longer.
- SHUTDOWN: A gate is in the process of terminating its service and releasing its resources. It may communicate with its dependent gates in order to inform them about the termination.
- DELETED: The gate is no longer active. The transfer service manager will remove it from the transfer service plane.

The transitions from START to INIT and from any other state to SHUTDOWN are initiated by the transfer service manager. All other transitions are initiated by the gate itself. Figure 4.4 shows a state diagram with the conditions for transitions.

4.3.1.3. Dependencies

Dependencies are more limited than in theory. Only a single dependent gate per gate and direction (one peer and one reverse gate) are allowed. Dependencies to gate types are handled in the creation process. Details about this process are given in Section 4.4.3, which describes the algorithm that maps requirements to gates. The access to reverse chains is implemented via reverse gates and the reverse route of a packet. Therefore, the implementation ensures that a gate and its reverse gate share their previous and subsequent forwarding node as shown in the example in Section 3.2.3.

4.3.2. Packet structure and relaying

In order to support the communication model and the requirements of the architecture, a relaying PCI format that supports routes composed of multiple segments is required. A FoG route contains elements from hop-by-hop forwarding (destination segment) and explicit routing (route segment). According to the classification introduced in Section 2.3.1, a hybrid relaying PCI format is required for implementing the architecture.

The only known hybrid relaying PCI format used by PIP is not appropriate. PIP is mixing information from different scopes. While the tunnel field in the routing directive is used for tunneling through subnetworks, the routing hints are used for inter-network relaying. These hints are all equally long and separated by markers for changing hierarchy levels. They are intended particularly for hierarchical routing. However, storing routes, names, and addresses with significantly different sizes in this structure is inefficient. Moreover, the QoS information fields (logical router and handling directive) are not enough to encode arbitrary functional requirements. The protocol field is also not suitable for a recursive layer. It has to name higher layer connections and not higher layer types.

In summary, FoG requires a new relaying PCI format that supports its route format efficiently without the overhead introduced by just "tunneling" or "stacking" existing protocol headers. In the following, the FoG packet structure with its relaying PCI is introduced. According to the classification introduced in Section 2.3.1, it is a hybrid relaying PCI format due to its destination and route segments. It provides orthogonal QoS fields since non-functional requirements can be stored in optional fields in destination segments.

4.3.2.1. Packet structure

The FoG packet structure is shown in Figure 4.5. Its header comprises the relaying PCI with all information required by a forwarding node to decide about the next gate. It enables a forwarding node to make the forwarding decision before the packet is received fully. The trailer is optional and contains authentication information and a reverse route for answer packets.

The fields are defined as follows:

- Header
  - Header length: Length of the header in octets. This field allows accessing the payload directly. Furthermore, subsequent versions

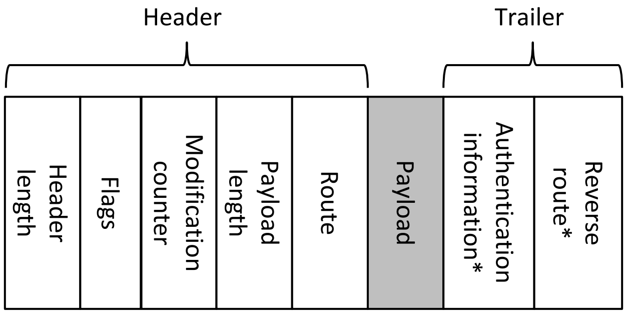

Figure 4.5.: FoG packet structure. Optional fields are marked with a star (*).

of the FoG protocol can add new header fields between the header fields defined here and the payload.

- Flags: Indicating the presents of optional parts that are located in the trailer. These flags are as follows:
  - Reverse route: Indicates if a reverse route in the trailer is present. It just indicates the presents of the field and does not indicate that the route should be traced.
  - Tracing reverse route: Indicates if a reverse route should be traced.
  - Signaling: Indicates if packet payload is a signaling message for the transfer service manager.
  - Authentication information: Indicates if the authentication information field in the trailer is present
  - Authenticate packet: Indicates if relay systems should authenticate the packet.
- Modification counter: Maximum number of remaining changes that might lead to a relaying loop. Relay systems that do not change the route or which extend the route must decrement the counter. If a relay system removes gate numbers, it does not need to change the counter. If the counter reaches zero, the packet must be dropped.
- Payload length: Length of the payload in octets. In combination with the length of the header, it enables access to the trailer.
- Route (variable): Stack of route segments

• Payload

- Trailer
  - Authentication information (variable): Optional list of authentication information
  - Reverse route (variable): Optional reverse route

A route is encoded with a starting length field that indicates the number of route segments. It is followed by the stack of route segments itself. Each segment starts with a type field indicating its segment type. Afterwards, a length field states the length of the segment in octets. The following list summarizes the structure:

- Route length: Number of segments
- Stack of segments; Each segment:
  - Segment type: Destination segment, explicit route segment, missing gate segment
  - Segment length: Size in octets
  - Content (variable)

In addition to the two segments defined by the architecture, FoGSiEm uses a third segment type called *missing gate segment*. Routing service entities use this segment type to request additional gates from transfer service entities requesting a route. This third segment type is an implementation artifact and may not be present in other implementations. It can be avoided by introducing another interface that can be used by the routing service to request gates from the transfer service.

The authentication information is formated like routes as a list of signatures. The structure is as follows:

- Authentication information length: Size in octets
- Stack of signatures:
  - Signature type: Type of signature in oder to enable the authentication service to use multiple types of signatures.
  - Signature length: Size in octets
  - Signature (variable)

The implementation uses a Java class with the fields mentioned above as packet. The serialization of a packet is done via the default serialization supported by Java. Since this mechanism adds some overhead (e.g. Java class names), no length values for the constant parts are given. An example for this overhead is shown in Appendix D.2. An optimized implementation can improve the length of a serialization significantly by a customized serialization based on a more detailed version of the packet structure.

4.3.2.2. Reverse route

The reverse route field contains the route from the receiver back to the sender traced during the relaying process. It can be used by the receiver to send an answer back to the sender. The main benefit of using a reverse route instead of a source address is twofold. First, routing requests for answer packets are avoided and, second, addresses for sending nodes are not required. The latter is useful for systems those applications acting only as clients. A server can send an answer back by using the reverse route without forcing the client to have an address.

The reverse route is used to determine bidirectional routes for connections with requirements. If an end system receives an answer packet with a traced reverse route, it knows the route to the answerer. In most cases, this route contains less destination segments and more explicit route segments as the original request packet sent by the end system. Therefore, the end system can use this route for subsequent packets in order to reduce the load for routing and the delay for its packets.

The reverse route has to include the reverse gates required by the forward route as described in the communication model. It does not need to be symmetric to the forward route. However, most reverse routes in the implementation run in parallel to the forward routes. An intermediate forwarding node that is not able or allowed to record the reverse route can insert a destination segment to the reverse route instead of a gate number.

4.3.2.3. Relaying process

Each forwarding node processes routes of packets in order to relay them to the next gate. This process is part of the communication model described in Section 3.2. In the following, the steps required by the implementation and based on the new relaying PCI format are explained in detail. In order to simplify the description, it does not differentiate between forwarding nodes and transfer manager entities.

Figure 4.6 shows the actions of a forwarding node per input packet. If the route is empty, the packet reached its destination. If the signaling flag is set, the packet contains a signaling message in the payload. The signaling message is handled by the local transfer manager in the context of the forwarding node. If the signaling flag is not set and the forwarding node represents a connection end-point, the payload is extracted and handed over to a higher layer entity. If the route is not empty, the forwarding node checks the type of the topmost segment. If it is a destination segment, it requests the next partial route from the routing service. The response of the routing service replaces the destination segment and the forwarding node restarts its relaying process. In order to prevent loops in the routing service lookup process and in the incremental

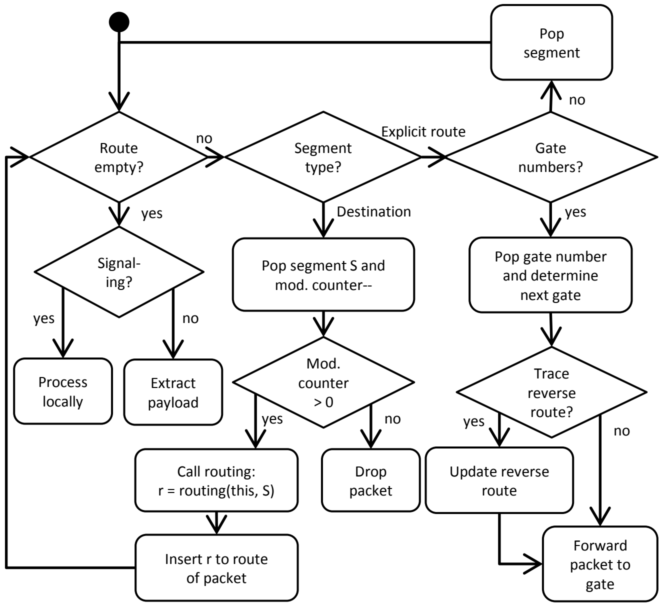

Figure 4.6.: Forwarding node procedure without error cases

routing process, the modification counter is decremented. If the counter reaches zero, the packet is dropped. If the topmost segment is an explicit route segment, the number of remaining gate numbers in the segment is checked. If there are gate numbers left, the topmost is removed from the segment and used to lookup the next gate. If no gate numbers are left, the segment is removed and the process restarted.

The missing gate segment that was mentioned in Section 4.3.2.1 is omitted in this description, because it is highly implementation specific. Basically, it introduces one more option for the check of the segment type.

The reverse route of a packet is recorded if the reverse route flag in the packet header is set. A forwarding node has to derive the reverse gate number from the gate chosen for the forward direction. If the reverse gate cannot be determined, a forwarding node inserts its routing service address as destination segment. If it does not have one or is not allowed to reveal it, the next steps depend on the topmost segment of the reverse route. If is a destination segment, the forwarding node can relay the packet without further changes. In all other cases, it has to remove the reverse route and re-set the flags.

4.3.2.4. Minimal relay process

The overall process is more complex than its pendant in a MPLS network. Thus, the question arises if the relaying PCI format of FoG can be used by backbones. Depending on the network setup, the actions of the relay process are performed more or less often. By announcing all its gates, a FoG node can reduce the number of partial routes ending at its transfer service entity.[4] Consequently, destination segments have to be handled less often and explicit route segments more often. If gateways ensure that all incoming packets contain explicit routes through their subnetworks, relay systems in the subnetwork do not have to handle destination segments at all. Thus, the handling of topmost explicit route segments determines the performance of backbones.

The first four header fields are fields with a fixed length, which can be accessed directly. The route is the first field with a dynamic length. Since the first fields of a route are also of fixed length, the first gate number of a topmost explicit route segment can be found at a fixed offset. This gate number must be removed and the route length value has to be decremented. If the packet has to be available as a single memory block for sending, the "prefix" of constant length before the gate number has to be shifted in memory by the length of the gate number. Thus, the basic relay process has a complexity of $O(1)$. Since the gate number was chosen by the relay system itself, the system can choose a gate number with a length suitable for its CPU and memory architecture. A 32 bit system may, for example, select a gate number with 32 bit length. The required operations (moving, decrementing) seem to be simple enough that the basic relay process can be implemented in hardware.

The reverse route update is more complex. If it is present, is determined by the flags, which are at fixed offsets in the header. If the reverse route is encodes in the same way as the forward route, the start offset of the reverse route can be determined as follows:

$$\text{Offset authentication information length} = \text{Header length} + \text{Payload length} \tag{4.1}$$

$$\begin{split}\text{Offset reverse route} = \text{Offset authentication information length} + \\ \text{Authentication information length}\end{split} \tag{4.2}$$

The revers gate number has to be added at the front of the route. If a packet should be available as a single memory block, the variable length part of the reverse route has to be moved and the reverse route length has to be

[4] This is a simplified view. More details about this issue are given in the performance study described in Section 5.3.

incremented. In order to avoid this copy operation, it might be more efficient to encode the reverse route in reverse order. Thus, the fixed length parts of the route are at the end of the packet and adding a reverse gate number would be as simple as removing a gate number from the forward route. End systems would have to reverse the order of the gate numbers of the reverse route in order to use it as forward route. The reverse operation requires the definition of a standard gate number size in order to reverse the octets of an explicit route segment. For example, the size of a gate number can be defined as one octet. If a relay system requires larger gate numbers, like two octets, it has to insert the octets in a way that the reverse operation arranges them correctly (e.g. correct order for little/big endian).

4.3.3. Access protocol

The FoG *access protocol* is used to assign connections to chains and to maintain functional blocks independently of connections. It is a simple request/response protocol that includes relay systems by exploiting the incremental routing process. The protocol does not support a negotiation of the requirements used for a connection. In case the destination or a relay system participating in the incremental routing process rejects a request, an indication about the reason for the rejection is send back. Based on this error indication, a requester may decide to initiate a subsequent modified request.

The access protocol follows an "at-most-once" sematic. The sender includes a *signaling process identifier* in each request. The identifier combines the sender's authentication service name and a number, which acts as a local label for the process. If the sender does not receive a response, it sends the request with the same signaling process identifier once again. If the receiver receives a request with a known identifier, it sends the response without performing the requested action twice. Figure 4.7 shows a small sequence diagram with one lost response message.

The EFCP for the access protocol is integrated in the access protocol. First, the access protocol includes a message counter to separate different messages for the same process. Second, the check for authenticity and the check sum involved in it detect bit errors. Since the protocol is concerned with functional block creation, the implementation stores the signaling process identifier and the message counter in the signaling states of the gates created by the request. The same signaling process identifier is also used in update, keep-alive or release messages. However, a request with a known identifier is accepted only if the message counter of the request is higher than the last message counter stored by the receiver. This ensures a logical order of requests.

The two-way-handshake can be used to setup gates for bi-directional and uni-directional communication. The behavior depends on the requirements listed in the request message.

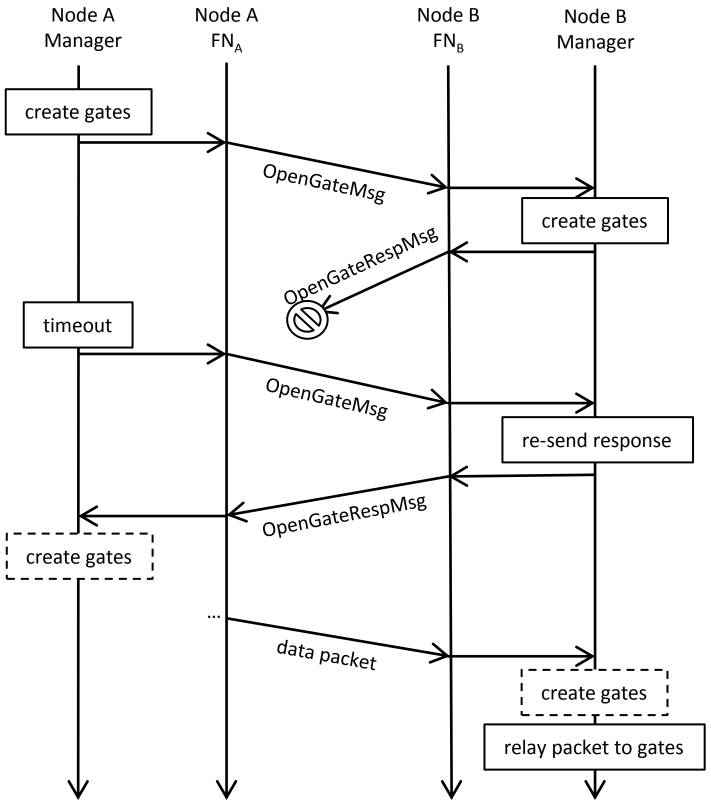

Figure 4.7.: Sequence diagram for gate setup with one lost response message. Dotted rectangles indicate point in time for lazy creation.

Depending on the policy of a node, it creates gates immediately or "lacy". As shown in the sequence diagram in Figure 4.7, node A can either create the gates before sending its request or after a response from node B. The lacy creation is preferable if a lot of gates have to be created and the rejection probability is high. Node B may delay the creation of its gates to the arrival of the first packet for its gates. However, it has to reserve gate numbers and resources for the gate(s) immediately. If delay guarantees are involved, the lacy creation must not exceed these guaranteed delays.

A node can trigger the start of a signaling process on another node by sending a request for a gate requiring a peer or reverse gates without information about these dependencies. If a node receives such a request, which does not include any information about the peer/reverse gates located at the requester, the node will respond with a request for these gates on its own. This procedure extends the normal two-way-handshake to a three-way-handshake. It is especially useful for situations where intermediate gates should be set up but the route of the first packet is subject to detours. Such detours occur, if the access protocol has to determine the location of a binding by contacting nodes not on the direct path between the source and the destination (cp. Section 2.3.3). The second packet is assumed to take the best way back to the requester. However, a node might refuse this procedure due to its policy.

The implementation supports gates with hard-states and soft-states. Gates with soft-states are released if no keep-alive signaling message is received during a time period. Gates of both types can be removed with release signaling messages. This mixture of soft-states with the possibility of explicit release is recommended in [JGKT07] as the most efficient strategy.

4.3.4. Error recovery

Layers using destination-based relaying PCI formats perform a late binding of connections to routes. Each relay system decides about the next edge a packet is send to at the latest point in time. Layers with route-based relaying PCI formats perform an earlier binding since the route is generally chosen before a packet reaches a relay system. Both layers react differently in case of a link or system failure. Destination-based relaying PCI formats require all relay systems on the alternative route to participate in a recovery. Each has to decide to relay a packet on an alternative route. In particular, no relay system on the alternative route should relay a packet back to the failed route. For example, if the first relay system on the alternative route is sending packets not on an alternative path but back to the failed path, the *detector* of a failure is not able to perform a recovery. In general[5], recovery is done by exchanging routing information and to wait until the routing decisions of all systems converge again. Route-based relaying PCI formats allow the specification of routes in advance and, thus, can perform an additional recovery mechanism before routing converges. The detector can specify an alternative route directly in the packet. The mechanism is transparent for the relay systems on the alternative route, because they do not have to be aware of the recovery process. However, the mechanism assumes that the detector has enough information about the network graph to be able to specify such an alternative route.

FoGSiEm provides three different *recovery algorithms* to handle single gate failures. Forwarding node failures can only be handled under specific circumstances. The algorithms are known from literature [JG06] and have been adapted to FoG. In the following, their reaction to a single gate failure (“stop-

[5] An evaluation of alternative solutions for IP is given in [GRY07]. However, most of these solutions tend to use “route-based ideas” in order to define a detour (e.g. an IP source route).

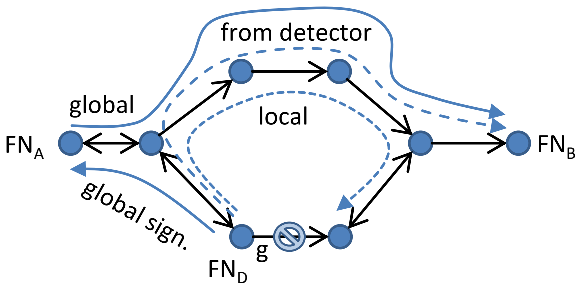

Figure 4.8.: Repair algorithms in a small scenario. Bidirectional edges represent gates in both directions.

fail errors") is discussed. Afterwards, the failure of a FoG entity, which causes a multiple gate and forwarding node failure, is discussed separately. Byzantine failures and multiple gate and/or forwarding node failures that are independent of each other have not been analyzed.

All algorithms assume that failed gates do not have any dependencies. The algorithms are used in particular to repair failed gates that represent lower layer connections as shown in the study in Section 5.4. Failures of gates with dependencies are recovered by the source. It sets up a new connection after getting an error signaling message (cp. global repair) or after an idle timeout.

In the following, the three algorithms are explained. Figure 4.8 depicts their ideas based on a small example graph of functional blocks. It shows a detector forwarding node FN$_D$ that detects the failure of gate g, which is part of the chain between FN$_A$ and FN$_B$.

4.3.4.1. Local recovery

The local recovery algorithm tries to bypass a defect gate locally. It calculates a route, which starts at the forwarding node FN$_D$ and ends at the outgoing forwarding node of the failed gate. This repair is transparent for the connection end points FN$_A$ and FN$_B$ (ClientFN forwarding nodes). Figure 4.8 depicts the idea with a dashed line with short spaces.

The algorithm depends on the routing service to calculate the bypass route. Therefore, the transfer service reports the failure of the gate before it requests the bypass route. The requirements for the bypass are derived from the description of the failed gate. The name of the destination forwarding node of the failed gate is used as destination name for the route request. This name has either to be exchanged upfront during the gate setup or it has to be known by the routing service entity of the transfer service entity hosting FN$_D$.

If the policy of a FoG node allows it, it creates a RerouteGate and replaces the failed gate with it. The replacing is executed as follows. A forwarding node detaches the original gates from its gate number and attaches the ReroutingGate to the very same gate number. The original gate is not terminated but no longer directly accessible from a forwarding node. The RerouteGate acts as cache for the bypass route and, thus, allows an efficient re-routing of multiple packets. The RerouteGate checks the status of the failed gate. If the failed gate can be repaired (e.g. by setting up the lower layer connection again), the RerouteGate removes itself and restores the former gate again. If required, the restored gate is reported to the routing service.

Due to its simplicity, the local repair is the default recovery algorithm of FoGSiEm.

4.3.4.2. Recovery from detector

The from-detector recovery algorithm calculates a new route from the detector forwarding node to the destination. Instead of locally bypassing the defect gate, it searches for a new route directly to the destination FN_B. This route replaces the remaining route in a packet. Figure 4.8 depicts the idea with a dashed line with large spaces.

This algorithm is applicable only for special individual packets. The prerequisite is that the routing service is able to determine the destination of a packet. There are two ways a routing service knows about the destination. First, it can determine the destination forwarding node based on the explicit route of the packet. It requires sufficient knowledge about the graph of functional blocks in order to find out. For realistic network setups, most routing service entities will not have enough information. Most probably, this method will only be applicable to failures within a subnetwork for which a routing service entity has the complete knowledge. Second, a packet can specify its destination directly via a destination segment in its route. Thus, this recovery algorithm is mainly suitable for the first packets of a connection that are subject to the incremental routing process.

4.3.4.3. Global recovery

The global recovery algorithm does not try to repair a route locally. It drops all packets that try to pass a failed gate. The source of the packet is informed about the failure via a signaling message that is send along the reverse route. This message is depicted with a straight line from FN_D to FN_A in Figure 4.8. The source has to request a route for its connection starting from the very beginning.

If no reverse route is given in a packet, the detector forwarding node does not send a signaling message. Moreover, the policy may restrict sending signaling messages too often. For example, a transfer manager may send only one signaling message per individual reverse route per time interval. Such restrictions avoid sending too many signaling messages while the first one is on its way to the source.

The algorithm assumes the routing service to converge fast enough that the routing request initiated by the source can be answered without using the failed gate. In case the routing service is not fast enough, the new route might use the failed gate once again. This either leads to a second error response or to a from-detector recovery.

4.3.4.4. Forwarding node failure

A forwarding node failure occurs, if the transfer service entity that hosts the forwarding node fails. Thus, not only one forwarding node but also all other forwarding nodes and all gates of the failed transfer service entity are lost. Except the forwarding nodes representing end points of a connection or bindings, the failure of the forwarding nodes themselves is a minor issue. Their multiplexing functionality can be compensated in other transfer service entities easily. More important are the gates attached to them. Thus, a forwarding node failure is in most cases equivalent to the failure of all its gates. In other words, a forwarding node failure triggers a multiple-gate failure. If a forwarding node failure is reported to the routing service, it removes the forwarding node and all adjacent gates from its graph.

It might be impossible for a detector to find the root cause of a failure. A detector, which detects that a gate representing a lower layer connection failed, may not be able to determine the state of the forwarding node "behind" the gate. Basically, it depends on the implementation of the lower layer whether a gate and a forwarding node failure can be distinguished. If the lower layer entity on the detector still acknowledges the existence of the binding of the remote transfer service entity, the detector has to assume that only the gate (meaning the lower layer connection) failed. Thus, if FoGSiEm reports either the gate or the remote forwarding node as failed depends on the lower layer. Consequently, not all failed functional blocks may be detected. The more failures are not detected, the higher the probability that repair actions fail or trigger additional repair actions.

The global and the from-detector recovery algorithms do not require special treatment of forwarding node failures. The local recovery fails because the destination forwarding node of the bypass route is not available. Therefore, the local repair algorithm of FoGSiEm uses the first forwarding node "behind" the failed forwarding node as destination for the bypass. This is only possible, if the routing service can determine the forwarding node based on the route of the packet. The same limitations as described for the from-detector algorithm apply here. Moreover, the result can no longer be cached with a RerouteGate.

4.3.5. Mobility

The reference model treads *mobility* as dynamic multihoming. Systems change their attachment points and reflect this by changing mappings. For example, if a system gets a new address due to its new position in the (inter-)network, the mapping from names to addresses is changed. Such changes are enabled by encapsulating the addresses and the mapping within a layer. The performance depends on the update performance of the mapping information base.

A transfer service support for mobility is required if the mapping update is too slow or if it is not sufficient for already established connections because they rely on cached addresses or explicit routes. Since mobility itself is not part of the FoG communication model, the chains affected by a teardown of an attachment point have to be adapted to the new location of the mobile system. A straight forward solution, which mimics the behavior of MobileIP [Per10], uses tunnels between the mobile system and a fixed "anchor point" in the network. The tunnel is represented by one gate per direction. One of them has to be included in every chain. A mobile system has to update the tunnel gates once per handover in order to re-establish all its connections.

The transfer service implements the tunnel solution by providing "wrong" information to the routing service. Figure 4.9 shows the setup for a mobile end system M and a relay system A that acts as anchor point. The important point is the difference between the transfer service entity T_A and the routing service entity R_A in node A. T_A reported a named forwarding node FN_B reachable by a gate g_t to R_A. However, the gate g_t is a HorizontalGate representing a tunnel to node M. If the routing service includes g_t in a chain, each packet traversing g_t is redirected to the forwarding node FN_B, which represents the real binding on node M. The tunnel hidden by g_t is not visible to the routing service. If node M changes its point of attachment, it has to inform node A about the new route for gate g_t. Moreover, it has to update its own gate g_h with the route to node A in order to update its sending direction. The authentication service ensures that only the mobile system that requested the creation of the tunnel is able to update the route of the tunnel.

Like MobileIP, this scheme has some disadvantages. Most important, it requires the tunnels in both directions. A "triangular routing" as in MobileIP can be implemented with a destination segment in the reverse route, which is traced on the way from a correspondent node to a mobile node. However, that prevents the usage of gates with QoS capabilities and may lead to the same security problems as known from MobileIP.

The location of the anchor point and, thus, the impact of the tunnel depends on the scenario and the scope of the FoG layer. In a subnetwork scope, a mobile system typically registers its named forwarding node at the gateway of the subnetwork it is attached to. If the mobile system is communicating with an end system in another subnetwork, the anchor point is on the route

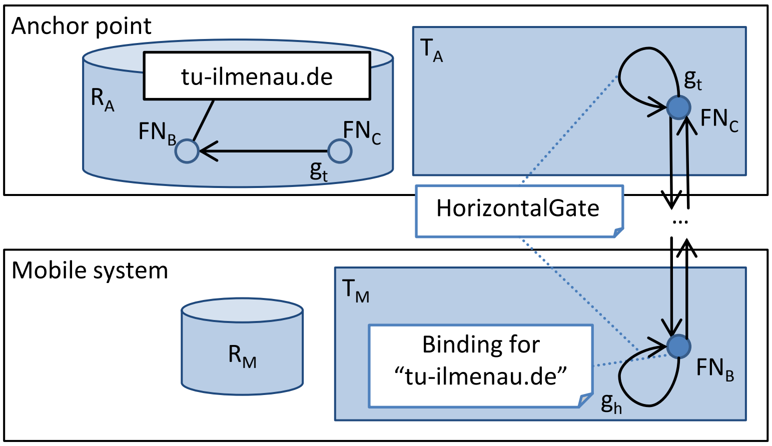

Figure 4.9.: Gate and forwarding node setup for mobility

of the packets. Therefore, handovers within the subnetwork can be supported efficiently. It may even be possible to introduce multiple smaller tunnels. For scenarios combining MPLS and IP, I present an algorithm for optimizing the location of anchor points in a subnetwork in [LBMT06]. Similar solutions may be applicable for FoG. However, research in this direction is subject for future work and not include in my thesis.

Since the mobility approach is so close to MobileIP, I desist from a quantitative comparison. The qualitative discussion shows that mobility support is possible although it was not a design goal.

4.4. Routing services

The implementation contains two routing services that are important for the performance studies. The first one simulates a real *routing service* and does not use a routing protocol. It is used for analyzing the amount of information a routing service would have to store. This routing service is used for all studies presented in Chapter 5. It stores the detailed and abstract information about the transfer service in one graph each. Although is support a hierarchical arrangement of information, it does not support addresses to aggregate location information. The second routing service is an adapted BGP implementation from the SSFnet simulator. It supports a full featured BGP protocol implementation. It was adapted to FoG by using the FoG layer interface to connect to neighbors and by naming forwarding nodes with IP addresses. It uses a graph for the detailed information and the structures known from BGP for the direction information. This implementation is used for emulator scenarios and shows the feasibility of a real routing service with a routing protocol.

Further implementations outside of the scope of this book are available as well. For example, a routing service implementation for the hierarchical routing management [Osd12] is available on [FoG].

Although my thesis assumes that only a single routing service is used, FoGSiEm supports multiple routing services in parallel. This differs from a single routing service entity in two aspects:

- A transfer service entity reports its functional blocks not to a single routing service entity but to all entities registered at it.
- A route is calculated by one entity out of the set of registered routing service entities. The entity is determined by iterating over all entities and using the first positive response.

Multiple routing services are especially useful to implement interoperability solutions because the routing service entity responsible for the interoperability can map the names of the external world (e.g. DNS names of IP) to names used by FoG (e.g. binding name of interoperability gateways). Thus, interoperability solutions can be implemented without modifying the original routing service.

4.4.1. Simulated routing service

The simulated routing service mimics a hierarchical routing service within one FoG layer without a routing service protocol. It uses method calls within a single memory address space to distribute its information between entities. In a distributed environment, the RMI framework Apache River is used to interconnect the objects.

The number of levels for the hierarchy depends on the setup of a scenario. The following three levels represent a typical setup:

- Node level: A transfer service entity reports to a routing service entity that is responsible for this entity only.
- Area level: Multiple routing service entities responsible for nodes report to a higher level entity. This level can combine the knowledge from nodes of multiple autonomous systems or from all nodes of a single autonomous system.
- Global level: The highest level combines the abstracted knowledge of all routing service entities responsible for areas.

These levels cluster the inter-network graph of a single FoG layer. Information about the structure of lower layers (e.g. subnetwork graphs) is not included.

Each entity that is not the highest level entity announces itself to the entity in the next higher level. At the lowest level, the transfer service entities are announcing themselves to their assigned routing service entities. Thus, each routing service entity knows the lower level entities reporting to it. Each entity reports at least the gates leading to a lower level entity that does not report to it. The transfer service has to report at least its DirectDownGates. The report includes not only the gate but also the source forwarding node and the destination forwarding node of the gate. As discussed in the architecture description of the routing service, each entity is free to report more gates and forwarding nodes to the higher level. Since the simulated routing service uses labels and no addresses for naming the forwarding nodes, the forwarding nodes labeled with higher layer names must be reported to the higher level as well. An implementation with hierarchical addresses would be able to derive the location of such a forwarding node by its address and not by knowing it explicitly.

Figure 4.10 shows an entity $R_{1,n}$ of an arbitrary level n. It contains the knowledge about lower level entities $R_{i,n-1}$, which are depicted as small circles. Forwarding nodes are drawn as dots and gates as edges. The functional blocks drawn with dashed lines leave the areas of the entities and must be reported to the next higher level. The forwarding node FN_B represents a higher layer binding and must be announced as well.

Figure 4.11 shows the views of the entities of the lower level $R_{2,n-1}$ and $R_{3,n-1}$. $R_{4,n-1}$ is not shown because it is not important for the following example. For the example, n equals 1 and the levels n and n-1 correspond to the detailed and abstract information bases, respectively. The entities of level n-2 are the transfer service entities. At first, an example with best-effort requirements is presented. Afterwards, the handling of requirement is described.

A route from forwarding node FN_S to FN_B is calculated as follows. First, the transfer service entity hosting FN_S sends a route request to its routing service entity $R_{2,n-1}$. The transfer service requests a route from FN_S to the binding with name "tu-ilmenau.de" without QoS requirements. $R_{2,n-1}$ translates "tu-ilmenau.de" to the label of FN_B via its name mapping service. $R_{2,n-1}$ does not know FN_B and, thus, forwards the route request to its higher level entity $R_{1,n}$. $R_{1,n}$ knows FN_B, due to the report from $R_{3,n-1}$. Since $R_{1,n}$ do not know a complete route, it returns a partial route. The route contains the label of the "outgoing" forwarding node FN_O for $R_{2,n-1}$ and the gate number of the "outgoing" gate g_3. Since the destination is not reached by the route, the original destination label is added to the route. Thus, $R_{2,n-1}$ receives the route: [label(FN_O), [g_3], label(FN_B)]. If $R_{1,n}$ do not decide about the "outgoing" gate, it returns the alternative route: [label(FN_I), label(FN_B)]. FN_I is known by $R_{2,n-1}$ since it had reported it to $R_{1,n}$. However, for best-effort routes, the implementation uses the "greedy" version and decides about gates as early as possible. In this case, $R_{2,n-1}$ can now calculate a route to FN_O, since it is located

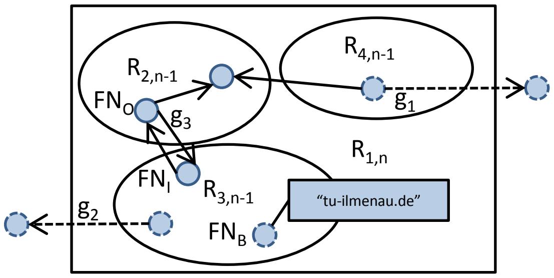

Figure 4.10.: Entity $R_{1,n}$ of simulated routing service on level n with knowledge from the lower level n-1 entities. Dotted elements have to be reported to the higher level n+1.

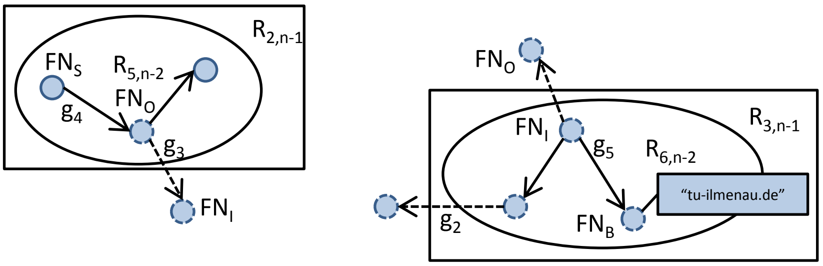

Figure 4.11.: Entity $R_{2,n-1}$ and $R_{3,n-1}$ of simulated routing service. Detailed view related to Figure 4.10.

in its area. If $R_{2,n-1}$ does not know an explicit route it can request missing gates from the transfer service. Assuming enough knowledge, the transfer service receives a partial route that leads the packet to the area of the next routing service entity $R_{3,n-1}$. FN_I detects the end of the explicit route segment and calls the routing service once again. This time, $R_{3,n-1}$ has to calculate a route from FN_I to FN_B. It knows FN_B and do not has to ask $R_{1,n}$.

The same route request with non-functional requirements would lead to a different result. If, for example, the transfer service requests a route supporting 10 Mbit/s with a maximum of 25 ms delay, the routing service has to decide if the available functional blocks provide enough capabilities. In the example, only best-effort gates are available. Thus, the routing service would react by splitting the requirements and by requesting new gates. $R_{1,n}$ might return the route: [label(FN_I)+requ(10 Mbit/s, 10 ms), label(FN_D)+requ(10 Mbit/s, 15 ms)]. $R_{2,n-1}$ is not able to route to FN_I with the given sub-requirements because no suitable gates are available. Consequently, $R_{2,n-1}$ requests one new gate per best-effort gate from the transfer service. The route is constructed in parallel to the existing best-effort gates.

If multiple best-effort routes are available, the implementation uses the shortest one. Obviously, that strategy is not optimal for routes with QoS requirements because it does not take advantage of the capacity information known by the routing service.

4.4.2. BGP for FoG

The FoG-BGP routing service implementation is based on the BGP implementation of the SSFnet [SSF] implemented in the context of the PhD thesis of Premore [Pre03]. It does not support dependencies between gates and provides only best-effort routes. However, the key contribution of this routing service implementation is twofold. First, it proves that a full-featured routing protocol can be used in a FoG layer. It shows in particular the boot-strapping of such a service. Second, it illustrates that FoG routes can be calculated by traditional routing services.

The adaptation of the original implementation showed the flexibility of the FoG layer architecture. Since the routing service can choose its own addresses, the FoG-BGP routing service uses IPv4 addresses without affecting other parts of the implementation. Forwarding nodes are named with IPv4 addresses that are taken from a manual configured IP address range. The range is given as IP prefix, and BGP announces the same prefix to its peers.

The bootstrapping relies on the reports from the transfer service and bindings of the FoG-BGP entities. An example is depicted in Figure 4.12. If the transfer service announces a gate g, which starts at FN_{LA} and leaves the area of the BGP entity (meaning that the destination forwarding node FN_{LB} has an address with another prefix), the FoG-BGP entity starts automatically to set up a

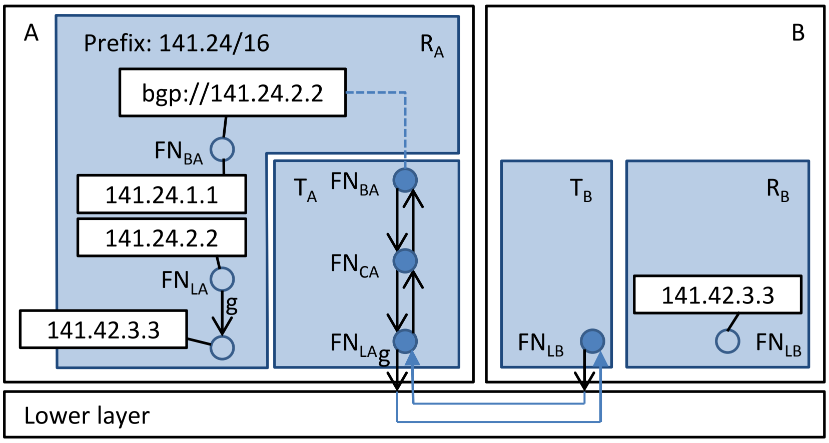

Figure 4.12.: FoG with BGP routing service. Node A is shown in detail.

peering with a potential neighbor entity. The FoG-BGP entity uses the `connect` function of the FoG interface in order to initiate a loss-free connection with in-order packet delivery. Since the peer itself is not known, it specifies a destination name, which can be mapped to the gate to the potential peer (e.g. "bgp://mynewneighbor" or "bgp://141.24.3.3"). The transfer service will request a route from the very same routing service entity by passing along the destination name specified by the entity before. The FoG-BGP entity can now return a route such as [addr(FN_{LA}), [g], "bgp://141.24.3.3"] for its own connection. The access protocol message signaling the connection setup leaves the node via gate g and triggers a routing service request on the neighbor node due to the last destination segment of the route. The FoG-BGP entity B on the node "behind" gate g calculates a route to its binding with the name "bgp://141.42.3.3". Each FoG-BGP entity establishes such bindings for each "border" forwarding node at startup. The forwarding node with the address 141.24.3.3 does not have to be the forwarding node representing the binding with a similar name. For example, the binding "bgp://141.24.2.2" on node A is represented by forwarding node FN_{BA}, which has the address 141.24.1.1.

If a partial route ends on a node with a FoG-BGP entity, the entity searches the longest prefix from its FIB matching the IP address in the destination segment. An entry in the FIB contains the next forwarding node and the "outgoing" gate number as direction information. A result may look like the following: [addr(FN_{next}), [$gate_{outgoing}$], addr(destination)]. This mapping implements step 2 of the abstract algorithm described on page 85. The first destination segment specifies the forwarding node Z that is "closer" to the destination.

The detailed information level of the routing service is implemented like the detailed level of the simulated routing service. For example, a route from FN_{LA} to FN_{BA} through the local transfer service entity is calculated based on a graph of functional blocks not related to BGP.

4.4.3. Mapping from requirements to gates

The requirements mapper subcomponent of the routing service determines the functions required in order to satisfy a set of requirements. Based on the dependencies of these functions, it composes a construction plan for interconnecting the functions. This plan is used by the routing service to construct a chain that contains functional blocks implementing the functions proposed by the plan. Depending on the available blocks and their dependencies, chains can either be constructed by reusing existing blocks or by creating new ones.

The following description uses a "TCP-like function" as example. It implements loss- and error-free and in-order transmissions. This set of requirements will be named "Transport" requirement later on.

4.4.3.1. Construction plan: From requirements to functions

The key idea of the algorithm is to interpret a word constructed by a context-free grammar as a plan for a chain. Each *terminal* of the word represents a function required for the chain. The order of the terminals in the word represents the order of the functions in the chain. Thus, the production rules (or short: *"production"*) for the grammar define the dependencies between the functions. The basis is a context-free grammar, since it prevents "interleaving" dependencies (cp. Section 3.2.2.2).

The grammar was extended by names for productions and the non-functional impact of a production. Such an extended production contains the following elements:

- Production rule name: The name indicates the requirement fulfilled by the production. Named productions can only be used once per part of a word as explained in the example later on.
- Variable: Non-terminal symbol which is replaced by this production.
- String of terminals and variables: This string replaces the variable.
- One or more names of non-functional requirements that indicate the impact of a production.

The following four productions pick off the example with the TCP-like functions and illustrate a typical usage pattern:

1. Transport: S $\rightarrow$ NSO
2. N $\rightarrow$ TCP_sender_gate
3. O $\rightarrow$ TCP_recv_gate
4. S $\rightarrow$ *

Production 1 is a named production that defines how to satisfy the "Transport" requirement. Productions 2 to 4 map variables to terminals. While Productions 2 to 3 define the mapping to terminals indicating functions for gates, Production 4 marks a separation between parts of a chain. The separation can be filled with functions relaying packets between two end systems. Production 1 is the important production rule that defines the order of the functions. N has to be executed before O. Either both or none of them has to be included in a word, since the productions do not allow the construction of a word using, e.g., N only. In between N and O, a pipe transporting packets (S) is required. The variable S represents such a pipe. It is also used as start variable, from which the following words can be constructed with a normal grammar:

a) "*"
b) "TCP_sender_gate * TCP_recv_gate"
c) "TCP_sender_gate TCP_sender_gate * TCP_recv_gate TCP_recv_gate"
d) and so on

Words constructed by using Production 1 multiple times are not appropriate in the context of function chains. Such rules represent a service constituted by several functions satisfying a requirement. The service represented by Production 1 satisfies the requirement "Transport". If a service fulfills a requirement, it is not useful to include the service in a chain a second time.[6] Thus, the modified grammar used by the algorithm does not allow the repeated usage of named rules. Only the word (a) and (b) are valid words of the modified grammar.

For grammars containing productions that introduce functions on relay systems the usage of services has to be formulated more precisely. If the following production is added to the grammar, the usage of services is restricted to a "part" of a word.

5. Relay: S $\rightarrow$ SS

Each S from the right hand side represents an independent part, which is allowed to use a rule once. This enables the sequence (e) "NSONSO". For each S inside (e), Production 1 is forbidden since it was already used.

[6] At least if it covers the same scope as in the example word (c).

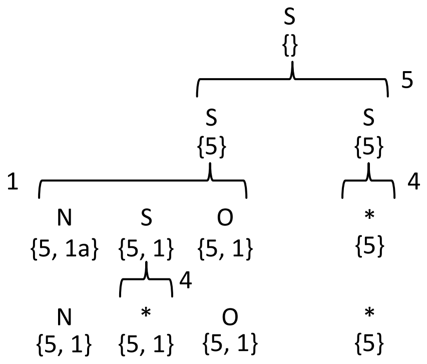

Figure 4.13.: Word creation with modified grammar. The list of productions used in the upper part of the tree is written below a variable/terminal.

Figure 4.13 shows the steps for creating one word. For simplicity reasons, it omits Production 2 and 3. The set below each variable/terminal of the word contains the productions used in the upper part of the tree. On the one hand, this set contains the named productions not permitted any more. On the other hand, it indicates the requirements that are already fulfilled for this part of the chain. The resulting chain fulfills the "Transport" requirement for the first three functions. However, the requirement is not fulfilled for the last "*". Thus, a service has to specify if it fulfills a requirement once and for all or if it provides a "partial" solution. A consequence is that not all words of a grammar are useful plans for chains.

There may be several services fulfilling a requirement. For example, the following second service fulfills the "Transport" requirement as well:

6. Transport: S $\rightarrow$ NASRO | +Delay -Datarate
7. A $\rightarrow$ Adding_forward_error_correction_information_gate
8. R $\rightarrow$ Reconstruct_via_forward_error_correction_information_gate

In order to provide a base for choosing between Production 1 and Production 6, a service can indicate, if it supports (+) or hampers (-) the fulfillment of other requirements. For example, Production 6 supports a better delay by avoiding retransmission and hampers a data rate requirement since it introduces forward error correction overhead.

Obviously, this is more a coarse hint rather than a detailed analysis. It does not specify the relation between the delay improvement and the throughput reduction. Such details are neglected in favor for a simpler algorithm.

The whole grammar used in the implementation is listed in Appendix C.

4.4.3.2. Selecting a construction plan

The construction plans created by the modified context-free grammar described before have to be evaluated in the context of a set of requirements R. A suitable plan has to fulfill all functional requirements. If a requirement is fulfilled only partially as shown in the previous example, the plan is not suitable. All plans suitable for a requirements set form the set of suitable construction plans P. The requirement mapper component computes a ranking for the elements of P that indicates their fitness. The better the fitness the smaller the ranking is. The plan with the smallest ranking is selected.

The following equation defines how the ranking is calculated:

$$\forall w \in P : \text{ranking}(w, R) = \frac{|\text{fulfilled}(w)|}{|R|} \tag{4.3}$$

The function fulfilled(w) returns the set of requirements fulfilled by the plan w. Thus, the ranking equals 1 in the best case and increases, if the plan fulfills more requirements than necessary. For example, a plan fulfilling the "Transport" and "Privacy" requirement would include more functions than necessary for fulfilling only "Transport".

4.4.3.3. Creating a chain: From a construction plan to a chain

After selecting the best plan, the routing starts to construct a chain. It iterates the terminals in the word and selects or creates functional blocks providing the function indicated by the terminal symbol. Additionally, the construction process has to adapt the plan to the conditions in the transfer plane. It is free to add forwarding nodes in between two functions defined by the plan. Furthermore, it must replace the "*" terminal symbol with functional blocks implementing the relaying of packets between systems. This involves three steps:

1. The construction plan is split in the parts for the two end systems and in the part for relay systems. The "*" terminal symbol marks the transitions. For example, the plan (b) is split in the two parts "TCP_sender_gate" and "TCP_recv_gate" for the end systems. The plan does not require any specific functions on relay systems.

2. Each system processes its part and creates or reuses gates. For example, the routing service at the sender checks if it already knows a "TCP_sender_gate" that depends on a "TCP_recv_gate" at the receiver. If it knows one and if it is allowed to reuse the gate, the gate will be

reused for the new chain by including its gate number in the route. If the routing service does not know such a gate, it requests a new one from the transfer service.

3. The routing service searches for gates, which can be reused to close the gap between the "TCP_recv_gate" and the "TCP_sender_gate". If no non-functional requirements are given, it can reuse relaying gates with best-effort capabilities for the chain. If non-functional requirements are given, it searches for relaying gates that fulfill them. Otherwise, suitable gates are again requested from the transfer service.

The access protocol is used to signal the required gates between the end systems. The implementation uses the incremental routing process to create missing gates on relay systems during the relaying process of the access protocol messages. Since the algorithm for the requirement mapping is a deterministic one, only the input requirements are signaled. It assumes that all nodes use the same grammar. If this is not the case, the access protocol has to transport the detailed plan.

4.4.3.4. Discussion

The proposed algorithm is a basic one that proves the feasibility of the requirements mapper component. However, it is not the only possible implementation of the component. Other algorithms may be more flexible in satisfying requirements, use fewer gates, or be more efficient. For example, the mapping from requirements to functions can directly take the available gates into account. Adaptations of the solutions presented in Section 2.5, such as SONATE, may provide a better trade-off between runtime and design-time complexity. Since this topic is not the main focus of this thesis, the algorithm demonstrates a basic solution. Its extension or the integration of other solutions is subject to future work.

The algorithm does not limit the granularity of functions. However, its memory complexity depends on the number of functions. Thus, it prefers a small number of rather large functions. I used functions comparable to the functions of an OSI layer. The TCP-like function is used as a single block and is not split in multiple smaller once (e.g. a block for calculating the check sum and a block for checking sequence numbers).

4.5. Authentication services

The *authentication service* is used mainly for signing signaling messages. A receiver of a signaling message checks the list of signatures in order to verify its authenticity. FoGSiEm provides two authentication service implementations:

- Simple authentication service: The simple authentication service is for simulation scenarios, which do not require real authentication. A "signature" just contains the name of the signing entity and has no dependency to the signed message (in particular there is no (encrypted) checksum). This authentication service does not introduce computational overhead and is useful for fast simulations.
- Public-private-key authentication service: The public-private-key authentication service uses asymmetric keys for creating and checking signatures. The private key is used to encrypt a checksum of the payload of the packet that should be signed. At the receiver, the public key is used to decrypt the checksum. If this checksum equals a checksum calculated locally, the verification of the authentication is considered to be successful. This implementation provides an authentication service for simulation scenarios including malicious nodes or emulation scenarios. The distribution of keys is not part of this implementation and requires manual setup.

Authorization can be done based on the verified sender entity. However, FoGSiEm's authorization policy is simple:

- All entities are allowed to create new gates and forwarding nodes independent from the requested non-functional capabilities.
- The creator of a gate or forwarding node (the entity that requested the creation) is allowed to change or delete it.
- The function provider has "administrator rights" and is allowed to delete any function provided by it. The function provider is in particular allowed to delete gates that do not receive any refresh signaling messages from its creator.

4.6. Emulator

FoGSiEm can operate as emulator and span a FoG network without IP. A lower layer implementation that accesses a link layer directly enables an *emulation* of a FoG network without IP being present. I use the term *emulator* to refer to usage scenarios of the FoGSiEm software that replace IP with FoG.

The important aspect of the emulator implementation is that the core part (plug-in *fog*) is not aware of the emulator aspects, which are added transparently. Only the events have to be executed close to real-time. Thus, the term emulator might be misleading regarding to two aspects. First, it depends on the set of used plug-ins and not on different mechanism in the FoG core. Second, the concept does not fit well in the recursive reference model. This is comparable to the concept of an overlay, which is discussed on page 110. FoG operates always over lower layers regardless if such lower layers are implemented with IP or Ethernet. Even the distributed simulation would fulfill the criteria of an emulation.

Due to the recursive model, FoG requires lower layers supporting the FoG layer interface (cp. Section 3.3.1). Since such implementations have not been available, one has been implemented in cooperation with Thomas Volkert. It is based on Ethernet and adds some mechanisms on top to support the FoG layer interface. The implementation uses the library *LibPCap* [Libb] to receive packets from a network interface and the library *LibNet* [Liba] to send packets. Since both libraries are available as C code only, they are connected to FoGSiEm via the Java Native Interface (JNI). The implementation assumes the network interface to represent an Ethernet-like broadcast domain with MAC addresses and a frame format including an indication about the higher layer type such as the "EtherType" field of Ethernet. The implementation uses the EtherType value of 0x6060. The value is not officially reserved for FoG. However, it is not used in current networks and does not cause conflicts. The payload of the frame contains a binary representation of a Java object that includes the FoG packet fields specified in Section 4.3.2.1 as attributes. It is encoded with the standard Java object serialization [Ora]. On the one hand, this encoding causes larger packets compared to an optimized serialization since it contains, e.g., Java class names. On the other hand, it reuses the stable standard de-/serialization routines from Java in order to shorten the implementation time. One consequence is that the packet sizes of FoGSiEm do not match the packet sizes achievable with a productive implementation.

If the serialization of a FoG packet is too large for one Ethernet frame, it is split in multiple fragments. Each fragment has an additional header, which specifies if the fragment is the first, intermediate or last fragment of a packet. All fragments of one packet are sent directly after another. A sender is not allowed to interleave fragments of multiple packets for one receiver. If the last fragment of a packet is received, the packet is reconstructed and the FoG Java packet object deserialized. If fragments are missing (first fragment received but no last fragment) or if the object creation failed (intermediate fragments missing or bit errors), the packet/fragments are dropped silently. Successfully reconstructed packets might contain bit errors in the FoG header and/or in the payload. Thus, the implementation provides connections between two FoG nodes with a loss and error rate higher than zero. This is reflected by the gates representing those connections. If FoG requires loss- and error-free lower layer connections, additional gates that handle these issues, e.g., with retransmissions and checksums, have to be created.

The neighbor discovery aspects of the FoG layer interface are implemented with an exchange of "hello" messages. Each Ethernet entity with a FoG entity attached to it, broadcasts "hello" messages periodically. Each receiver of such a message adds the sender to its neighbor list. If no “hello” message is received during a defined time period, the entry is removed from the list. Their behavior is comparable to the behavior of LDP in MPLS networks. The “hello” messages and their format are specific for the implementation of the Ethernet-based lower layer and are not related to FoG.

5. Performance studies

An architecture represents a set of implementations and summarizes their invariances. If two architectures should be compared, both sets of implementations have to be compared. Chapter 3 approached the comparison of FoG with other architectures with qualitative arguments. Quantitative arguments are more problematic, since they require performance measurements of the best implementation of each architecture. Since the best implementation is, in general, unknown, a representative subset of implementations is used. For new architectures like FoG, there is no such set but a single implementation. The lack is especially severe, since it is a rapid prototype of a three-year research project, which is known to have performance drawbacks. The performance of such an implementation cannot stand against, e.g., mature and highly optimized IP implementations. In his keynote talk at the EuroView 2012, Hisashi Kobayashi [Kob12] stated such a lack of quantitative arguments for this area of research in general.

Even a comparison of two individual implementations – each from a different architecture – has two main limitations. First, the set of setups, meaning network graphs and implementation configurations, is too large to analyze all of them. A grouping of setups requires grouping criteria. Due to uncertain forecasts for Internet use cases (cp. Section 3.1.4), such criteria are highly unreliable. I approached the problem by executing my simulations for a set of network graphs representing different views on the current Internet. In order to reduce the configuration space, I used only simple policies. Second, one architecture may not support a setup that is supported by another. As shown in Chapter 3, FoG supports setups not supported by IP. The advantage of FoG is the support for these setups and not a better performance. The evaluation whether it is beneficial for an architecture to support a setup faces the same problems as the grouping criteria mentioned above. Basically, I assume that it is beneficial to support functions within networks. The motivation for this assumption is given in Chapter 1 and Section 3.1.

The performance studies presented in this chapter support the qualitative arguments from Chapter 3 with quantitative results. The studies approach the comparison by measuring abstract performance values revealing aspects of architectures. Therefore, they focus on aspects of implementations that are related to the architecture such as the number of states. They do not measure, for example, the bytes of memory required for such states, since this is highly implementation dependent. If FoG would be implemented with the programming language C, it might be possible to store a state with less bytes than required by the Java implementation. In contrast, the abstract performance measurements hold for a large set of implementations. The comparison goes a bit further and assumes that the measurements hold for the complete set of an architecture. Thus, there is a thin line between too abstract measurements that are not relevant any longer and measurements not representing all implementations.

The following studies are exploring the possible trade-off between flexibility and scalability that are enabled by FoG. Therefore, FoG is configured with different policies that influence the placement of states in networks. The placement influences mainly the transfer and the routing service. These influences are studied separately for each service. If a configuration achieves similar features as an IP setup, the results for both architectures are compared. Moreover, the robustness of explicit routes is studied. It profits from the various repair algorithms, which are enabled by the flexibility of the transfer service. The studies can be summarized as follows:

- **Scalability of transfer service:** FoG can place states on end systems in order to provide functions in a scalable way. The study in Section 5.2 compares this to an IP-based solution, which places them on relay systems. It shows that the flexible placement of mapping states for reusable functions leads to a more homogenous distribution of states in a FoG network. In particular, it shows the dramatic reduction of states stored in core relay systems. Additionally, it shows that the length of pure explicit routes between end systems causes acceptable overhead even for large inter-networks.
  - Influencing factors:
    - Architecture (Two-Tier reference architecture combining IntServ & DiffServ vs. FoG)
    - Network graph
  - Metrics:
    - Number of states (IP)
    - Sum of gate numbers in routes (FoG)
    - Route lengths (FoG)
- **Scalability of routing service:** The routing service policies of subnetworks influence the number of routing service requests required by the incremental routing process and the size of the routing service information bases. The study in Section 5.3 shows that a hop-by-hop IP-like configuration and a source routing configuration have the biggest routing information base. For the algorithms of FoGSiEm, a configuration

in between these both extremes is optimal. The study shows that the optimal trade-off depends on the algorithms used by implementations and that FoG's flexibility allows a network to adjust its configuration to this trade-off.

  - Influencing factor:
    - Routing service policy (Set of routing service entities that is informed by another routing service entity); Special cases:
      - IP-like hop-by-hop forwarding
      - Source routing
  - Metrics:
    - Number of routing service requests
    - Size of routing service graphs

- **Robustness of connections:** The explicit routes (or route parts) are a disadvantage of FoG, because they require an early binding of destinations names to routes, which is vulnerable to failures. At the same time, however, these routes are an advantage of FoG since functions can be directly selected. This gives a source more control over the path of a packet, which can be used to define detours in case of failures. The study in Section 5.4 shows that explicit routes which are affected by failures of functional blocks representing relaying functions can be repaired efficiently. FoG's flexibility enables various repair algorithms. Thus, the static nature of explicit routes does not necessarily cause a less robust network.
  - Influencing factors:
    - Repair algorithm
    - Network graph
  - Metrics:
    - Fraction of successfully repaired routes
    - Length of original routes
    - Length of repaired routes
    - Number of hops for signaling messages

The studies had been executed with FoGSiEm in simulation mode. The simulation setups are common for all studies and are described in Section 5.1.

The emulation performance of FoGSiEm had been measured as well. Since these results are specific for the implementation and not for the architecture, the results are minor important for my thesis. Nevertheless, the results are shown in Appendix D.

Graph	Number of nodes	Number of edges	Node degree	
			Maximum	Median
GLP	5.000	12.721	262	1
DIMES	25.506	77.419	3.917	2
Ark	25.266	64.900	3.002	2

Table 5.1.: Sizes and node degree values of the graphs

5.1. Simulation setup

The networks for the simulations are constructed according to two types of real Internet graphs and one randomly generated graph with comparable characteristics. These graphs represent the structure of a large-scale inter-network on subnetwork level by focusing on the interconnection of subnetworks and not on the internal structure of the subnetworks. A vertex of such a graph represents a subnetwork and an edge represents a bidirectional link between two subnetworks. Table 5.1 summarizes the size of the following graphs:

- *GLP graph*: The *Generalized Linear Preference* (GLP) algorithm implemented in BRITE [MLMB01] has been used to generate 10 random inter-network graphs. The GLP algorithm ensures that such graphs have similar statistical characteristics as the real Internet graph. The parameters of the GLP algorithm have been adjusted to the optimal values derived in [HFU+08]. Since their graph characteristics and initial simulation results do not differ significantly, only one GLP graph was used for all studies. It has fewer nodes than the other graphs in order to enable more extensive simulation scenarios.

- *DIMES graph*: The DIMES project measures the graph of the Internet. It uses results from `ping` and `traceroute` commands executed by multiple agents hosted by volunteers. The simulations use the graph of the Internet measured in February 2012. Since the original graph is partitioned, only the biggest partition is used. [SS05]

- *Ark graph*: The Caida project measures the graph of the Internet as well. Their Ark tool (formally known as Skitter), which runs on multiple locations in the Internet, traces the routing advertisements and transforms them into an inter-network graph. The simulations use the graph derived at the $2^{nd}/3^{rd}$ of February 2012. Indirect edges of this graph (links between two subnetworks with unknown intermediate subnetworks) are treated as direct ones. [HHA+]

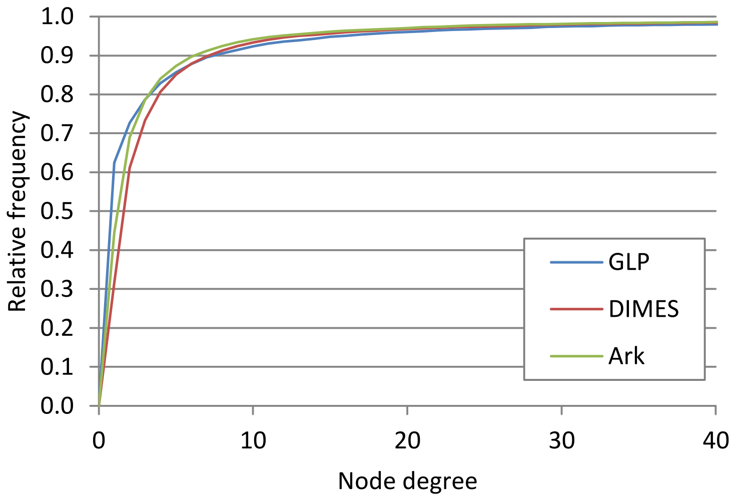

Figure 5.1.: Empirical distribution functions of node degrees for the three Internet-like graphs

These graphs differ significantly. In particular, the measured graphs are not identical due to the difficulty of measuring the graph of the Internet. They approximate the real Internet graph. However, I assume that all three graphs have an Internet topology and, thus, have similar characteristics as the current and future Internet graphs.

The distribution of the node degrees[1], which is shown in Figure 5.1, is especially important for the studies. It shows that only a few nodes have a very high node degree. These nodes have many direct neighbors and, thus, play a major role in the inter-network. Since the axis is cut at a node degree of 40, the maximum node degrees are given in Table 5.1. More details on the Internet topology in general are given in the references stated above.

5.1.1. Representing subnetworks

Due to the focus of FoG on the inter-network scope, simulation scenarios focus on inter-networks as well. They model the graph structure of inter-networks and assume homogenous internal structures of subnetworks. Moreover, they do not assume any specific subnetwork layer implementation (a subnetwork might, e.g., use IP with DiffServ). Thus, the basic idea is to represent a

[1] The node degree is the number of directly adjacent neighbor nodes.

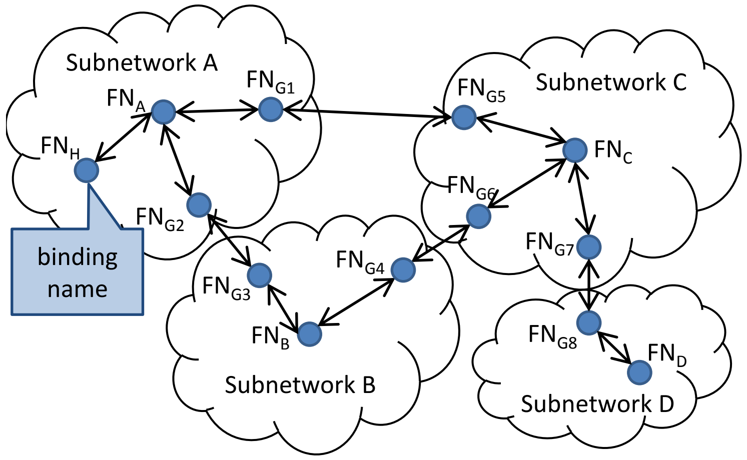

Figure 5.2.: Example for a FoG setup that models an inter-network. Bidirectional edges represent gates in both directions.

subnetwork with a node. Network interfaces of the node represent gateways of the subnetwork. The detailed setup for FoG is described in the following.

Figure 5.2 shows a small inter-network example with four subnetworks and how they are represented with FoG. A subnetwork is modeled with a single central forwarding node representing the subnetwork as a whole (for example, FN_A represents subnetwork A in Figure 5.2). Gateways are modeled with additional forwarding nodes $\text{FN}_{\text{G}i}$. The central forwarding node connects all gateway forwarding nodes in a star topology, which represents the connectivity within the subnetwork. The model assumes that

- a subnetwork is not partitioned (meaning that all gateways can communicate with each other via the subnetwork) and that
- one gateway is connected to exactly one gateway of another subnetwork (meaning that the lower layers between the subnetworks provide only point-to-point connectivity).

Bindings (cp. Section 3.3.1) within a subnetwork are represented by forwarding nodes as well (e.g. FN_H). These forwarding nodes and their higher layer names are reported to the routing service. They are treated as if they belong to end systems located within subnetworks. Since the number of these forwarding nodes depends on the number of bindings, the state analysis ignores them.

The above representation of a subnetwork is equivalent to the general setup of a FoG node, which is described in Section 4.3. Thus, the simulation represents each subnetwork with one FoG node that contains the forwarding nodes and gates as described above. Links between FoG nodes represent the inter-network edges between gateways of subnetworks. There is exactly one lower layer instance from plug-in *fog.bus* per subnetwork link, which emulates a point-to-point connection between two gateways. A FoG node creates one forwarding node per attached lower layer. This forwarding node corresponds to the forwarding node representing a gateway. The central forwarding node of a FoG node represents a subnetwork. In summary, a scenario contains at least two forwarding nodes, two best-effort DirectDownGates, and four TransparentGates per inter-network edge and one forwarding node per subnetwork.

5.1.2. Network load and applications

Network load is generated by establishing connections between applications on randomly chosen FoG nodes. The chain setup of each connection is tested by sending data and checking the correctness of an echo response from the peer.

The random selection of nodes as source and destination was chosen, although connections in the real Internet are not set up between random nodes. However, there seems to be no statistical data about how the node degree of a subnetwork correlates with the number of connection of its end systems.

The application requirements for connections depend on the scenario and are described for each study separately. The simulations assume that all nodes support all functions.

5.2. Scalability of transfer service

The following study investigates if the flexible state placement of FoG can be used to enhance the scalability. FoG should place states in a more homogenous way than other solutions that overload core relay system with states. In particular, function users should take over mapping states from function providers. The drawback of this distribution of states is that the function user has to transmit its decision about the usage of functional blocks along with packets. Thus, the study analyses the overhead introduced by this transmission for large inter-networks.

In this study, applications establish connections that should be treated with a higher priority than other connections[2]. The networks described in Section 5.1 can provide these connections by creating functional blocks that represent a prioritized packet transmission between nodes/subnetworks. These blocks can handle the aggregated traffic of multiple connections and, thus, are reusable. The number of states required for reusable functional blocks is dominated by

[2]Such low priority connections are not established, since the real non-functional characteristics of the packet delivery are not important for this study. It is just concerned about the states required to support a reusable function.

mapping states. Thus, the number of mapping states required on end and relay systems is the metric used to evaluate the scalability. The function states (for example buffers and scheduler parameters) are not measured, since they are equivalent for both architectures.

The following two architectures are simulated:

- **Reference:** The reference architecture is a typical two-tier architecture combining IntServ and DiffServ, which is described in Section 3.1.2. The gateways of DiffServ-based subnetworks participate in an IntServ-based inter-network. These gateways have to store classification states in order to map packets to connections and connections to service classes used in the DiffServ-based subnetwork. In the simulation, the numbers of IntServ states per relay system are counted. They correspond to the states required on the gateways of a subnetwork. The states on end systems (e.g. socket states) are ignored, since they are of constant size.
- **FoG:** FoG is able to place mapping states on systems of function users instead of relay systems of function providers. The simulation assumes that FoG is allowed to place them on end systems[3]. Thus, end systems store explicit routes that contain the selection of the prioritized transmission functions for each hop. The simulation sums the route lengths of all routes an end system has to store. Since FoG has to transmit the states from function users to function providers via its routes, the lengths of the individual routes at the source are measured as well.

The reference scenario was simulated with FoGSiEm. Therefore, FoG was configured to create per-connection states on all nodes along a path in order to emulate the behavior of the reference architecture.

The priority requirement is just an example requirement that prompts the usage of reusable functions. The results of this simulations hold for all other requirements leading to the usage of other reusable functions as well.

In the following, the question whether the overhead induced by transmitting routes of variable length in packets is reasonable in large-scale inter-networks is answered. Afterwards, the state distribution is analyzed.

5.2.1. Packet overhead due to route length

Figure 5.3 shows the distribution of FoG route lengths of explicit routes in gate numbers for the graphs. The sample contains 100,000 connections per graph. The minimum route length is four gates, since two gate numbers are required to leave the source system and two to reach the forwarding node representing

[3]Since a FoG node represents a subnetwork (cp. Section 5.1.1), an end system is equivalent to the subnetwork a “real” end system is located in.

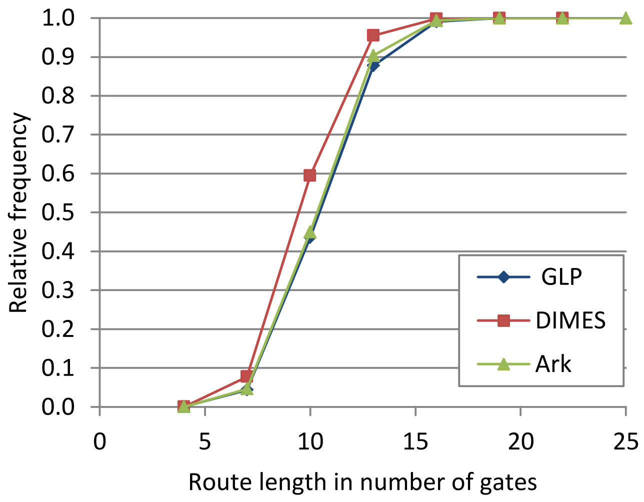

Figure 5.3.: Empirical distribution functions of route length of explicit end-to-end routes for the three Internet-like graphs with FoG

Average route length in	**GLP**	**DIMES**	**Ark**
number of gates	11.95	11.12	11.82
number of hops	3.65	3.37	3.61

Table 5.2.: Average FoG route length for the three Internet-like graphs

a binding on the destination system. Since relay systems/subnetworks require three gate numbers for encoding the route through them, the length values increase in steps of three. Table 5.2 lists the average values in number of gates and number of hops.

The routes are relatively short, because Internet-like graphs have a short graph diameter between 3 and 4 subnetworks [MKF$^+$06]. The diameter is independent of the overall number of subnetworks in an inter-network.

In order to compare the overhead of FoG routes with, e.g., IP addresses, an encoding for the elements of the FoG packet header is required. Since the architecture does not define an encoding, it is a characteristic of an implementation and, thus, out of scope for an architectural comparison. However, an encoding would have to be standardized in order to enable different vendors to develop interoperable FoG equipment. Therefore, the encoding is important, and a comparison would be of interest. In the following, I assume that a gate number, the route length field, the route segment length field, and the route

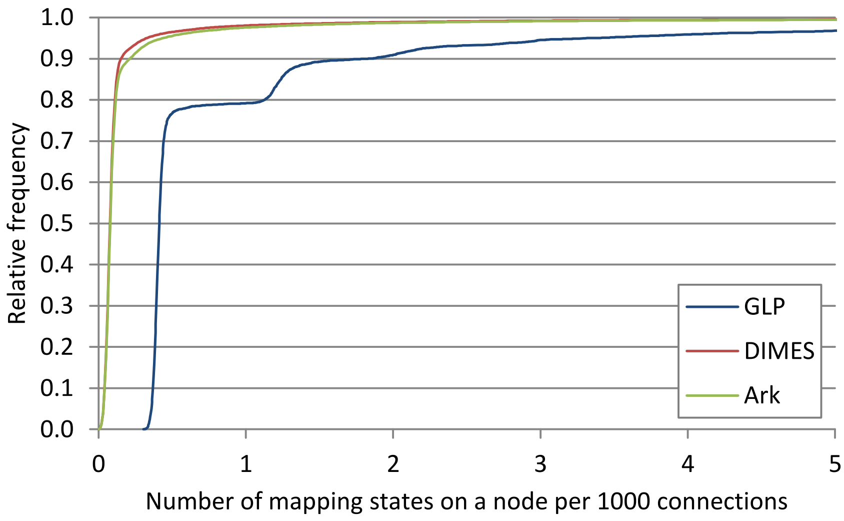

Figure 5.4.: Empirical distribution functions for state information in reference architecture. Maximum number of mapping states: GLP=112.3, DIMES=399.9, Ark=392.3. A plot with the full x-axis is shown in Appendix G.3.

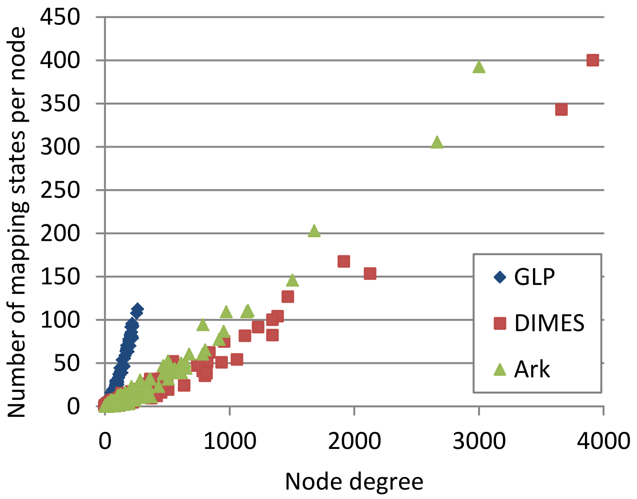

Figure 5.5.: Node degree vs. number of mapping states for reference

segment type field from the FoG header (cp. Section 4.3.2.1) are encoded in one octet each. That enables routes with a maximum of 255 route segments each having a maximum of 255 octets.[4] Based on this encoding, a complete average route with one explicit route segment containing approx. 12 gate numbers (cp. Table 5.2; three gate numbers per transit subnetwork) would require 12 octets plus an overhead of three octets. Explicit routes that are equally long than one IPv6 address (16 octets) would contain 13 gate numbers. According to Figure 5.3, at least 87% of the routes have exactly 13 or less gate numbers and are, thus, equally long or shorter than one IPv6 address. Routes get even shorter during the transmission of a packet. An average route during transmission is only half the gate numbers long. That leaves some space for transit networks inserting more than three gate numbers or for end systems requiring more gate numbers to encode the route through their "stack". Thus, the overhead for large networks seems to be comparable with IPv6 or may be even smaller.

5.2.2. State distribution

Figure 5.4 shows the distribution of the number of mapping states on an IP node (that represents a subnetwork) per 1000 connections for the reference architecture. The sample contains 500,000 connections for the GLP graph and 100,000 connections for each of the other graphs.[5] The overall number of states per graph is only affected by the average route length of a graph. Since the route lengths are similar as shown in the previous section, the overall number of states is similar as well. However, the curves for DIMES and Ark differ from the curve for GLP due to the different number of nodes in the graphs. A similar number of states distributes over more nodes. Nevertheless, the distributions of all graphs have similar characteristics. The number of states is relatively low for nearly 75% of all nodes but increases rapidly for the last 5%. These few nodes have to handle high numbers of connections and, thus, are expected to have a "central" position in the network graph. Figure 5.5 shows that the node degree correlates linearly with the number of states a node has to handle.[6] It confirms that the scalability problem is most severe at transit subnetworks. This result holds for all graphs of the Internet topology because the heavy tail distribution of the node degree is a characteristic of the topology.

[4]These limits are not a problem for explicit routes, since they can easily be spread over multiple segments. The limits are more important for destination segments. Maybe, such segments have to be split in a "destination name segment" and a "destination requirements segment" in order to have enough space to encode names and requirements. Another option would be to reserve the value 255 or one bit of the octets or a bit of the segment type field for a flag indicating that the next segment belongs to the current one.

[5] The smaller number of connections for the large graphs is caused by the long simulation time. Each simulation took approximately one month on an Intel Xeon CPU (X5660; 6 cores) with 2.8 GHz and 24 GB RAM.

[6] Details about the correlation are given in Appendix F.

FoG places these states on end systems. There are no connection-specific states at the relay systems any more. However, end systems have to store these states in form of gate number in the routes for connections. Figure 5.6 shows the distribution of the overall number of gate numbers a node has to store for all its "own" connections per 1000 connections of the simulation. It shows the much more homogenous distribution of states among the nodes.

At first glance, the distribution seems to be a normal distribution. This seems to be an artifact of the random selection of the source and destination nodes. As shown in Appendix E, the distribution, however, is not a normal distribution. It has slightly more nodes with more gate numbers than nodes with less gate numbers than in average. This tail seems to be caused by the distribution of the route lengths, which has a positive skew as well. The differences between the results for the GLP graph and the results for the other two graphs are again caused by the different number of nodes in the graphs. There is no linear correlation between the node degree and the number of gate numbers as shown by Figure 5.7. However, the figure shows that "core" subnetworks of an inter-network, which have a high node degree, can exploit their central position and have to store shorter routes (in average) than nodes with a low node degree at the "periphery" of an inter-network. More details about this relationship are given in Appendix F.

In reasonable implementations, both states are not necessarily comparable. An IntServ gateway would have to store at least the IP address tuple and protocol type number (or flow label in IPv6) per connection. That sums up to 9 octets and 35 octets for IPv4 and v6, respectively. Overhead for data structures that might be required to store lists of connections is omitted. In contrast, a gate number could be encoded with a single octet due to its small context. Since such short gate numbers seems to be rather unrealistic for traversing whole subnetworks, relay systems may use longer encodings or multiple short gate numbers in order to encode a partial route through their subnetwork. Due to the setup of forwarding nodes within a FoG node, the FoGSiEm implementation requires three gate numbers to traverse a relay system. If a relay system optimizes its gate number length according to its CPU register size, 8, 16, or 32 bit per gate number seems to be reasonable. Thus, an end system would have to store 3, 6, or 12 octets, respectively, per relay subnetwork.

If we assume that three gate numbers are comparable to one IntServ state, the values shown in Figure 5.8 are the result. It shows the values from Figure 5.6 divided by three in order to reflect the number of gates per subnetwork and the unmodified values from Figure 5.4 in a single chart. The sum of all IntServ states equals the scaled sum of all gate numbers stored in routes on FoG end systems. The (empirical) maximum of 112.3 states per node in the reference scenario is reduced significantly to 2.9. These states are now stored on the end systems, which have to store more states than in the reference scenario. This

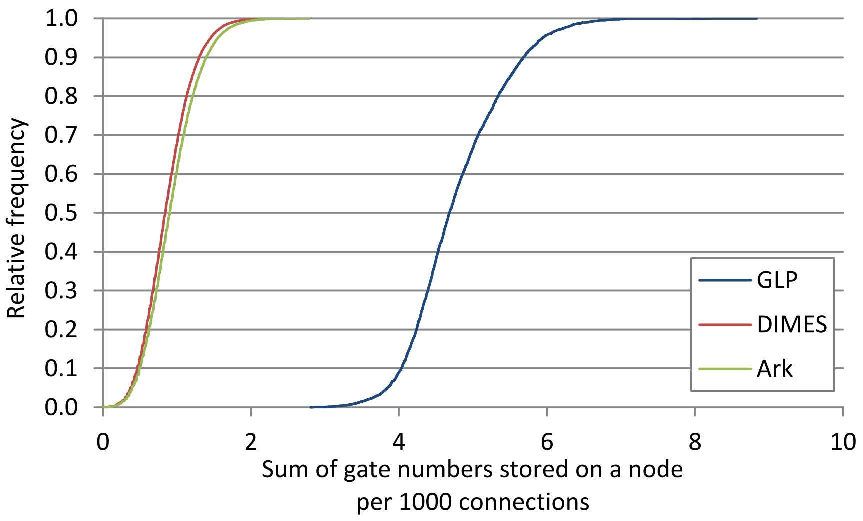

Figure 5.6.: Empirical distribution function for the number of gate numbers stored on a node. The value is calculated by counting all gate numbers in all routes stored on a node.

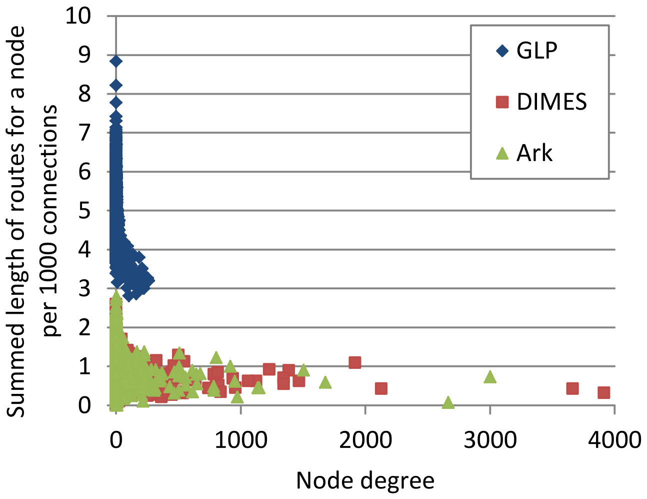

Figure 5.7.: Node degree vs. number of gate numbers on a node per 1000 connections for FoG configuration

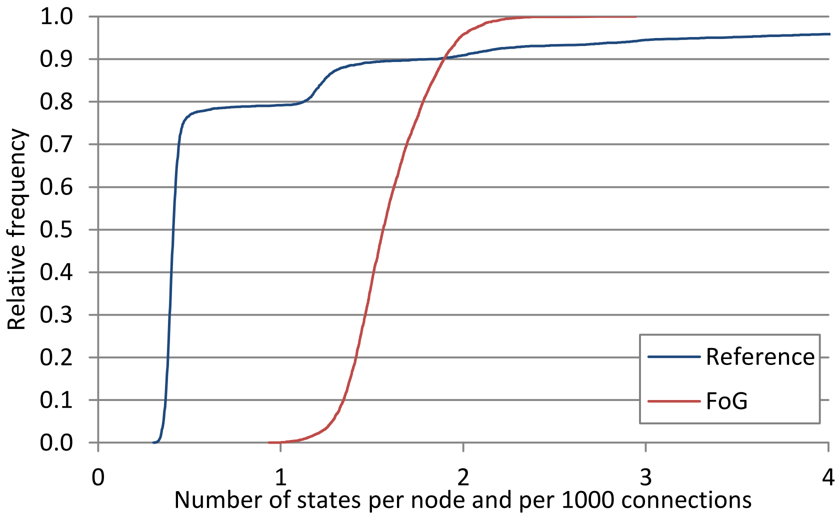

Figure 5.8.: Empirical distribution functions for number of states per node for FoG and reference architecture (for GLP graph). FoG values are taken from Figure 5.6 and divided by three. Reference curve reaches maximum value 1.0 at 112.3. FoG curve reaches maximum value 1.0 at 2.9.

shows the "movement" of states from the 10% highly loaded nodes to the 90% low loaded nodes. Even if the assumed scaling of the values from Figure 5.8 is not correct (meaning: divisor is not equal to three), the types of distribution still differ. The heavy tail of the reference result is cut by the FoG result for all reasonable setups[7].

5.2.3. Discussion

The results show that FoG's flexibility regarding the placement of states has a huge impact on the scalability of a network. The topology of inter-network graphs with a heavy-tailed distribution of the node degree (cp. Section 5.1) causes a central role of a few systems. IntServ overloads these core relay systems with mapping states. As stated in Section 3.1.2, IP-based solutions that preserve flexibility do not seem to be able to avoid this. FoG can avoid overloaded core relay systems by storing mapping states at function users. Thus, FoG achieves a much more homogenous distribution of states.

[7] An average gate number requires less memory than approx. 10 IntServ states.

FoG scales, because the homogenous distribution of states increases the number of entities that can store states. The number of states in a FoG network still depends linearly on the number of connections (multiplied with the fairly constant graph diameter for inter-networks) and does not change to a, e.g., logarithmic dependency. However, the resources available for handling these states increase as well. Subnetworks joining an inter-network do not just increase the number of connections with the connections of their end systems but take over some of the load of storing states. This is comparable to the overall number of TCP states in the Internet, which increases linearly with the number of connection as well. The scalability is ensured by the increasing number of end systems storing these states. FoG is improving its scalability for reusable functions in the same way by distributing the states among more entities.

The study assumes the best case that connections require only reusable functions. If the functions are not reusable, the mapping states are no longer the dominating factor. Since each connection requires its own functional block per hop, the function states get much more important. FoG cannot place these states at different locations since they are bound to the function provider. Thus, the per-connection states stored on relay systems for FoG and for the reference architecture would be similar. A more realistic setup mixes reusable and non-reusable functions. The ratio between both functions influences the results, which are expected to be somewhere in between the two curves shown in Figure 5.8.

How stable are the simulation results?

Besides the importance of the average route length as indicator for the overhead, the metric serves as a reference for the stability of the results from all simulations. The average route lengths of the different simulations do not differ significantly. In particular, the nearly identical average route lengths of the robustness study, which includes only 20,000 connections for the large graphs, indicate that the sample sizes of the simulations are reasonable. If a simulation of a study is influenced by other important factors, the description of this study contains some further comments on the reliability of the statistic.

5.3. Scalability of routing service

The following study investigates the impact of routing service policies on the network performance. The performance is influenced, in particular, by the size of the routing service information base and the number of routing service requests. Both can be balanced with the incremental routing process, which bases on the ability to operate with partial routes. Partial routes are the key tool for a routing service to reduce its routing service information base, since they allow a routing service entity to limit its detailed knowledge about a network to a small area. However, the smaller the areas are the more partial routes have to be computed. The study analyses the trade-off between the sizes of the areas and the number of requests in the context of the algorithm used by the routing service.

The study explores the trade-off by varying the routing service policies of subnetworks. The routing service entity of a subnetwork can announce gates to other routing service entities, in order to delegate the usage decision and, thus, the role of a function user to them. They have to store the information in their routing graph. On the one hand, that increases their graphs and the amount of memory required therefore. On the other hand, it increases their knowledge about functional blocks available for reuse. This, in turn, enables the calculations of routes over a "longer distance", with more gate numbers, and with a higher fraction of explicit route segments. The policies used in the study are created based on a clustering of subnetworks. Section 5.3.1 describes the creation of the policies and the assumptions of the creation.

The subsequent sections show the results, which are average values of 20 simulation runs. Each run reports average values for 20,000 established connections with best-effort requirements[8]. Some figures show error bars indicating the maximum and minimum average values of the 20 runs. Some more results regarding the stability of the statistics are given in Appendix G.1.

The study has been executed only for the GLP graph. The other graphs are too large to execute the study with all configurations in reasonable time. However, their results are expected to be similar, since these graphs belong to the same topology.

5.3.1. Creation of routing service policies and assumptions

In the study, the routing service policies are defined artificially by clustering subnetworks to groups of subnetworks. All subnetworks of a *cluster* inform all other cluster members about their graph of functional blocks. More specific, the routing service entities of all cluster members inform all other entities about their gates, gate numbers, and forwarding nodes. The larger a cluster the more information each member has to store, the more gates can be reused, and the longer the routes get.

A cluster is created by a subnetwork deciding to be a *cluster head*. A subnetwork that is not a cluster head joins the cluster formed by the nearest cluster

[8] A reusable function as in the study described in Section 5.2. If, e.g., prioritization is added as second reusable function, the number of gates would double. The number of forwarding nodes would stay constant.

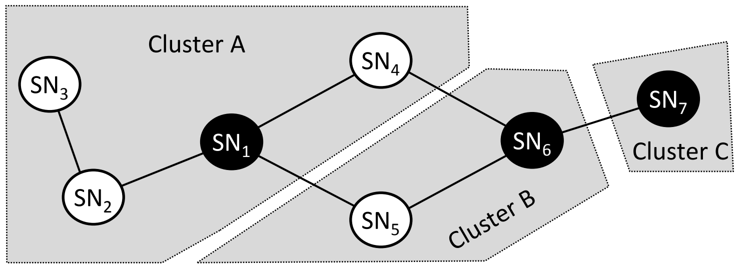

Figure 5.9.: Clustering example for an inter-network with seven subnetworks SN_i and three cluster heads depicted as black dots. The assignment of SN_4 and SN_5 to Cluster A and Cluster B, respectively, bases on random decisions.

head. If two or more cluster heads are available within the same distance, one it chosen randomly. Figure 5.9 shows a small example with seven subnetworks with the three cluster heads SN_1, SN_6, and SN_7. The cluster head is just a helpful construct to define the clusters and not of importance for the routing itself. After the exchange of information, all the cluster participants have the same routing information base and can calculate routes on their own. They do not depend on the cluster head.

The study assumes the following:

- Each subnetwork decides whether it would like to become a cluster head randomly at the start of a simulation run. If no subnetwork in the scenario decides to become a cluster head, a single one is chosen randomly. Thus, there is at least one cluster.
- The routing service entities of a cluster know each other.
- Routing service entities accept and store all information announced to them.
- Routing service entities follow a greedy approach and try to calculate explicit route segments that contain as many gate numbers as possible.
- The functional blocks are reusable for all connections (e.g. best-effort relaying functions for connections without non-functional requirements).

Not all these assumptions hold for real inter-networks. A subnetwork in the real world will most likely have a more sophisticated policy. It may decide to delegate its routing only to specific neighbors. Furthermore, the choice is not done randomly but with respect to a business plan. Since these factors are hard to model, the main purpose of the assumptions is the simplification of the scenario and the reduction of influencing factors.

The set of policies constructed by the clustering contains two important policies known from today's networks:

- Pure source routing (for route-based relaying PCI formats): The probability to become a cluster head is zero and there is only a single cluster comprising the whole network. Thus, all subnetworks know the complete graph of functional blocks and can compute explicit routes between end systems. This configuration is equivalent to a source routing one.
- Pure hop-by-hop forwarding (IP-like): The probability to become a cluster head is one and each subnetwork forms its own cluster. The subnetwork and cluster levels are identical. A routing service entity can only determine the next hop (from the inter-network point of view) via the inter-cluster level and the route through its subnetwork. This configuration is equivalent to IP routing.

Moreover, there is a broad variety of policies in between that lead to configurations that are neither supported by IP nor by source routing approaches. These configurations are enabled by FoG's flexibility.

The study uses the simulated routing service with a hierarchy of three levels:

1. Subnetwork level: Contains only the information local to a subnetwork. Routes requested on this level are routes between gateways of a subnetwork or from a gateway to a binding on an end system within the subnetwork.
2. Cluster level: Contains the information of all subnetworks of a cluster. If a route cannot be calculated on the subnetwork level, the routing service tries to calculate it with the knowledge about the whole cluster an entity belongs to. Such routes traverse the cluster or lead from the border of a cluster to a binding within. The cluster level is an example for a detailed information base I_d mentioned in Section 3.3.3.2.
3. Inter-cluster level: Contains the connectivity information between the clusters and provides the direction information required by a routing service. It is an example for the abstract information base I_a mentioned in Section 3.3.3.2. If a destination is not known on cluster level, the inter-cluster connectivity information reveals the next cluster and its gateway in the direction of the destination.

Since a node emulates a subnetwork, the setup is equivalent to the setup shown in Section 4.4.1.

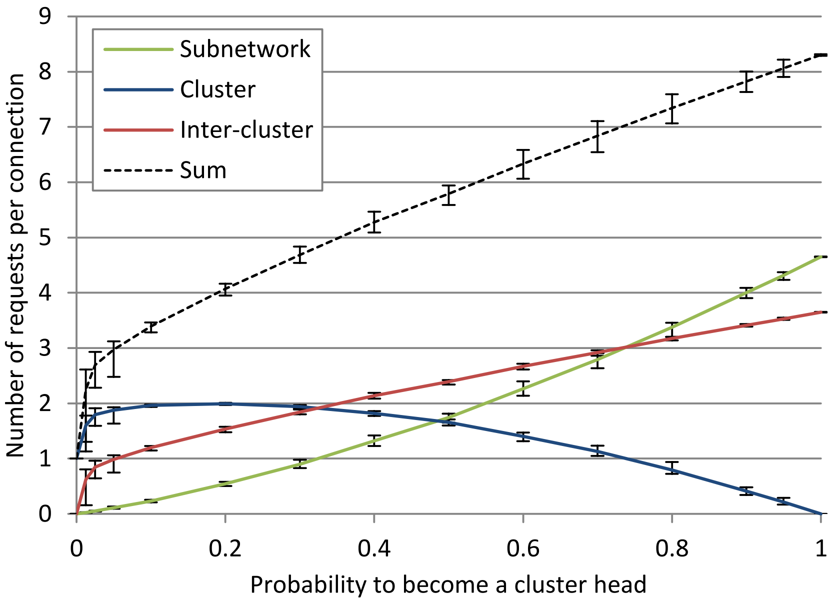

Figure 5.10.: Average number of requests to the routing service levels per connection. Error bars indicate the min./max. results.

5.3.2. Routing service requests

Figure 5.10 shows the average number of route calculations per routing service level over the probability for a subnetwork to become a cluster head. The two important special cases mentioned before are shown in the figure. On the left hand side, the probability is zero (exactly one big cluster) and each source subnetwork can calculate explicit routes to all destinations. This results in just a single routing service request on cluster level. On the right hand side, the probability is one (each subnetwork forms its own cluster) and each subnetwork can only determine the next hop via the inter-cluster level. Consequently, the source subnetwork and each relay subnetwork have to look up the next cluster in their inter-cluster level, which results in 3.65 decisions per connection. As shown in Section 5.2.1, that value marks the average number of hops per connection in the GLP graph. For each of these calculations, a way through a subnetwork has to be found. Moreover, the destination subnetwork has to calculate the way from its gateway to the end system[9]. Both results in 4.65 requests per connection on subnetwork level. In this configuration, the cluster level is not useful anymore and not contacted at all.

[9] More specific: To the forwarding node representing the destination binding on an end system.

The sum of requests grows continuously with increasing probability. This holds even if the requests from the subnetwork level are omitted. However, the sum of requests is not an estimation of the cost regarding CPU and memory. A better cost estimation has to weight each curve according to the complexity of its algorithm. Moreover, the runtime of such an algorithm usually depend on the size of the information base. This size is analyzed in the next section.

5.3.3. Size of routing service graphs

Figure 5.11 and Figure 5.12 depict the average size of the graphs on cluster and inter-cluster level, respectively. The sizes of the subnetwork graphs are independent of the probability and are equal to the size of the cluster graphs at probability one. Forwarding nodes that represent bindings and the gates required to reach them are not included since the number of bindings depends on the number of higher layer entities willing to provide a service.

As expected, the cluster level graph reaches its maximum size with approx. 76k edges and 30k vertices in the source routing configuration with probability zero. It shrinks disproportionally with increasing probability and converges to the subnetwork graph sizes at probability one. The inter-cluster graph shows the revers behavior and starts with size one (a single cluster) and increases with increasing number of clusters. Its size increases disproportionally till a probability of approx. 0.1 and continues to increase linearly to 76k edges and 30k vertices. The disproportionally increase is causes by the large number of links between subnetworks and the heavy tail distribution of the node degree.

A routing service entity has to store its cluster graph and the overall inter-cluster graph. The subnetwork graph is included in the cluster graph and is not counted separately. Figure 5.13 shows the sizes of each graph as the sum of edges and vertices. Moreover, it depicts the overall size as sum of both other curves. A routing service entity has its biggest information base in the source routing and in the hop-by-hop configurations. The minimum is at a probability of 1.25% with an average of 60.3 clusters. An even better trade-off between both graphs may be achievable with slightly less clusters. Even if such a small amount of subnetwork clusters is not achievable in an inter-network due to political and business constraints, all other configurations with a probability in the range of $[0.0125, 1)$ provide a better trade-off with a smaller overall graph size than the hop-by-hop configuration.

5.3.4. Runtime performance and trade-off

The simulated routing service uses the Dijkstra algorithm in order to calculate a route through a graph of relaying functional blocks. The runtime of the algorithm depends on the number of edges E and number of vertices V of a graph. Its worst-case complexity is $O(E + V * \log(V))$ [FT84]. For an optimal

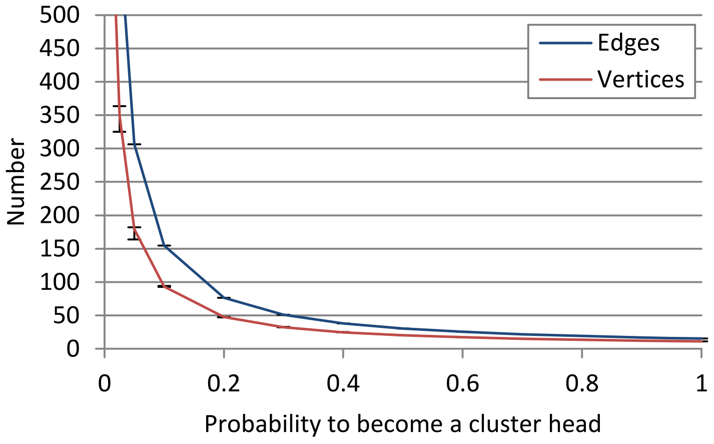

Figure 5.11.: Average number of edges and vertices of cluster level graphs. Maximum numbers at probability zero are: 76k edges and 30k vertices.

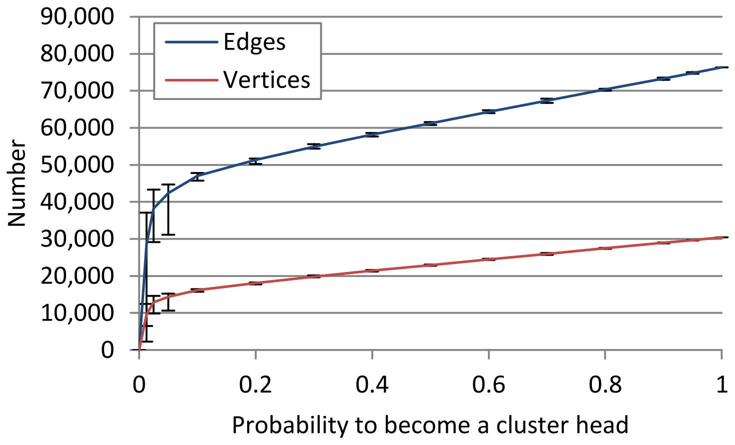

Figure 5.12.: Average number of edges and vertices of inter-cluster level graphs

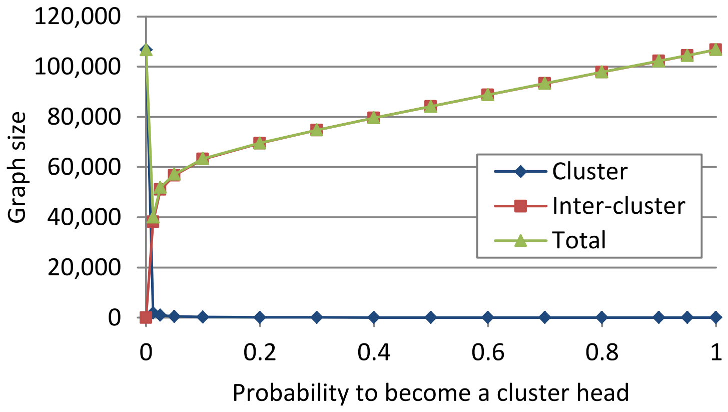

Figure 5.13.: Average sizes (number of edges plus number of vertices) of routing service entity graphs per entity in total

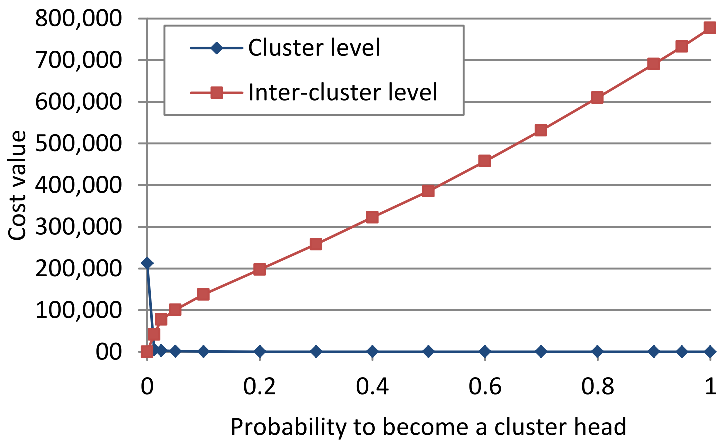

Figure 5.14.: Cost approximation for runtime of Dijkstra algorithm for calculating routes for two routing service levels.

system performance, the number of requests and the graph sizes have to be balanced. In order to approximate the runtime cost of the Dijkstra algorithm, a cost value C is derived from the number of requests per connection R and the graph sizes shown before. The value is computed as follows:

$$C = R * (E + V * \log(V)) \tag{5.1}$$

Figure 5.14 depicts the cost values for the two important routing service levels. Due to the high number of routing service requests per connection, the cost for the inter-cluster calculations is 3.6-times the cost of the cluster level calculations. The cost values for the subnetwork level are always below 126 and, thus, are negligible in relationship to the others. The sum of all three cost values has a minimum again at a probability of 1.25% as shown in Figure 5.15.

However, the inter-cluster graph represents a lot of details that are not necessary for determining the next hop. First, the graph contains a lot edges per inter-cluster link (three gates per direction). Second, not all the inter-cluster links are required for a shortest route[10]. For example, BGP stores routes (as list of autonomous system numbers) to IP prefixes. These routes can be depicted as graph with the IP prefixes attached to nodes representing subnetworks. Since a BGP entity normally does not announce all its knowledge, its graph will be less dense than the graphs of the simulated routing service entities in this study. A minimal BGP graph has a tree topology and contains only the shortest path to each subnetwork/prefix. Thus, the graph sizes of the study are an upper bound for the size of a BGP graph.

Figure 5.16 shows the raw number of edges and vertices of the inter-cluster graph without the overheads for gates and forwarding nodes. It shows the number of clusters and the links between those clusters (each link represents two gateways). The high connectivity among the core nodes of this inter-network graph causes a high number of edges. Such a reduced graph cuts the maximum cost dramatically to 54% of the maximum cost of the cluster level. The total cost minimum is still at a probability of 1.25% as shown by the "clusters and links" curve in Figure 5.15. If only a tree version of the graph is stored, the number of edges reduces even more (to the number of clusters minus one). The tree-version reduces the cost to 40% of the maximum cost of the cluster level. This moves the total cost minimum to 5% probability as shown by the "Tree" curve depicted in Figure 5.15.

5.3.5. Discussion

The results show the flexibility the FoG layer architecture offers to routing services. Partial routes in combination with the incremental routing process enable various routing service implementations and policies. Moreover, the

[10] However, they are important for error recovery as shown by the next study.

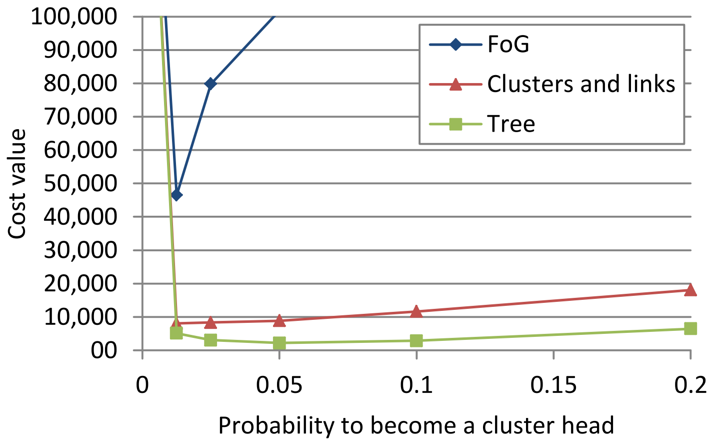

Figure 5.15.: Summed cost values of all three levels for different inter-cluster level graphs. The "FoG" curve is the sum of both curves from Figure 5.14 plus the small cost values for the subnetwork level.

architectural separation of detailed and abstract routing service information bases proved to be useful for a more general approach to routing. In particular, the study showed that hop-by-hop forwarding and source routing are two extreme cases of how the abstract and the detailed information bases relate to each other. FoG shows that other configurations are possible and that they may be even more efficient than the two extreme cases.

The delegation policies influence the amount of routing information an entity has to handle and the number of routing service requests. The study reveals that hop-by-hop forwarding and source routing configurations cause the biggest amount of routing information. Configurations between both require less routing service information. Source routing configurations require the least number of routing service requests, and the hop-by-hop forwarding configurations require the most requests. In order to determine the optimal configuration, both metrics – size of information base and number of request – have to be combined with a cost function that depends on the routing algorithm.

The optimal configuration of a routing service depends on the algorithms used by the service. The cost calculation presented in Section 5.3.4 is biased by the Dijkstra algorithm, which is used by FoGSiEm for the route computation. It requires an optimal configuration with few medium-size clusters. However, different algorithms lead to different cost equations and to different trade-offs between cluster and inter-cluster level. Most common are FIBs storing pre-computed next hops for the inter-cluster level. The size of the FIB and its access algorithm influence the runtime performance. In IP, the size of the FIB depends

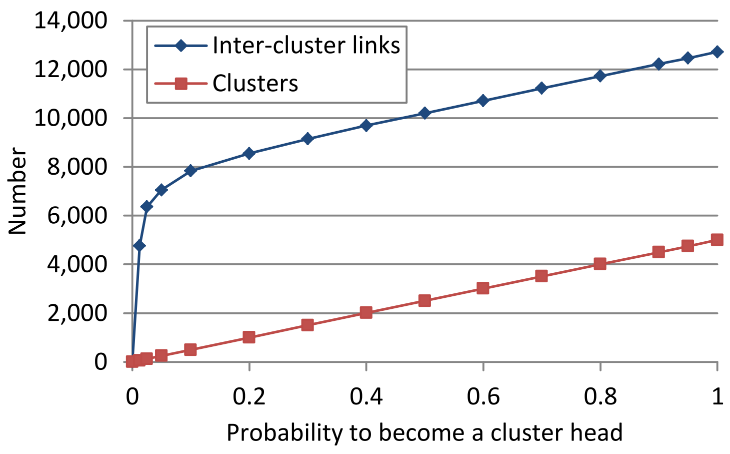

Figure 5.16.: Inter-cluster graph reduced to clusters and inter-cluster links

on the distribution of IP prefixes and how good they can be aggregated. The access algorithm has to find the shortest prefix in the FIB matching a destination address. Some more information and an enhanced algorithm are required for supporting QoS. Since the simulated routing service does not support address prefixes, an estimation of the FIB size and an estimation of the cost for accessing it is subject to future work.

The study shows that the FoG layer architecture gives a routing service enough degrees of freedom to adjust its configuration to an optimal one. Thus, *tussles* (cp. Section 3.1.4) can be resolved within the bounds of the architecture.

5.4. Robustness of connections

This study investigates robustness of connections that uses explicit routes for their packets in case of link failures. This is of special importance for FoG, because the static nature of explicit routes may hamper their adaptation in case of failures. The study analyses if FoG's transfer service can compensate this issue with its ability to repair routes of connections. The reparation bases on explicit routes, since their static nature enables a repair without having to wait for a convergence of the routing service.

An explicit route segment is the result of a routing service request for a route towards a forwarding node named in a destination segment. It encodes the routing service decision for a source and destination pair with respect to some requirements and the current network graph. Since the route is normally reused for multiple packets over a longer time, it is more static in nature than the original destination segment. Unlike forwarding based on destination segments, it does not "react" to changing conditions in a network and not to failures in particular. The early binding of a destination name to an explicit route trades reliability for relaying performance. However, the network can react to failures in order to increase the reliability again. The three *recovery algorithms* presented in Section 4.3.4 are able to coop with failures of functional blocks representing relaying functions. The study presented in this section uses these algorithms to recover from single link failures. The performance metrics are the percentage of explicit routes that can be recovered and the overhead of the recovery in terms of the length of route extensions.

The following steps have been executed 100,000 times for the GLP and 20,000 times for each of the other graphs:

- Two random nodes A and B are chosen and a connection with best-effort requirements between a client on node A and a server on node B is established. The length of the explicit route of this connection is used as reference route length.
- For each recovery algorithm R, the following steps are executed:
- 1. "Break" a link in the middle of the route between A and B. The element is marked as failed and will no longer relay packets. The gates (type DirectDownGate) using the failed link switches in the FAILED state.
  2. A packet is send along the original route from A to B. At the detector node, the packet is subject to the repair algorithm R. R tries to repair the route.
  3. The route length of the new route used by the packet in order to reach B is measured and compared with the original one.
  4. The failed element is reactivated again. All functional blocks are re-established by the nodes.

Routes and, in particular, the detour routes are calculated by a routing service entity knowing the complete network (best-case scenario; cp. probability equal to zero in previous study). Therefore, the graph of the inter-network solely determines if a repair action is successful or not. If the failure of an element partitions the graph, the repair action fails.

I chose to break the element in the middle of a route as mentioned in step 1, since the middle of a route goes normally through the better connected core nodes of an inter-network graph, this link-breaking strategy increases the probability that the repair algorithms can succeed. The higher success probability, in turn, increases the sample size of the successfully repaired connections that is the statistical base for the results shown in the following.

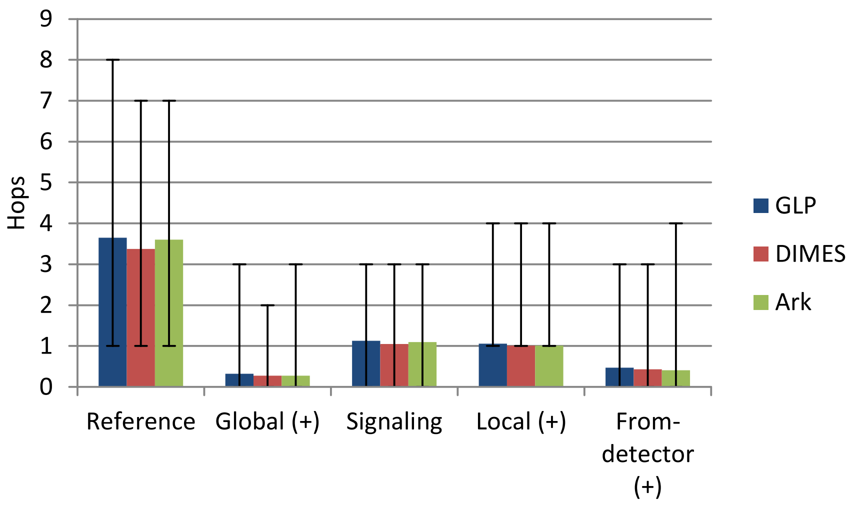

Figure 5.17.: Routes length in hops calculated by global, local, and from-detector repair algorithms in comparison to reference route length. Values marked with (+) are relative to reference. Error bars indicate minimal and maximal results.

This is a crucial issue due to the long simulation times. Some results for a random selection of failed links in the GLP graph are shown in Appendix G.2.

Figure 5.17 shows the average hops (between nodes/subnetworks) a packet has to travel for the three different graphs and for the three algorithms. The reference is the number of hops a packet has to travel without failures analog to the results presented in Section 5.2.1. Despite the number of hops a signaling message from the global repair algorithm has to travel, the other numbers are relative values to this reference. The average alternative route derived by the global repair algorithm is between 0.27 to 0.32 hops longer than the original shortest route. Its signaling messages have to travel between 1.04 to 1.12 hops from the detector to the source. The local repair algorithm extends the routes by 1.01 to 1.06 hops. The from-detector algorithm introduces detours with 0.41 to 0.47 more hops. Depending on the graph, 97 to 99% of the connections have been successfully repaired.

The results show the promising opportunities a highly connected inter-network core offers for repairing explicit routes. A relatively simple local repair algorithm can fix most of the failures with one additional inter-network hop. The complete route is approx. 29% longer as the original one. The from-detector algorithm introduces only half the overhead by avoiding loops (like the one depicted in Figure 4.8; gates right from destination forwarding node of gate g used twice). However, the global recovery algorithm shows that, in the best case, new routes are just 8.9% longer. Since the global recovery algorithm may take too long to inform all sources using routes with the failed gate, a combination of the global recovery algorithm with one of the other two seems to promise the best performance for real setups. Since the from-detector algorithm suffers from the limitations described in Section 4.3.4, a combination of the local and global recovery algorithms seems to be a good trade-off between performance and applicability.

6. Conclusions

In my thesis, I suggest a holistic layer architecture that combines a new communication model for arbitrary functions with a new method to place states in a flexible way. Its flexibility enables a broad variety of supported use cases and scalable network configurations. It is based on the recursive reference model from John Day (cp. Section 2.2.4), which provides a full encapsulation of a layer. Thus, the FoG layer architecture comprises roughly the functionality of the OSI network and transport layers. It is designed for the scope of an inter-network.

FoG opens extensive opportunities for new business models. For example, FoG opens up a neutral market for functional blocks. Similar to today's app stores for smartphones, end users may select suitable functions for their connections from a function store. The routing service would determine the function providers that support the selected functions, select the "nearest" suitable location (taking, e.g., load balancing aspects into account), request the creation of such gates, and direct the traffic to them. End users as well as network operators profit from such a function store. End users can, in particular, select function providers of their choice. Since they have more control over routes, they are able to direct the traffic to functional blocks important for them. Network operator can influence these decisions by their policies and by their function offers. Since FoG is flexible regarding the type of functions, operators can even extend their offers with new functions dynamically. For example, distributed clouds could represent their application-related functions as gates in a FoG network. Thus, the competition among operators could focus on innovative ideas for new functions instead of being limited to prices for bit transport.

FoG comes along with new opportunities for algorithms as well. The most prominent example is the FoG routing service, which has much more information about a network, its capabilities, and its supported functions than traditional routing services in the IP world. Moreover, its implementations are much more flexible since they can define their own addressing, name-to-address mapping, algorithms, and information bases. This book describes already two very different routing service implementations. More implementations for QoS-enabled routing [VMT12, VOBMT13] or strict QoS-routing are possible. Instead of optimizing the placement of functions and the routing in between individually as in today's networks, routing services in FoG can derive both in a combined way. This enables truly optimal network configurations for a holistic service provisioning by networks.

FoG reduces administrative overhead and operational expenditure by aligning the management of functions in networks and end systems. It operates in a world constructed out of functional blocks and, thus, its management is concerned with blocks only. This results in a single signaling protocol for all functions. Moreover, FoG's security features enable an automatic negotiation with users and allow accounting.

The architecture has the following unique features:

- It supports large-scale inter-networks composed of multiple autonomous subnetworks. The flexible placement of states and the incremental routing process enable scalable implementations of the transfer and the routing service, respectively. Autonomous subnetworks are enabled by giving their operators the ability to define their own policies independently. The operators can decide about their set of offered functions, their involvement in routing decisions, and their level of security.

- Connection-specific mapping states can be distributed in a more homogenous way than compared with a reference architecture that combines IntServ and DiffServ. In particular, FoG is able to reduce the load of the core relay systems of an inter-network significantly (cp. Section 5.2). The overhead regarding the information required in packets is acceptable in comparison to IP, due to the small diameter of inter-network graphs (cp. Section 5.2.1). The placement of states works best for reusable functions (see Section 3.7.1). Non-reusable functions are still a problem and FoG cannot improve the scalability for them in comparison to the reference architecture.

- Routing services are enabled to balance the number of requests with their information base size in order to adjust to an optimal configuration. Moreover, the routing information bases can contain a mixture of detailed and abstract knowledge about a network. The optimal trade-off between these three aspects depends on the algorithms used for routing. For routing service implementations that use the Dijkstra algorithm without a FIB neither hop-by-hop forwarding nor source routing are optimal. They prefer a configuration in between (cp. Section 5.2).

- FoG offers a middle ground between connection-oriented and connectionless communication (cp. Section 3.7). In contrast to the static nature of IP and MPLS networks, FoG supports both extreme cases and variations in between in one network at the same time. Network operators profit from this flexibility, because they can select a suitable mixture for each kind of application in order to support it in the most efficient way. In addition, it allows nodes to participating in a network without having addresses.

An analysis of the broader applicability of the architecture reveals the following results:

- Political: The FoG layer architecture can resolve tussles within its bounds due to its flexibility. It is suitable for socio-economic environments favoring neutral function offers by function providers. It is not applicable, if political constraints forbid any non-best-effort relaying functions (e.g. relaying with non-functional guarantees) in a network.
- Deployment: The success of a new architecture depends strongly on the business opportunities and political constraints. Only if the enhanced features and reduced operational expenditure due to less administrative overhead open business opportunities, FoG has a change to be adopted.
- Interoperability: FoG can interoperate with IP either via adapted interfaces on end systems or by gateway systems connecting both worlds. Various interoperability solutions lower the burden of a deployment by integrating legacy networks and applications in a FoG network.
- Robustness: The architecture supports various algorithms for repairing routes. Three different recovery algorithms known from literature have been adapted to FoG. They increase the reliability of connections in case of failures by repairing explicit routes in different ways. The study in Section 5.4 shows the good performance of these algorithms in finding short alternative routes in the core of an inter-network. In contrast to IP networks, the algorithms do not depend on synchronized routing service information bases.
- Security: The architecture enables network entities to check the authenticity of messages and provides a base for authorization and accounting. In contrast to the IP suite, the authentication is part of the FoG layer architecture and, thus, integrated in a scalable way.
- Mobility: The architecture supports mobility by adjusting routing information and by adapting the packet relaying. The latter is based on tunnels and is comparable to MobileIP. The mobility support illustrates the strategic advantages of the recursive reference model, which encapsulates a layer and enables FoG to consider mobility as dynamic multihoming.

The feasibility of the architecture is shown by an implementation called FoGSiEm. FoGSiEm supports simulations of large-scale networks with more than 25,000 nodes. Moreover, it is able to replace IP in real networks by implementing a FoG network directly on top of Ethernet. The software is available as open source [FoG] and runs on Windows, Linux, and OSX.

7. Outlook

FoG is the base for various new research opportunities, which are only partially covered by my work. In the following, some promising research questions are highlighted.

The trade-off between the detailed and abstract information bases (in Section 5.3: cluster and inter-cluster level, respectively) was analyzed for one routing service implementation that used the Dijkstra algorithm. The trade-off is of interest for other routing service implementations as well. In particular, the trade-off for a BGP-like routing service with FIBs is of interest in order to evaluate if the Internet is configured in an optimal way. A comparison of the trade-offs for multiple routing service implementations might reveal some new aspects of implementation strategies for routing services in general.

The requirements mapper component of the routing service determines the set of functional blocks that satisfy some requirements. My thesis includes a basic implementation that proves its feasibility (cp. Section 4.4.3). However, this component could be enhanced with more intelligent algorithms in order to combine functional blocks more flexibly and to increase the reuse potential of blocks. It seems to be beneficial to build upon other selection and composition approaches, such as SONATE, and to extend them regarding the chain parts ("workflows" in SONATE) required on relay systems. Moreover, the mapping has similarities with at least two other problems. First, the mapping from QoS to NoS ([Day08a] page 43/44) includes the same non-functional aspects. Second, the mappings between physical, transport, and logical channels in UMTS access networks (cp. Section 5.2 and 6.3.1 in [HT11]) seems to be a special case. The invariances of these problems might reveal deeper insights into the general problem of mapping higher layer requirements to lower layer capabilities.

Ensuring the authenticity of messages is a first step towards a more secure inter-network. The discussion about attacking a system by guessing gate numbers on page 92 suggests that it is not the last step. A deeper analysis of the security aspects surrounding the guessing or sniffing of gate numbers is required. One extension of the current implementation could, for example, encrypt the signaling messages and introduce a scheme for switching gate numbers after some defined time. Thus, an attacker may sniff the initial gate number, but it is not aware of the switch. If packets with the initial gate number are received after the switching time (and if they are not just delayed), they might be an indication for a function provider to watch out for attackers.

Failures and performance degradations in a layer are signaled to upper layers with events. FoGSiEm exploits this mechanism to notify upper layers, for example, about failed repair actions or newly created bindings. Thus, FoG is able to handle scenarios with links having a variable data rate as described in [DEB12] for satellite links. Whether this mechanism is sufficient to handle all problems related with wireless communications and, thus, to prevent "cross-layer" information flow, is an open question. This could be investigated by combining FoG with more complex wireless lower layers encapsulating, for example, cognitive radio links.

Finally, the ideas of FoG can be transferred to networks not operating with arbitrary functional blocks but relaying functions only. The comparison of FoG with the reference model (in Section 3.7.2) raised already the question if arbitrary functional blocks or application-specific layers are more suitable for a use case. In such networks, gates would represent only transmission capabilities of lower layers between FoG nodes. A comparison of FoG with the expected results from the IRATI project mentioned in Section 2.2.4.1 may answer this question. However, most results of the performance studies already apply to networks with such a reduced set of functions.

A. Protocol control information formats

The following list contains the format of the PCI of several protocols. The payload is omitted if no trailer is present.

- ATM: Format for Network-Network-Interface (NNI) [MP02]
  - Virtual path identifier (12 bit): Names a virtual path
  - Virtual channel identifier (16 bit): Names a virtual channel
  - Payload type (3 bit): User data or signaling payload
  - Cell loss priority (1 bit): Priority in congestion cases
  - Header error control (8 bit): Header check sum
- CIGALE / CYCLADES [Pou74a]:
  - Header format (4 bits)
  - Header length (4 bits)
  - Text length (8 bits): Length of payload
  - Packet identification (16 bits): Not altered by network; just used for error reports
  - Facilities, accounting (16 bits)
  - Destination network (8 bits)
  - Source network (8 bits)
  - Destination host (16 bits)
  - Source host (16 bits)
  - [Time-out (3 bit): Planned for future version to prevent packets from looping infinitely.]
- CLNP [Int86]:
  - Network layer protocol identifier (8 bit): Equal to binary 1000 0001 for ISO 8473
  - Length indicator (8 bit): Header length in octets

  - Version / protocol Id Extension (8bit): Version of the standard (equal to one)
  - Lifetime (8 bit): Remaining lifetime of the PDU in units of 500ms
  - Flags (3 bit): Flags for segmentation and error reporting
  - Type (5 bit): Type of PDU
  - Segment length (16 bit): Length of the PDU
  - Checksum (16 bit): Checksum over the whole PDU
  - Destination address length indicator (8 bit): Length in octets
  - Destination address (variable): NSAP address as defined in ISO 8348/DAD2
  - Source address length indicator (8 bit)
  - Source address (variable)
  - Segmentation part (optional)
    * Data unit identifier (16 bit): Identifies segments of a PDU
    * Segment offset (16 bit): Relative position of segment in PDU in octets
    * Total length (16 bit): Total length of the initial PDU in octets
  - Options part (optional): List of options such as padding, source routing, security, route recording, QoS maintenance, and priority. Each option is encoded as follows:
    * Parameter code (8 bit): Type of the parameter
    * Parameter length (8 bit): Length of the parameter value in octets
    * Parameter value (variable)

- DDP [App94]: Long header (short header does not contain network numbers) without link layer frame:
  - Unused (2 bit)
  - Hop count (4 bit)
  - Datagram length (10 bit): Length in octets (maximum of 586 according to implementation guide)
  - DDP checksum (16 bit): Optional checksum (zero if not set)
  - Destination network number (16 bit): Identifier of the destination subnetwork
  - Source network number (16 bit)
  - Destination node ID (8 bit): Identifier of the destination node

    - Source node ID (8 bit)
    - Destination socket number (8 bit)
    - Source socket number (8 bit)
    - DDP protocol type (8 bit): Type of data transported by packet

- Ethernet [IEE08]:Without physical layer parts such as preamble, start of frame delimiter, and interframe gap.
    - Destination MAC address (48 bit)
    - Source MAC address (48 bit)
    - Ethertype or length (16 bit)
        * Ethernet II: Ethertype indicates the higher layer protocol
        * IEEE 802.3: Length of payload (max. 1500 octets)
    - Payload (42-1500 octets): Includes optionally a 802.1Q tag (4 octets)
    - Frame check sequence (32 bit)

- Frame relay [Sta93]:
    - Flag (1 octet)
    - Address (2-4 octets)
        * DLCI (6 bit +4bit [+7 [+7]])
            · DLCI = 0 is signaling to frame relay control point
            · DLCI = 8191 is management
        * Command/response (use is app specific) 1 bit
        * Address field extension 1 bit at the start of each octet (1=end of address)
        * BECN backward explicit congestion notification
        * FECN forward explicit congestion notification
    - Information (variable): user data
    - FCS 2 octet and frame sequence field as in LAPD and LAPB
    - Flag (1 octet)

- IPv4 [Pos81]:
    - Version (4 bit): Indicating the version of the IP protocol (equals 4)
    - Internet Header Length (4 bit): Length of header in 32 bit words
    - Type of service (TOS, 8 bit): Indicates the requirements and the desired quality of service for the packet. It includes:

        * Precedence (3 bit)
        * Delay (1 bit)
        * Throughput (1 bit)
        * Reliability (1 bit)
        * Reserved (2 bit)
    - Total length (16 bit): Length of datagram in octets
    - Identification (16 bit): Aid for assembling fragments
    - Flags (3 bit): Control flags for fragmentation
    - Fragment offset (13 bit): Offset of a fragment in 64 bit steps
    - Time to live (TTL, 8 bit): Remaining number of hops the packet is allowed to remain in the Internet. Originally designed as time limit in seconds. However, each node has to decrement the value regardless the time it requires for relaying the packet.
    - Protocol (8 bit): Indicates the higher layer protocol
    - Checksum (16 bit): Checksum of the header
    - Source address (32 bit)
    - Destination address (32 bit)
    - Options (variable): Various options, e.g., source routes, route recording, stream identifier, and time stamps. Padding has to be used in order to align header to next 32 bit boundary.

- IPv6 [DH98]:
    - Version (4 bit): Indicating the version of the IP protocol (equals 6)
    - Traffic class (8 bit): Support for "differentiated service" analog to the TOS field of IPv4
    - Flow label (20 bit): In combination with source address, the flow label identifies a sequence of packets, which might be subject for QoS.
    - Payload length (16 bit): Length of the payload in octets
    - Next header (8 bit): Identifies the next higher layer, like the protocol field in IPv4. It might indicate the presents of IPv6 extension headers. Such headers are located in the payload field and are following the normal header.
    - Hop limit (8 bit): Remaining number of hops the packet is allowed to remain in the Internet.
    - Source address (128 bit)

  - Destination address (128 bit)

- IPv6 extension headers defined in RFC 2460 [DH98]:
  - Hop-by-Hop Options
  - Routing (Type 0): Source route
  - Fragment: Datagram fragmentation done by source
  - Destination Options
  - Authentication
  - Encapsulating Security Payload

- IPX (Chapter 31 in [FLSS99]): IPX names contain a 32 bit network name and a 48 bit node name.
  - Checksum (16 bit)
  - Packet length (16 bit): Length of header and payload
  - Transport control (8 bit): Number of routers passed (max. 16)
  - Type (8 bit): Higher layer protocol (e.g. SPX)
  - Destination name (80 bit)
  - Destination socket (16 bit)
  - Source name (80 bit)
  - Source socket (16 bit)

- MPLS [RTF+01]:
  - Label (20 bit): Name for the LSP
  - Experimental use (3 bit): For example for traffic classes analog to TOS field of IPv4
  - Bottom of stack (1 bit): Indicates if this label is the last one of the stack
  - Time to live (8 bit): Remaining number of MPLS router the packet is allowed to travel through.

- OvIP [FRM97]:
  - OvIP protocol value (8 bit)
  - Source route (variable): List of forwarding directives. The length of a forwarding directive depends on the encoding of a routers decision how to forward a packet. In general, it contains the outgoing network interface, link layer encapsulations, and an indication if OvIP is terminated.

- PARIS [CG88]: Format without delimiters:
  - Control (16 bit)
    - Priority (2 bit)
    - Copy bit (1 bit): Copy packet to network control unit for signaling purposes
    - Broadcast bit (1 bit): Copy packet to all outgoing links
  - Automatic network routing (variable): List of two-or-more bit words. Contains one entry per intermediate hop
- PIP [Tsu92]: PIP was developed in the IPng context. Later it was merged with the Simple Internet Protocol to SIPP [Fra94].
  - ID type (4 bit): Length and type of source and destination IDs in octets
  - Options length (4 bit): Number of 32 bit options
  - Total length (24 bit): Total datagram size in octets
  - Protocol (8 bit): Equivalent to protocol field in IPv4
  - Handling directive (16 bit): Non-route-effecting QoS information (e.g. congestion avoidance)
  - Hop count (8 bit): Equivalent to hop limit in IPv6
  - Routing directive
    - Tunnel (32 bit): Override the routing hint within a domain. It enables efficient "self-encapsulation".
      - Source Exit ID (16 bit): Source of tunnel in order to enable error messages
      - Destination Exit ID (16 bit): Destination of tunnel where the routing directive ends
    - Logical router (18 bit): Selects the logical role of a router (e.g. the active FIB for a packet). It might include route-effecting QoS information required by the routing hint (e.g. metrics and selection of logical role of a router), hierarchy level, or multicast indication.
    - Routing hints
      - Length (4 bit)
      - Descriptor (10 bit)

        Routing field offset (6 bit): Currently active hint Routing field length (4 bit): Length of hints (all have the same length)

        · Hints (variable): Used by routing if no tunnel in the routing directive is given. It includes addresses, source-routes, and virtual circuit information. Between to hints, a 2-bit routing hind field relator indicates the hierarchical relation between two entries (up, down, none).

  - Options (variable): Optional
  - Source and destination IDs (variable): Optional labels identifying source and destination (with a reference to NIMROD)
  - Padding to next 32 bit word boundary

- PUP [BSTM80]: PUP names contain an 8 bit network name and an 8 bit host name. The payload contains up to 532 octets. The designers of PUP are aware of the short names and suggest prolonging them for larger networks (from the perspective of 1979). It does not support fragmentation. It supports explicit congestion notification (to sockets).
  - Pup length (16 bit): Length of datagram in octets
  - Transport control (8 bit): Includes hop count and options flags
  - Pup type (8 bit): Format of Pup content
  - Pup Identifier (16 bit): Used, e.g., for sequence numbers (included in Pup in order to enable error messages related to an identifier)
  - Destination name (16 bit)
  - Destination socket (32 bit)
  - Source name (16 it)
  - Source socket (32 bit)
  - Payload (variable between 0 and 532 octet)
  - Software checksum (16 bit): Optional checksum
- SIPP [Hin94]:
  - Version (4 bit): IP version number 6
  - Flow label (28 bit)
    * Reserved (1bit)
    * Drop priority (3 bit): Relative priority in case of congestion
    * Flow ID (24 bit)
  - Payload length (16 bit): Length of payload following the header in octet
  - Payload type (8 bit): Equivalent to IPv4 protocol field

  - Hop limit (8 bit): Hop counter decremented by 1 per relay system
  - Source address (64 bit)
  - Destination address (64 bit): If the optional routing header is present, it might be just an intermediate destination.
  - Options (variable): Routing, fragmentation, authentication, security encapsulation, and hop-by-hop options

- PFRI [BCG+06]:
  - Forwarding directive (Where) [CGP07]:
    - Partial path (variable): Specifies a sequence of link addresses (nodes are not named) a packet should traverse. The list might be "partial" meaning that is might contain gaps, which have to be resolved by relay systems (like loose source-routing in IP).
    - Current location: Current position in the partial path
    - Flag: Indicates if a packet is a signaling packet or data packet
  - Motivation (Why): Encodes why a relay system should relay a packet (e.g. because the source is a customer of operator of a subnetwork). Enables that the "customer-provider relationships can exist apart from" a graph.
  - Accountability (Who): Authenticates the entity responsible for a packet.
  - Knobs (How): Specify requirement for the packet transmission
  - Dials (What): Transporting information about errors or states between layers in order to perform cross-layer optimization.

- Viper [Che89]: Viper is an implementation of Sirpent. It supports cut-through relaying in addition to store-and-forward. Timestamps in trailer of packet for transport layer protocols (e.g. VMTP) in order to avoid TTL field and its modifications.
  - Sequence of header segments (per intermediate router)
    - Port info length (8 bit)
    - Port token length (8 bit): Zero indicates that the token is not present
    - Port (8 bit): Identifies output port (multiple segments to model larger port numbers)
    - Flags (4 bit): E.g. congestion handling
    - Priority (4 bit)

        * Port token (variable): Optional authorization token for the port
        * Port info (variable): Information required by the port. The format depends on the network and/or the router. For Ethernet network, it contains the information Ethernet requires to transmit the packet (e.g. addresses).
    - Payload (0 to 1500 octets)
    - Trailer: Sequence of header segments tracing the return route

- X.25 Packet Layer Protocol: The X.25 Packet Layer Protocol is used between the Data Terminal Equipment (DTE) and the Data Circuit-terminating Equipment (DCE). In non-X.25 terms, the protocol is used between a host and a network. Thus, it is a network access protocol and not the protocol used within the network.
    - General format identifier
        * Qualified data bit (1 bit): Data for user or PAD
        * Delivery confirmation bit (1 bit): Acknowledgment requested
        * Protocol identification (2 bit): Defines if the packet has a module 8 or 128 or an extended sequence scheme.
    - Logical channel identifier (12 bit): This field is composed of the following subfields.
        * Logical channel group number (4 bit)
        * Logical channel number (8 bit): Logical channel number of the DTE-DCE link in the channel group
    - Packet type: The type of the packet depends on the command exchanged between the host and the network. Examples are call accept, call request, data packet, and reject. The format of this field depends on the sequence scheme:
        * Module 8:
            · Packet receive sequence number (2 bit)
            · More data bit (2 bit): Indicates that the packet belongs to a sequence of packets
            · Packet send sequence number (2 bit)
            · Reserved (1 bit)
        * Module 128:
            · Analog to module 8 but with 7 bit sequence numbers and only a 1 bit more data field.

B. FoG layer interface

The interface designed for the FoG layer architecture is a non-blocking name- and requirements-based interface, which provides feedback via events. The following Java classes show the most important functions of the interface. They have been extracted from FoGSiEm. The full class definitions can be accessed via the FoGSiEm homepage [FoG].

B.1. EventSource

The `EventSource` class (Listing B.1) is a base class for objects that informs others about asynchronous events.

Listing B.1: EventSource

```
public interface EventSource
{
        /**
         * Registers observer for the event source.
         *
         * @param observer entity, which will be informed
             about event
         */
        public void registerListener(EventListener
           observer);

        /**
         * Unregisters observer for the event source.
         *
         * @param observer entity, which should be
             removed from the observer list
         * @return true, if observer had been
             successfully unregistered; false otherwise
         */
        public boolean unregisterListener(EventListener
           observer);

        /**
```

```
	 * Inferface for observer of the event source
	 */
	public interface EventListener
	{
		/**
		 * Called if an event is occuring at the
		 * event source. This callback method is
		 * not allowed to block. It must return
		 * as fast as possible since it it
		 * executed in the thread of the event
		 * source.
		 *
		 * @param source Source of the event
		 * @param event Event itself
		 * @throws Exception On error; Exceptions
		 *    are ignored by the caller.
		 */
		public void eventOccured(Event event)
		   throws Exception;
	}

}
```

B.2. Layer

A `Layer` (Listing B.2) offers the possibility to announce own services to other peers of a layer and to access services from others. Moreover, the class provides methods for retrieving neighbor and capability information that may guide the usage of a layer. All internal issues of the layer, like addresses, protocols, and routes, are hidden. Users of a layer are not allowed to get knowledge about such issues in order to preserve the encapsulation of a layer.

Listing B.2: Layer subinterface

```
public interface Layer extends EventSource
{
	/**
	 * Registers an entity with a given name at the
	 * layer. Afterwards, clients can connect to
	 * this service by using the same name. This
	 * method does not block. Errors are indicated
	 * via events of the Binding object.
	 *
 * @param parentSocket Optional parent connection
      (optional; might be null if no)
 * @param name Name for the service
 * @param requirements Description of the service
      requirements. These requirements are
     enforced for all connections to this binding.
 * @param identity Optional identity of the
     requester of the registration
 * @return Reference to the service registration
     (!= null)
 * @throws NetworkException On error
 */
public Binding bind(Connection parentSocket, Name
     name, Description requirements, Identity
    identity);

/**
 * Connects to a Binding with the given name.
 * The method does not block. The establishment
 * of a connection is signaled with an event.
 * All other feedbacks are given via events as
 * well. In particular, the error exceptions are
 * not triggered by the method but handed over
 * via events of the Connection object.
 *
 * @param name Name of the Binding, to which
     should be connected to
 * @param requirements Description of the
     requirements of the caller for the connection
 * @param requester Optional identity of the
     caller. It is used for signing the connect
     request.
 * @return Reference for the connection (!= null)
 */
public Connection connect(Name name, Description
    requirements, Identity requester);

/**
 * Checks whether or not a Binding with this
 * name is known by the layer. This does not
 * imply that the connect method can construct
 * a connection to this name.
 *
```

```
     * @param name Name to search for
     * @return true, if name is known; false
       otherwise
     */
    public boolean isKnown(Name name);

    /**
     * Determines the capabilities of this layer.
     * Since the whole set of capabilities may be too
     * large, the request can be filtered. Possible
     * filter sare the destination name and some test
     * requirements. If such filters are present, the
     * method just determines the capabilities
     * regarding this destination and these test
     * requirements.
     *
     * @param name Optional destination name to focus
         the capability analysis
     * @param requirements Optional test requirements
         (if, e.g., maximum bandwidth is included in
       the test requirements, the method will
       determine the possible bandwidth)
     * @return Capabilities of the layer (!= null)
     * @throws NetworkException On error (e.g. filter
         invalid)
     */
    public Description getCapabilities(Name name,
       Description requirements) throws
       NetworkException;

    /**
     * Determines neighbor information about the
     * Bindings reachable via this layer.
     *
     * @param namePrefix Optional filter for the
       request. If present, only neighbors with a
       name having this prefix will be listed.
     * @return List of reachable neighbors or null if
         lower layer is broken
     */
    public NeighborList getNeighbors(Name namePrefix)
        throws NetworkException;
}
```

B.3. Binding

A `Binding` (Listing B.3) is an service offering to all peers with access to a layer. It can be created at a layer via the method `Layer.bind`. Others can create a `Connection` to a binding via Layer and with the name of the binding. A binding provides methods for terminating the service offering and for retrieving incoming connections for it.

Listing B.3: Binding subinterface

```
public interface Binding extends EventSource
{
        /**
         * Requests the next new incoming connection for
         * a binding. The method does not block and will
         * return null, if no connection is available.
         *
         * @return Reference to a new incoming connection
              or null if none waiting
         */
        public Connection getIncomingConnection();

        /**
         * @return Number of new connections waiting in
             queue
         */
        public int getNumberWaitingConnections();

        /**
         * @return Name used for this binding
         */
        public Name getName();

        /**
         * Closes registration and makes the binding
         * unaccessible for peers. The method does not
         * block.
         */
        public void close();
}
```

B.4. Connection

A `Connection` (Listing B.4) represents the possibility of two or more peers to exchange data. It can be set up via a layer with the method `Layer.connect`. Depending on the requirements used to set up the connection, the features of a connection differ. The features may include (but are not limited to) encryption, in-order data delivery and error-free transmission. Most important are the methods for data exchange (`read`/`write`) and for terminating a connection.

Listing B.4: Connection subinterface

```
public interface Connection extends EventSource
{
	/**
	 * Check status of the socket.
	 *
	 * @return If sendData can be called without
	 *   errors
	 */
	public boolean isConnected();

	/**
	 * @return The name of the binding the connection
	 *    was established to.
	 */
	public Name getBindingName();

	/**
	 * Signatures may be provided by remote peer(s)
	 * to verify their authenticity. Whether such
	 * signatures are available depend on the remote
	 * peer and which parameter it used for its call
	 * to Layer.connect.
	 *
	 * @return List of signatures (!= null). If no
	 *   signatures are available an empty list is
	 *   returned.
	 */
	public LinkedList<Signature> getAuthentication();

	/**
	 * The requirements for a connection are
	 * influenced by the requirements of a Binding,
	 * the requirements of the Layer.connect call,
 * and further requirements added by the layer
 * management.
 *
 * @return Set of requirements used for this
    connection
 */
public Description getRequirements();

/**
 * Sends data through the layer to all peers of
 * the connection. The method blocks as long as
 * required to access the buffer of the send
 * data. Depending on the implementation, the
 * method may return after copying the data to a
 * buffer, to wait until such a buffer is free,
 * or to wait until the data was send. However,
 * it never blocks until an acknowledgment is
 * received.
 *
 * @param data Data to send
 * @throws NetworkException On error during
    sending (e.g. connection is closed)
 */
public void write(Serializable data) throws
   NetworkException;

/**
 * Called by application in order to get new data
 * received by this socket. This method does not
 * block and returns null if no data is
 * available.
 *
 * @return Received data object
 * @throws NetworkException On error (e.g.
    connection is closed)
 */
public Object read() throws NetworkException;

/**
 * @return Number of available bytes (if stream
    is used) or objects (if read is used)
 */
public int available();
```

```
        /**
         * Terminates the possibility to exchange data
         * via this connection. If the connection is
         * closed at the other peers depend on the
         * requirements of a connection and the number
         * of peers. The method does not block.
         */
        public void close();
}
```

C. Productions for mapping requirements to functions

Section 4.4.3 introduces an algorithm that maps requirements to required functions. The following modified context free grammar for describing valid chains of gates has been used. Some variables structure the mapping by representing the following ideas:

- S_a: Original stream from application and start symbol
- S_f: Fragments of original stream, which might be transmitted several times
- S_x: Not readable fragments (maybe encrypted)

The star (*) is a terminal symbol representing any other valid chain (in particular of TransparentGates and DirectDownGates).

1. VirusFree: $S_a \rightarrow CS_a \mid S_aC$
2. Transport: $S_a \rightarrow T_dS_fT_u$
3. Encryption: $S_a \rightarrow E_dS_xE_u$
4. $S_a \rightarrow S_f$
5. Base64: $S_f \rightarrow B_dS_xB_u$
6. VirusFree: $S_f \rightarrow S_fCS_f$
7. VideoOSD: $S_f \rightarrow S_fV_{OSD}S_f$
8. VideoDecoding: $S_f \rightarrow S_fV_{dec}S_f$
9. Intermediate: $S_f \rightarrow S_fS_f$
10. $S_f \rightarrow S_x$
11. Encryption: $S_x \rightarrow E_dS_xE_u$
12. $S_x \rightarrow *$

The following productions translate variables to gate type names (terminals):

13. $B_d \rightarrow$ Base64EncoderGate
14. $B_u \rightarrow$ Base64DecoderGate
15. $E_d \rightarrow$ EncryptionEncoderGate
16. $E_u \rightarrow$ EncryptionDecoderGate
17. $T_d \rightarrow$ NumberingGate
18. $T_u \rightarrow$ OrderAndCheckGate
19. $C \rightarrow$ VirusScanGate
20. $V_{dec} \rightarrow$ VideoDecodingGate
21. $V_{OSD} \rightarrow$ VideoOSDGate

This and further examples for productions can be found at the FoGSiEm home page [FoG].

D. Performance studies for implementation

The following studies focus on the performance of the FoG implementation FoGSiEm. In contrast to the studies in Chapter 5, they are not related to architectural issues. They focus on the performance of FoGSiEm in the context of the use case video streaming, which was defined by the SIG "Functional composition". It was chosen as an example application due to its growing importance for the Internet [Cis12]. However, it is an open question whether video streaming will be the "killer application" for a future Internet.

The studies verify that FoGSiEm is suitability for demonstration purposes. They can be summarized as follows:

- **Video streaming performance:** Video streaming was defined as a common use case by the SIG "Functional composition". FoGSiEm has to support this use case. The study in Section D.2 shows the high influence of the CPU load of the video handling on the emulator performance. The demonstration setup can transmit and display up to three video streams generated by a native IP application without degrading the video quality.
  - Influencing factor:
    - Number of parallel video streams
  - Metrics:
    - CPU load
    - Data rate on link
- **Throughput:** In order to analyze the emulator performance without the video processing overhead the maximum throughput on application level without this overhead was measured. The study in Section D.3 shows that the prototype achieves a throughput of 50 Mbit/s, which is sufficient for demonstration purposes.
  - Influencing factors:
    - Architecture (UDP/IP vs. FoG)
    - Size of payload in one packet
  - Metric:

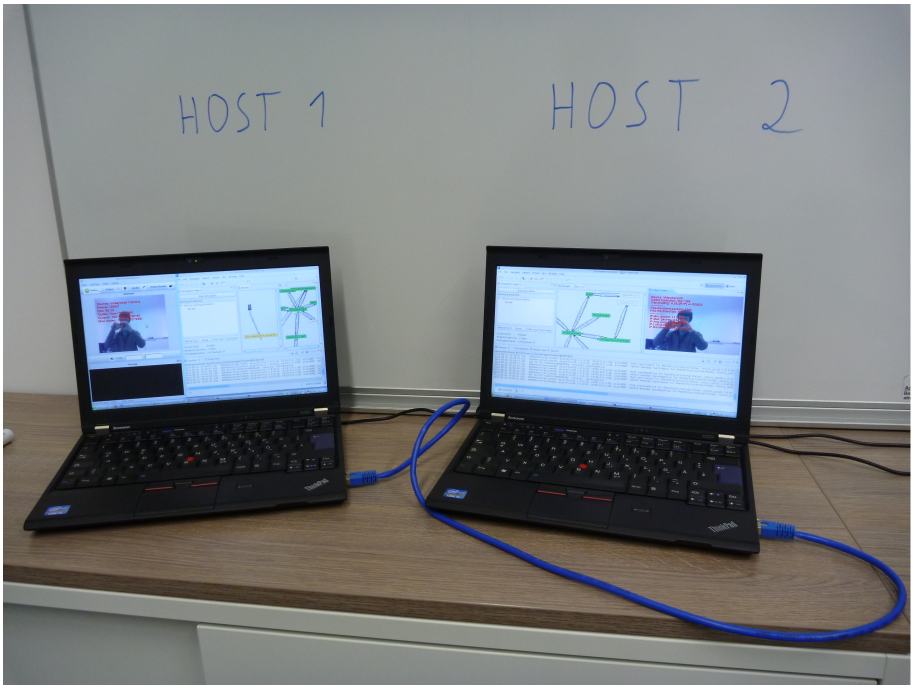

Figure D.1.: Picture of the emulator setup with two notebooks and a live video stream.

* Maximum throughput

The studies had been executed with the software described in Chapter 4 in emulation mode. Their network setup with real equipment is described in Section D.1.

D.1. Emulation setup

FoGSiEm can emulate a FoG network directly on top of Ethernet as described in Section 4.6. The studies regarding the performance of the emulation are conducted with two Lenovo X220i notebooks (each with 4 GB RAM and an Intel i3-2310M CPU with 2.1 GHz) with OpenSUSE 12.1 and Linux 3.1.0. Both are directly connected to each other via a 1 GBit/s Ethernet link. The notebooks had been chosen to create a mobile emulation setup that can be taken to conferences and trade-fairs. Figure D.1 shows a picture from the setup.

The FoGSiEm emulator setup transports FoG packets directly over Ethernet. IP is not involved for the transmission. The emulation uses the gateway interoperability solution implemented by Thomas Volkert in order to connect the IP applications to FoG. The interoperability uses a local loopback and does not transmitting IP packet over the Ethernet link.

The emulator setup uses the FoG-BGP routing service with manually assigned address prefixes and autonomous system identifiers.

D.2. Video streaming performance

This study measures the performance of FoGSiEm in the context of a video streaming use case that was defined by the SIG "Functional Composition". Therefore, it measures the number of parallel video streams FoGSiEm can transmit without a degradation of the video stream.

A variable number of parallel video streams is transmitted through the FoG network described in Section D.1. The details of the setup are depicted in Figure D.2. The Homer-Conferencing software version 0.24.0 [Hom] captures a live video from the webcam of the notebook Host 1 and streams the video to a local IP port, which represents the interoperability gateway to FoG entity A. FoG entity A is attached to one lower layer entity, which represents the Ethernet. The entity sends FoG packets from the Java VM via the JNI to LibNet as described in Section 4.6. During the setup, FoG entity A established a gate to FoG entity B by setting up a connection through the Ethernet layer. On Host 2, LibPCap captures the Ethernet packets and hands them over to FoG entity B via JNI. The receiver is a FoG-enabled application, which receives FoG packets directly. Since the viewer application runs in the same address space as FoGSiEm, no further inter-process communication is required. The video decoding is done by a VideoDecodingGate. This gate type was implemented by Thomas Volkert in the plug-in *fog.video* [FoG]. The uncompressed RGB video frames are handed over to the viewer via the Connection interface.

Homer-Conferencing is streaming a live video with the following characteristics:

- Video size: 352x288
- Frame rate per second: 30
- Codec: H.261
- Quality: 100%

For the video transmission, it measured the following:

- IP bandwidth[1]: 2.064 Mbit/s

[1] Estimation of Homer-Conferencing including IP header, UDP header and RTP header.

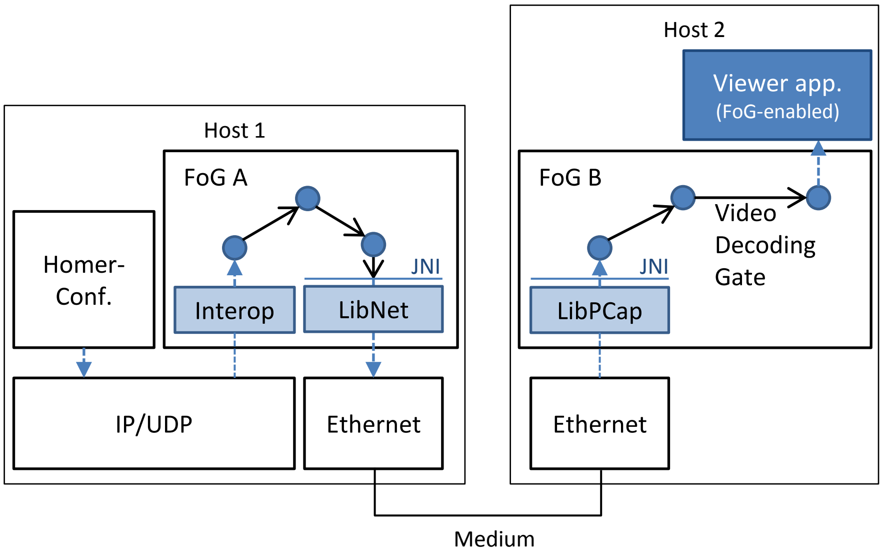

Figure D.2.: Setup for video throughput measurements

- Average packet size: 1100 bytes [min: 18, max: 1184]

Figure D.3 shows the average CPU load of some programs and the Ethernet data rate required to transmit multiple streams. Homer-Conferencing constantly requires 3% of the CPU on Host 1. The CPU load of the video decoding and resizing to the size of the viewer is the limiting factor because it increases with the number of transmitted parallel streams. Besides FoGSiEm A, the X Window system of Host 2 generates significant CPU load. In total, three streams lead to a CPU load of 60% at Host 2. Four parallel streams overload its CPU, which causes dropped packets and missed video frames. Since the output is not comparable any longer, the results for four streams are not shown in the figure. The CPU load of FoGSiEm A increases as well but is far from critical.

The transmitted data rate is much higher than the "IP bandwidth" approximated by Homer-Conferencing. The overhead is introduced by the encoding of a FoG packet as serialized Java object. Since all packet elements such as route segments, gate numbers, and authentication information are encoded as Java objects, a typical FoG packet in this setup has a serialized size of 1238 bytes without payload. Including the average payload size of 1100 bytes, this sums up to an average FoG packet size of 2338 bytes. Such FoG packets are fragmented and transmitted with two subsequent Ethernet frames.

The video streaming performance of FoGSiEm is limited by the CPU rather

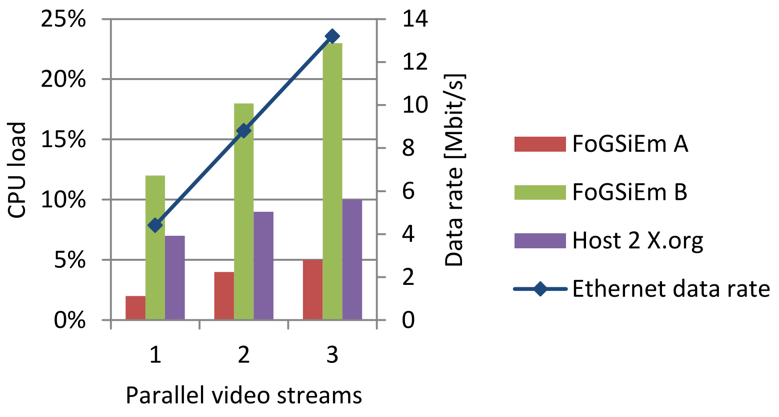

Figure D.3.: CPU load and Ethernet data rate required to transmit several parallel video streams

than the data rate of Ethernet. Thus, faster computers would enable even more paralell video streams. However, the emulator setup is fast enough for demonstration purposes.

D.3. Application throughput

This study measures the maximum throughput an application can achieve with FoGSiEm without video processing overhead. It uses a native IP application for the measurements in order to use a common and reliable software for the measurements.

The study measures the throughput that can be achieved via one connection in the emulator setup described in Section D.1. It compares the maximum throughput of FoGSiEm and IP. Since FoGSiEm is a rapid prototype, which trades performance for implementation time, its performance is worse than the performance of IP. However, FoGSiEm should be fast enough for demonstration purposes as mentioned in Section 4.1. The study quantifies the performance degradation of the FoG *emulation* in comparison to IP without the video overhead of the previous study.

The IP tool Iperf version 2.0.5 [Ipe] is used for the throughput measurements[2]. The Iperf instance S is sending data to the Iperf instance R as depicted in Figure D.4. The sender side of the setup is comparable to the setup of the

[2] Jperf 2.0.2 is used as graphical front end

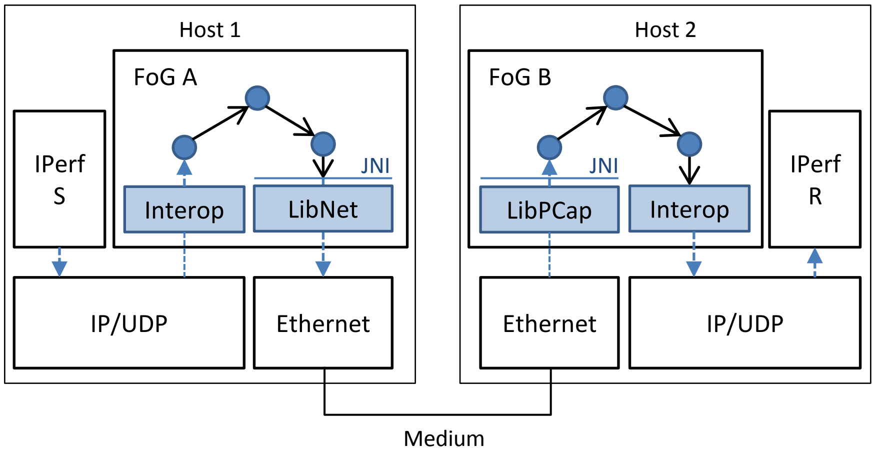

Figure D.4.: Setup for Iperf measurements. Gates are shown with arrows. Additional data flows are depicted with dashed arrows.

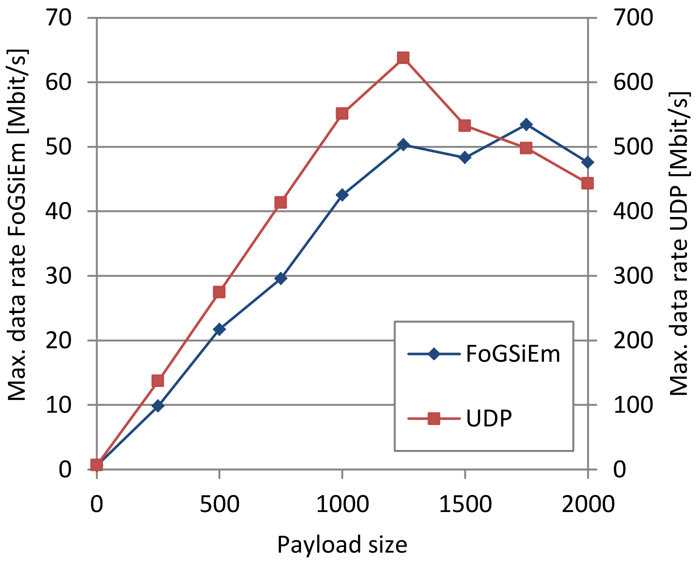

Figure D.5.: Maximum data rate of Iperf over FoGSiEm and UDP directly over Ethernet. Values measured with Iperf averaged over 30 seconds.

study described in Section D.2. The receiver side of Host 2 is similar as well. However, Host 2 contains a second interoperability gateway, which converts the FoG packets back to UDP packets and delivers them to Iperf R. Thus, the measurement includes two transitions between FoGSiEm and the Linux kernel (to and from IP and Ethernet, respectively) and one transition through JNI (from Java to C and C to Java, respectively) per notebook. Moreover, it includes the physical transmission via Ethernet.

Figure D.5 shows the average maximum data rate measured with Iperf averaged over 30 seconds. The size of the payload sent by Iperf is varied in steps of 250 bytes in order to measure the overhead per packet and the impact of the fragmentation. With a payload size of 1 byte, the overhead of processing packets reduces the data rate to 600 kbit/s. It increases to a maximum value of approximatly 50 Mbit/s. The data rate transmitted over Ethernet is significantly higher due to the encoding of FoG packets (cp. previous study). For comparion reasons, the maximum data rate with UDP/IP without FoGSiEm is depicted in Figure D.5 as well. It uses the y-axis on the right hand side and is between 9 to 14 times higher than the maximum data rate achievable with FoGSiEm.

As in the video streaming study, the maximum throughput is limited by the processing overhead and not by the data rate of the Ethernet link. The optimized implementation of UDP (in particular its implementation in the kernel) is able to achieve a much higher throughput. If FoG is implemented in the kernel as well, it would result in a much better performance. Depending on the setup, the performance might even be better than the performance of IP. The minimal processing steps required per FoG packet are discussed in Section 4.3.2.4.

E. Analysis of state distribution for FoG network

The distribution of the number of gate numbers stored on a FoG node shown in Section 5.2.2 might be normal distributed. However, the following tests reveal that this is not the case. Both tests are using values from Figure 5.8. Since these values are only shifted, the results hold for the original data shown in Figure 5.6 as well.

- Chi-squared test: The values from the experimental sample are grouped according to intervals. Each group has to have a frequency of at least 5% in order to be suitable for the test. The equidistant intervals are 0.06 broad, because this leads to a distribution with a high number of intervals fulfilling this requirement. The border "intervals" include the tails (to minus infinity and to plus infinity, respectively) in order to create a group with a frequency of at least 5%. The frequencies are shown in Figure E.1 graphically and in Table E.1 as values. Figure E.1 shows that the sample seems not to be normal distributed, because it is shifted to the left and has a longer tail on the right side. The high differences leads to a high $X_S^2 = 256.89$. With a degree of freedom $df = 12 - 1 - 2 = 9$, $P(X^2 > X_S^2) \approx 0$. According to the chi-squared test, the sample is not normal distributed.

- Jarque-Bera test [JB87]: The Jarque-Bera test checks whether skewness and kurtosis of a sample matches a normal distribution. The skewness of the sample data is 0.546 (positive skew; meaning that the right tail is longer) and the kurtosis is 3.581 (leptokurtic distributions: positive in comparison to a normal distribution). Thus, a high $JB = 319.327$ with $df_{JB} = 2$ results in $P(X^2 > JB) \approx 0$. Thus, the Jarque-Bera test also rejects the assumption that the sample is normal distributed.

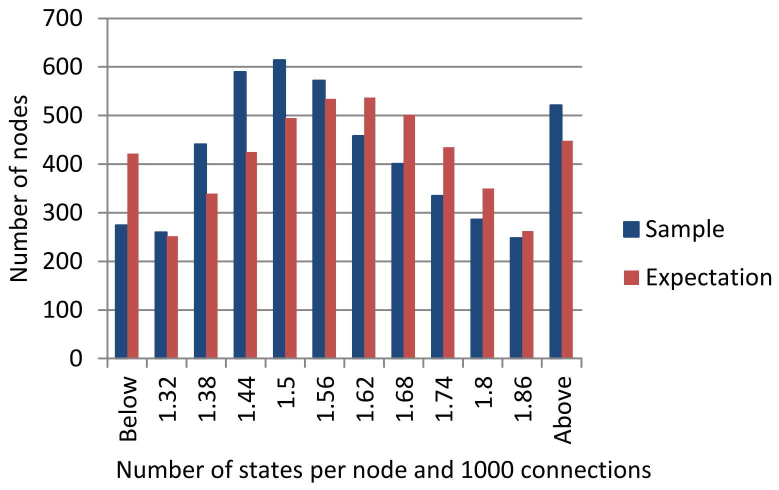

Figure E.1.: Comparison of experimental and expected frequency according to a normal distribution. X-axis shows mainly the centers of intervals. The border bars include the tails. Sample size is 5,000.

Interval center	Below	1.32	1.38	1.44	1.5	1.56	1.62	1.68	1.74	1.8	1.86	Above
Sample	274	260	441	590	614	572	458	401	335	286	248	521
Expectation	422	252	339	425	495	535	537	501	435	350	262	448
Difference	-147	8	101	164	119	37	-79	-100	-99	-64	-14	73

Table E.1.: Values for Figure E.1

F. Node degree correlation analysis

In Section 5.2.2, the relationship between the node degree and the number of mapping states for a reference architecture is visualized in Figure 5.5. The relationship between the node degree and the number of gate numbers on a node in a FoG network is shown in Figure 5.7. In the following the linear and non-linear correlation of both relationships is discussed in more detail.

The linear correlation coefficients R for both relationships are shown in the middle column of Table F.1. The first relationship shows a very strong positive linear correlation, while the second one shows a weak negative linear correlation.

In order to check for non-linear correlations, Spearman's rank correlation coefficients R_S (with ties) had been calculated as well. They are shown in the right column of Table F.1. Since they are based on ranks and not on the actual values, they only indicate whether there is a monotonic dependency. The R_S values indicate a strong correlation for the first relationship and a medium correlation for the second one. According to the test equation given in [Göh99, p. 113], the correlation values R_S are both highly significant (since $n = 5000 > 20$: $47.1 > 3.3$ and $38.3 > 3.3$, respectively).

Spearman's rank correlation coefficients confirm the tendency from the linear correlation coefficients. The first relationship seems to be really correlated according to both coefficients. However, the correlation of the second relationship is less clear. While the linear correlation is only weak, the R_S seems to indicate a non-linear correlation.

In order to analyze the correlation, Figure F.1 shows the experimental distribution functions for eight different categories of nodes. They are categorized

Node degree vs:	**R**	**R_S**
1. Number of mapping states (Reference)	0.975	0.666
2. Sum of gate numbers (FoG)	-0.348	-0.541

Table F.1.: Correlation coefficients for relationship between node degree and two other measurements

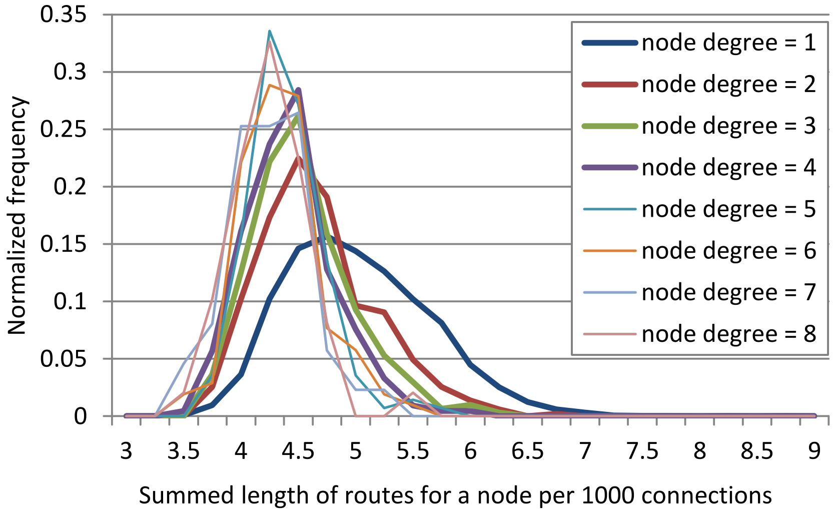

Figure F.1.: Experimental distribution functions for different node degrees. One distribution corresponds to one "vertical cut" in Figure 5.7. X-axis depicts the centers of the equidistant intervals (0.25). Each curve is normalized by the number of nodes having the same node degree.

according to their node degree. Only 9.5% of the nodes have a node degree higher than eight and are not included in the figure. The figure shows that the distribution seems to depend on the node degree: Nodes with a low node degree are more located at the "periphery" of an inter-network and, thus, have longer routes to other nodes of the inter-network. In contrast, nodes with a high node degree are more located in the "core" of an inter-network and, thus, have shorter routes to other nodes. Moreover, the distribution seems to change from a distribution with a heavy tail to a normal distribution. End users within "core" subnetworks[1] seem to have an advantage compared to end users located at the periphery. However, this assumes that the route length is not influenced by the internal structure of subnetworks. An analysis regarding the question whether this advantage is overcompensated by the larger internal structure of such "core" subnetworks is subject to future work.

[1] According to DIMES, e.g., AS7018 "ATT-INTERNET4 - AT&T Services, Inc." with a node degree of 1919, AS6680 "Deutsche Telekom AG" with a node degree of 785, and AS680 "DFN Verein zur Foerderung eines Deutschen Forschungsnetzes e.V." with a node degree of 147 would be such core subnetworks.

G. Additional simulation results

This section contains additional results or different plots of results from the simulations described in Chapter 5.

G.1. Statistical reliability of routing service graph sizes

The routing service graph sizes depend strongly on the distribution of cluster heads and, thus, of the clusters. The performance study in Section 5.3 used 20 simulation runs per probability in order to achieve a reliable average. However, the smaller the probability to become a cluster head, the higher the variations are. Especially the assignment of the core nodes with a high node degree to clusters influence the cluster size significantly (in comparison to the assignment of a node with just a single neighbor).

Originally, the simulations had been carried out three times with 10 simulation runs per probability each. Their results did not differ significantly. Since the first set of simulations had been carried out with an older version of the simulator with a slightly different statistic output format, only the last two set of simulations have been combined.

To confirm this stability, the average number of clusters (equal to number of cluster heads) has been analyzed. Figure G.1 shows the difference between the average number of clusters and the expectation value in percent of the expectation value. For 1.25%, the average value of 60.3 differs from the expectation value of 62.5 (for 5000 nodes) by -3.52%. There have been runs with 37 up to 73 clusters resulting in high percentage values for the maximum and minimum values. The higher the probability the more stable the results get.

In summary, the analysis reveals that the number of clusters remains below the expected number of clusters. The absolute error remains below 0.3% for most of the probabilities. Only higher differences at 1.25% (-3.52%), 2.5% (-1.16%), and 10% (1.30%) decrease the overall average difference to -0.33%. However, the error is rather small and does not seem to be significant for the results of the simulations.

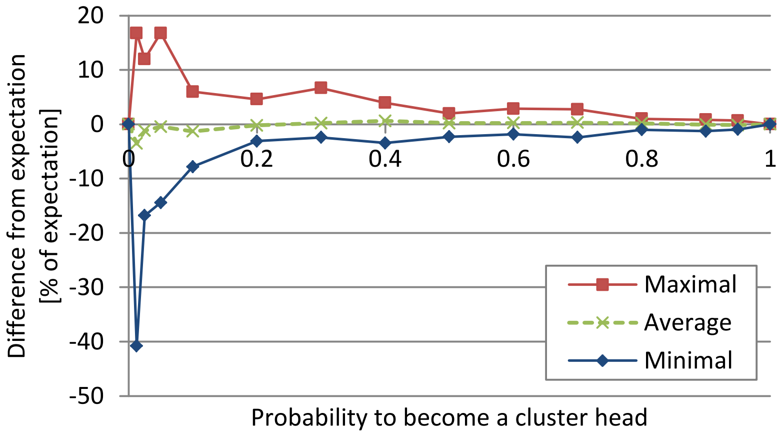

Figure G.1.: Average, maximal, and minimal number of routing service clusters compared to the expected number of clusters

G.2. Error recovery for random link failures

In Section 5.4, only the results for failing links in the middle of a route are shown. This error scheme allows repairing of 97 to 99% of the failed routes. If a random link of a route fails, the proportion of successful repaired connections in the GLP graph drops to 63%. In the remaining cases, the failure partitions the network. Due to long simulation times and the less reliable sample size, the study was not executed for the large graphs.

Figure G.2 shows the results for link failures in the middle of a route and random link failures for the GLP graph. The study contains 100,000 connections between random nodes per link-breaking strategy. The results are average values from the successfully repaired failures. Thus, the sample size for the random failures is smaller than the sample for the failures in the middle.

As expected, the reference route length did not change (cp. Section 5.2.1). The global repair algorithm can still find equally short alternative routes. Its signaling messages require 16% more hops. The local repair algorithm requires a 14% higher overhead to deal with random failures. Since this value is relative to the reference route length, the overall increase in comparison to the failures in the middle is just 3%. The from-detector algorithm requires a 56% higher overhead, which leads to an overall increase of 6%.

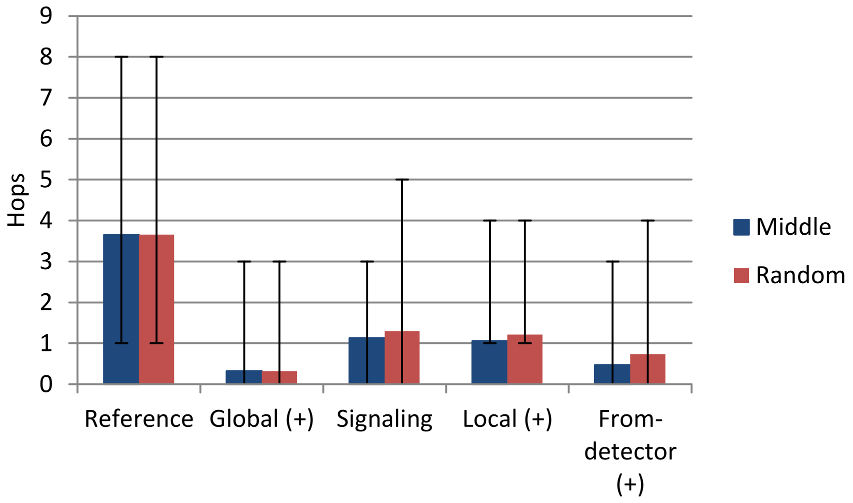

Figure G.2.: Route length resulting from repair actions in GLP graph (further explanation see Figure 5.17)

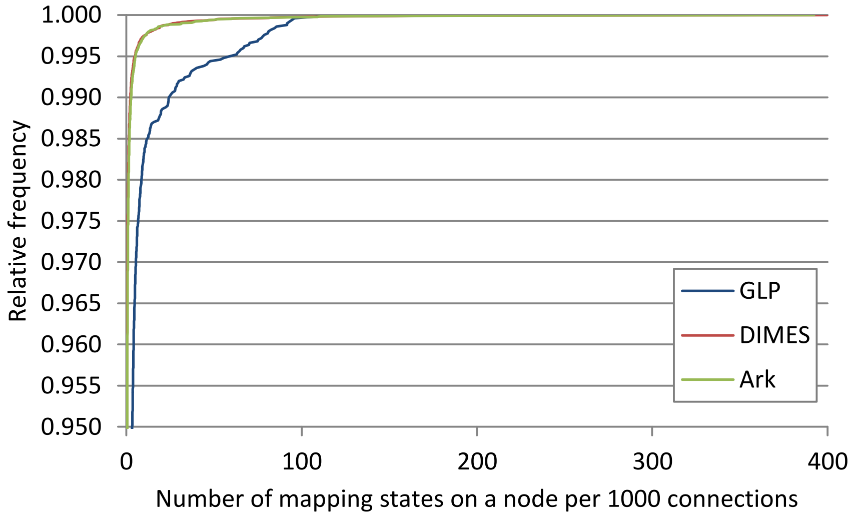

Figure G.3.: Number of mapping states as shown in Figure 5.4 with a different scale

G.3. Mapping states distribution

In addition to the figures shown in Section 5.2, Figure G.3 shows the same results with different scales for the axes.

Nomenclature

ANA	Autonomic Network Architecture
ANSI	American National Standards Institute
API	Application Programming Interface
ARP	Address Resolution Protocol
ARPA	Advanced Research Projects Agency
ATM	Asynchronous Transfer Mode
AutoBAHN	Automated Bandwidth Allocation across Heterogeneous Networks
BGP	Border Gateway Protocol
CCITT	International Telegraph and Telephone Consultative Committee
CLNP	Connectionless Network Protocol
CPU	Central Processing Unit
DDP	Datagram Delivery Protocol
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DiffServ	Differentiated Service
DNS	Domain Name System
DRUID	Dynamic Recursive Unified Internet Design
DSA	Distributed System Architecture
EFCP	Error and Flow Control Protocol
FIB	Forward Information Base
FoG	Forwarding on Gates
FoGSiEm	FoG Simulator/Emulator
GLP	Generalized Linear Preference
GMPLS	Generalized Multi-Protocol Label Switching
GUI	graphical user interface
ICMP	Internet Control Message Protocol
IDP	Information Dispatch Points
IMP	Interface Message Processor
IntServ	Integrated Service
IONL	Internal Organization of the Network Layer
IP	Internet Protocol
IPTO	Information Processing Techniques Office
IPX	Internetwork Packet eXchange
IRATI	Investigating RINA as an Alternative to TCP/IP

ISO	International Organization for Standardization
k	Kilo
LAN	Local Area Networks
LAPB	Line Access Protocol Balanced
LDP	Label Distribution Protocol
LISP	Locator/ID Separation Protocol
LLC	Logical Link Control
LSP	Label Switched Path
M	Mega
m	Milli
MAC	Medium Access Control
MCS	Modular Communication Systems
MPLS	MultiProtocol Label Switching
NAT	Network Address Translation
NCP	Network Control Protocol
NIRA	New Inter-Domain Routing Architecture
NoS	Nature of Service
NPL	National Physical Laboratory
NSIS	Next Step in Signaling
OSGi	Open Service Gateway Initiative
OSI	Open System Interconnection
OSPF	Open Shortest Path First
OvIP	Overpass IP
PCI	Protocol Control Information
PFRI	Postmodern Forwarding and Routing Infrastructure
PIP	P Internet Protocol
PUP	PARC Universal Protocol
QoS	Quality of Service
RBA	Role-based Architecture
RCP	Eclipse Rich Client Platform
RFC	Request for Comments
RIB	Routing Information Base
RIEP	Resource Information Exchange Protocol
RINA	Recursive INternet Architecture
RMI	Remote Method Invocation
RNA	Recursive Network Architecture
RSVP	Resource Reservation Protocol
s	Seconds
SAP	Service Access Point
SCORE	Scalable Core
SDH	Synchronous Digital Hierarchy
SDU	Service Data Units
SIG	Special Interest Group

SIP	Session Initiation Protocol
SIPP	Simple Internet Protocol Plus
SNA	Systems Network Architecture
SNMP	Simple Network Management Protocol
SONATE	Service Oriented Node Architecture
SSFnet	Scalable Simulator Framework
SVC	Scalable Video Codec
TCP	Transmission Control Protocol
TOS	Type of Service
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
XTP	Xpress Transfer Protocol

List of Figures

List of Tables

Listings

Index

Bibliography

[ABE$^+$04] Bengt Ahlgren, Marcus Brunner, Lars Eggert, Robert Hancock, and Stefan Schmid. Invariants: a new design methodology for network architectures. In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, FDNA '04, pages 65–70, New York, NY, USA, 2004. ACM. URL: `http://doi.acm.org/10.1145/1016707.1016719`.

[AD11] Saamer Akhshabi and Constantine Dovrolis. The evolution of layered protocol stacks leads to an hourglass-shaped architecture. *SIGCOMM Comput. Commun. Rev.*, 41(4):206–217, August 2011.

[Agu08] Rui L. Aguiar. Some comments on hourglasses. *Computer Communication Review*, 38(5):69–72, 2008.

[AMT07] L. Andersson, I. Minei, and B. Thomas. LDP Specification. RFC 5036 (Draft Standard), October 2007. Updated by RFCs 6720, 6790. URL: `http://www.ietf.org/rfc/rfc5036.txt`.

[Apa] Apache Foundation. Homepage Apache River. URL: `http://river.apache.org/` [retrieved 04/12/2013].

[App] Apple Computer Inc. Bonjour for developers. URL: `https://developer.apple.com/bonjour/` [retrieved 04/28/2013].

[App94] Apple Computer Inc. *Inside Macintosh: Networking*. Addison-Wesley Longmann, 2 edition, 1994.

[ASNN07] J. Abley, P. Savola, and G. Neville-Neil. Deprecation of Type 0 Routing Headers in IPv6. RFC 5095 (Proposed Standard), December 2007. URL: `http://www.ietf.org/rfc/rfc5095.txt`.

[BBC$^+$98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475 (Informational), December 1998. Updated by RFC 3260. URL: `http://www.ietf.org/rfc/rfc2475.txt`.

[BCG$^+$06] Bobby Bhattacharjee, Ken Calvert, Jim Griffioen, Neil Spring, and James Sterbenz. Postmodern internetwork architecture. Technical report, Universities of Maryland, Kentucky and Kansas, February 2006.

[BCS94] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational), June 1994. URL: `http://www.ietf.org/rfc/rfc1633.txt`.

[BFH03] Robert Braden, Ted Faber, and Mark Handley. From protocol stack to protocol heap: role-based architecture. *SIGCOMM Comput. Commun. Rev.*, 33(1):17–22, January 2003.

[BGT04] Tian Bu, Lixin Gao, and Don Towsley. On characterizing BGP routing table growth. *Comput. Netw.*, 45(1):45–54, May 2004. URL: `http://dx.doi.org/10.1016/j.comnet.2004.02.003`.

[BHMW11] Roland Bless, Christian Hübsch, Christoph P. Mayer, and Oliver P. Waldhorst. *Future Internet Services and Service Architectures*, chapter SpoVNet: An Architecture for Easy Creation and Deployment of Service Overlays. River Publishers, 2011.

[BIFD01] F. Baker, C. Iturralde, F. Le Faucheur, and B. Davie. Aggregation of RSVP for IPv4 and IPv6 Reservations. RFC 3175 (Proposed Standard), September 2001. Updated by RFC 5350. URL: `http://www.ietf.org/rfc/rfc3175.txt`.

[BM95] S. Bradner and A. Mankin. The Recommendation for the IP Next Generation Protocol. RFC 1752 (Proposed Standard), January 1995. URL: `http://www.ietf.org/rfc/rfc1752.txt`.

[Boc97] S. Bocking. Object-oriented network protocols. In *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, volume 3, pages 1245–1252 vol.3, 1997.

[Bra89] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (INTERNET STANDARD), October 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864. URL: `http://www.ietf.org/rfc/rfc1122.txt`.

[BSTM80] D. Boggs, J.F. Shoch, E. Taft, and R. Metcalfe. Pup: An internetwork architecture. *Communications, IEEE Transactions on*, 28(4):612–624, 1980.

[BZB$^+$97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), September 1997. Updated by RFCs 2750, 3936, 4495, 5946, 6437, 6780. URL: `http://www.ietf.org/rfc/rfc2205.txt`.

[Car96] B. Carpenter. Architectural Principles of the Internet. RFC 1958 (Informational), June 1996. Updated by RFC 3439. URL: `http://www.ietf.org/rfc/rfc1958.txt`.

[CCI70] Report on CCITT meetings on New Data Networks, 23–27 of November 1970. URL: `http://www.cs.utexas.edu/users/chris/DIGITAL_ARCHIVE/NPL/Davies11.pdf`.

[CCR+09] Vinton G. Cerf, Stephen D. Crocker, David P. Reed, Lauren Weinstein, and Daniel Lynch. Open letter to Julius Genachowski Chairman of Federal Communication Commission, October 2009. URL: `http://voices.washingtonpost.com/posttech/Net%20Pioneers%20Letter%20to%20Chairman%20Genachowski%20Oct09.pdf`.

[Cer06] Vinton G. Cerf. Prepared statement for U.S. senate committee on commerce, science, and transportation hearing on "network neutrality", February 2006. URL: `http://www.commerce.senate.gov/pdf/cerf-020706.pdf`.

[CG88] Israel Cidon and Inder S. Gopal. Paris: An approach to integrated high-speed private networks. *International Journal of Digital Analog Cabled Systems*, 1(2):77–85, April-June 1988.

[CGKR10] Lorenzo Colitti, Steinar H. Gunderson, Erik Kline, and Tiziana Refice. Evaluating ipv6 adoption in the internet. In *Proceedings of the 11th international conference on Passive and active measurement*, PAM'10, pages 141–150, Berlin, Heidelberg, 2010. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1889324.1889339`.

[CGP07] K.L. Calvert, J. Griffioen, and Leonid Poutievski. Separating routing and forwarding: A clean-slate network layer design. In *Broadband Communications, Networks and Systems, 2007. BROADNETS 2007. Fourth International Conference on*, pages 261–270, 2007.

[Che89] D. R. Cheriton. Sirpent: a high-performance internetworking approach. In *Symposium proceedings on Communications architectures & protocols*, SIGCOMM '89, pages 158–169, New York, NY, USA, 1989. ACM. URL: `http://doi.acm.org/10.1145/75246.75263`.

[Cis12] Cisco Systems. Cisco visual networking index: Forecast and methodology, 2011-2016. White paper, Cisco Systems, May 2012. URL: `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`.

[CK13] S. Cheshire and M. Krochmal. Multicast DNS. RFC 6762 (Proposed Standard), February 2013. URL: `http://www.ietf.org/rfc/rfc6762.txt`.

[Cla88] David D. Clark. The design philosophy of the DARPA internet protocols. *SIGCOMM Computer Communication Review*, 18(4):106–114, August 1988.

[Cla05] David D. Clark. What is "architecture"?, November 2005. URL: `http://find.isi.edu/presentation_files/Dave_Clark-What_is_architecture_4.pdf`.

[CPB$^+$05] David D. Clark, Craig Partridge, Robert T. Braden, Bruce Davie, Sally Floyd, Van Jacobson, Dina Katabi, Greg Minshall, K. K. Ramakrishnan, Timothy Roscoe, Ion Stoica, John Wroclawski, and Lixia Zhang. Making the world (of communications) a different place. *SIGCOMM Comput. Commun. Rev.*, 35(3):91–96, July 2005.

[CWSB02] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow's internet. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '02, pages 347–356, New York, NY, USA, 2002. ACM. URL: `http://doi.acm.org/10.1145/633025.633059`.

[Day95] John Day. The (un)revised OSI reference model. *SIGCOMM Comput. Commun. Rev.*, 25(5):39–55, October 1995. URL: `http://doi.acm.org/10.1145/216701.216704`.

[Day08a] John Day. *Patterns in Network Architecture - A Return to Fundamentals*. Prentice Hall, 2008.

[Day08b] John Day. Why loc/id split isn't the answer. Internet, 2008. URL: `http://pouzin.pnanetworks.com/images/LocIDSplit090309.pdf`.

[Day11] John Day. How in the heck do you lose a layer!? In *Network of the Future (NOF), 2011 International Conference on the*, pages 135–143, 2011.

[DEB12] Philipp Drieß, Florian Evers, and Markus Brückner. The MoSaKa QoS system: Architecture and evaluation. *International Journal On Advances in Telecommunications, vol 5, nr 3&4, 2012*, 5(3&4):216–228, September 2012.

[Dee01] Steve Deering. Watching the waist of the protocol hourglass. Talk at IETF 51, August 2001. URL: `http://www.iab.org/wp-content/IAB-uploads/2011/03/hourglass-london-ietf.pdf`.

[Deu] Deutsche Telekom GmbH. Entertain. URL: `http://www.telekom.de/privatkunden/fernsehen` [retrieved 04/12/2013].

[DF99] Christophe Deleuze and Serge Fdida. A scalable intserv architecture through rsvp aggregation. *Networking and information systems journal*, 2(5-6):665–681, 1999.

[DFV07] Mónica Domingues, Carlos Friaças, and Pedro Veiga. Is global IPv6 deployment on track? *Internet Research*, 17(5):505–518, 2007.

[DH95] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 1883 (Proposed Standard), December 1995. Obsoleted by RFC 2460. URL: `http://www.ietf.org/rfc/rfc1883.txt`.

[DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946. URL: `http://www.ietf.org/rfc/rfc2460.txt`.

[DMM08] John Day, Ibrahim Matta, and Karim Mattar. Networking is IPC: a guiding principle to a better internet. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 67:1–67:6, New York, NY, USA, 2008. ACM. URL: `http://doi.acm.org/10.1145/1544012.1544079`.

[DRB$^+$07] Rudra Dutta, George N. Rouskas, Ilia Baldine, Arnold Bragg, and Dan Stevenson. The SILO architecture for services integration, control, and optimization for the future internet. In *IEEE ICC*, pages 24–27, 2007.

[DSW09] Khaled Deeb, Sean P. O'Brien Sr., and Matthew E. Weiner. A survey on network neutrality - a new form of discrimination based on network profiling. *Int. J. Netw. Virtual Organ.*, 6(4):426–436, May 2009. URL: `http://dx.doi.org/10.1504/IJNVO.2009.025938`.

[DWJ09] J. Domzal, R. Wojcik, and A. Jajszczyk. QoS-aware net neutrality. In *Evolving Internet, 2009. INTERNET '09. First International Conference on*, pages 147–152, 2009.

[Ecl] Eclipse Foundation. Homepage Rich Client Platform. URL: `http://www.eclipse.org/home/categories/rcp.php` [retrieved 04/25/2013].

[ET12] Nicholas Economides and Joacim Tåg. Network neutrality on the internet: A two-sided market analysis. *Information Economics and Policy*, 24:91–104, August 2012.

[Fel07] Anja Feldmann. Internet clean-slate design: what and why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64, July 2007.

[FFML13] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830 (Experimental), January 2013. URL: `http://www.ietf.org/rfc/rfc6830.txt`.

[FLSS99] Merilee Ford, H. Kim Lew, Steve Spanier, and Tim Stevenson. *Internetworking Technology Overview*. Cisco Systems, Inc., June 1999. URL: `ftp://www.nwstc.noaa.gov/IT/NetBasics.pdf`.

[FoG] Homepage FoGSiEm on GitHub. URL: `https://github.com/ICS-TU-Ilmenau/fog/wiki` [retrieved 04/12/2013].

[Fra94] P. Francis. Pip Near-term Architecture. RFC 1621 (Informational), May 1994. URL: `http://www.ietf.org/rfc/rfc1621.txt`.

[FRM97] Gregory Finn, Craig Milo Rogers, and Rodney Van Meter. Datagram forwarding via stateless internetwork switching. In *IEEE Communications Society and NetWorld+Interop '97, Engineers Conference on Broadband Access - Technologies, Systems and Services*, May 1997.

[FT84] Michael L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 338–346, 1984.

[GÉA] GÉANT2. Bandwidth on demand. URL: `http://www.geant2.net/server/show/conWebDoc.1018` [retrieved 04/12/2013].

[GB04] Erich Gamma and Kent Beck. *Contributing to Eclipse: principles, patterns, and plug-ins*. Pearson Education, 2004.

[GGSS09] P. Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. *SIGCOMM Comput. Commun. Rev.*, 39(4):111–122, August 2009.

[Göh99] Wilhelm Göhler. *Formelsammlung höhere Mathematik*. Verlag Harri Deutsch, Frankfurt (Main), Germany, 14 edition, 1999.

[GRY07] Minas Gjoka, Vinayak Ram, and Xiaowei Yang. Evaluation of IP fast reroute proposals. In *IEEE COMSWARE*, 2007.

[Han06] M. Handley. Why the internet only just works. *BT Technology Journal*, 24(3):119–129, July 2006. URL: `http://dx.doi.org/10.1007/s10550-006-0084-z`.

[Hea93] Nicholas Heap. *An Introduction to OSI*. Blackwell Science Publications Ltd, Oxford, United Kindom, 1993. Deutsche Übersetzung: OSI Referenzmodell ohne Geheimnisse. Verlag Heinz Heise GmbH & CO KG, Hannover 1994.

[HFU$^+$08] Hamed Haddadi, Damien Fay, Steve Uhlig, Andrew Moore, Richard Mortier, Almerima Jamakovic, and Miguel Rio. Tuning topology generators using spectral distributions. In *Proceedings of the SPEC international workshop on Performance Evaluation: Metrics, Models and Benchmarks*, SIPEW '08, pages 154–173, Berlin, Heidelberg, 2008. Springer-Verlag. URL: `http://dx.doi.org/10.1007/978-3-540-69814-2_11`.

[HG05] Yaqing Huang and R. Guerin. Does over-provisioning become more or less efficient as networks grow larger? In *Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on*, pages 11 pp.–235, 2005.

[HHA$^+$] Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, Matthew Luckie, kc claffy, and Colleen Shannon. The IPv4 routed /24 AS links dataset - 2nd/3rd february 2012. URL: `http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml`.

[Hin94] R. Hinden. Simple Internet Protocol Plus White Paper. RFC 1710 (Informational), October 1994. URL: `http://www.ietf.org/rfc/rfc1710.txt`.

[HKLdB05] R. Hancock, G. Karagiannis, J. Loughney, and S. Van den Bosch. Next Steps in Signaling (NSIS): Framework. RFC 4080 (Informational), June 2005. URL: `http://www.ietf.org/rfc/rfc4080.txt`.

[HL03] Katie Hafner and Matthew Lyon. *Where Wizards Stay Up Late: The Origins Of The Internet*. Pocket Book, 2003.

[HMH13] M. Hoefling, M. Menth, and M. Hartmann. A survey of mapping systems for locator/identifier split internet routing. *Communications Surveys Tutorials, IEEE*, PP(99):1–17, 2013.

[Hom] Homer Conferencing. Homepage. URL: `http://www.homer-conferencing.com/en/index.html` [retrieved 04/12/2013].

[HT11] Harri Holma and Antti Toskala, editors. *LTE for UMTS: Evolution to LTE-Advanced*. John Wiley & Sons, Ltd, 2 edition, 2011.

[IAN13] IANA. Service name and transport protocol port number registry, April 2013. URL: `http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt`.

[IEE08] IEEE. Std 802.3 - 2008 part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, 2008.

[IEE13] IEEE. Organizationally unique identifier. Webpage, February 2013. URL: `http://standards.ieee.org/develop/regauth/oui/oui.txt`.

[Int86] International Organization for Standardization. Final text of DIS 8473, Protocol for Providing the Connectionless-mode Network Service. RFC 994, March 1986. URL: `http://www.ietf.org/rfc/rfc994.txt`.

[ION88] Information technology - telecommunications and information exchangebetween systems - the internal organization of the network layer, 1988. URL: `http://www.iso.org/iso/catalogue_detail.htm?csnumber=16011`.

[Ipe] Iperf. Homepage. URL: `http://sourceforge.net/projects/iperf/` [retrieved 04/12/2013].

[IRA] IRATI project. Homepage IRATI - Investigating RINA as an alternative to TCP/IP. URL: `http://irati.eu/` [retrieved 04/12/2013].

[Jac88] Van Jacobson. Congestion avoidance and control. *SIGCOMM Computer Communication Review*, 18(4):314–329, August 1988.

[JB87] Carlos M. Jarque and Anil K. Bera. A test for normality of observations and regression residuals. *International Statistical Review*, 55(2):163–172, August 1987.

[JG06] L. Jorge and T. Gomes. Survey of recovery schemes in MPLS networks. In *Dependability of Computer Systems, 2006. DepCos-RELCOMEX '06. International Conference on*, pages 110–118, 2006.

[JGKT07] Ping Ji, Zihui Ge, Jim Kurose, and Don Towsley. A comparison of hard-state and soft-state signaling protocols. *IEEE/ACM Trans. Netw.*, 15(2):281–294, April 2007. URL: `http://dx.doi.org/10.1109/TNET.2007.892849`.

[KAI04] J. Kempf, R. Austein, and IAB. The Rise of the Middle and the Future of End-to-End: Reflections on the Evolution of the Internet Architecture. RFC 3724 (Informational), March 2004. URL: `http://www.ietf.org/rfc/rfc3724.txt`.

[KHM$^+$08] A. Keller, T. Hossmann, M. May, G. Bouabene, C. Jelger, and C. Tschudin. A system architecture for evolving protocol stacks. In *Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference on*, pages 1–7, 2008.

[KMC$^+$00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, August 2000. URL: `http://doi.acm.org/10.1145/354871.354874`.

[Kob12] Hisashi Kobayashi. Keynote speech at euroview 2012. Internet blog, August 2012. URL: `http://hp.hisashikobayashi.com/keynote-speech-at-euroview-2012/`.

[KSRM12] R. Khondoker, A. Siddiqui, B. Reuther, and P. Mueller. Service orientation paradigm in future network architectures. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 346–351, 2012.

[LBMT06] Florian Liers, René Böringer, and Andreas Mitschele-Thiel. Optimal placement of anchor points within large telecommunication networks. In *Proceedings of the 4th international conference on Wired/Wireless Internet Communications*, WWIC'06, pages 96–107, Berlin, Heidelberg, 2006. Springer-Verlag. URL: `http://dx.doi.org/10.1007/11750390_9`.

[LCPM85] Barry M. Leiner, R. Cole, J. Postel, and D. Mills. The DARPA internet protocol suite. *Communications Magazine, IEEE*, 23(3):29–34, 1985.

[LHSS13] Florian Liers, Markus Hager, Sebastian Schellenberg, and Jochen Seitz. Recursive layering of forwarding on gates and traffic engineering middleware for ethernet. In *International Conference* *on Information Networking 2013 (ICOIN 2013)*, Bangkok, Thailand, January 2013.

[Liba] sam-github / libnet. Github web page. URL: `https://github.com/sam-github/libnet` [retrieved 05/27/2013].

[Libb] tcpdump & LibPCap. Project web page. URL: `http://www.tcpdump.org/` [retrieved 05/27/2013].

[LV11] Florian Liers and Thomas Volkert. Arbeitspunkt 1.5 - Spezifikation Interoperabilität. Unpublished project report for work item of project G-Lab_FoG, January 2011.

[LVM$^+$11] Florian Liers, Thomas Volkert, Denis Martin, Helge Backhaus, Hans Wippel, A. Erik Veith, Abbas Siddiqui, and Rahmatullah Khondoker. GAPI: A G-Lab application-to-network interface. In *11th Würzburg Workshop on IP: Joint ITG and Euro-NF Workshop "Visions of Future Generation Networks" (EuroView2011)*, Würzburg, Germany, August 2011.

[LVMT10] Florian Liers, Thomas Volkert, and Andreas Mitschele-Thiel. Demonstrating forwarding on gates with first applications. In *10th Würzburg Workshop on IP: Joint ITG, ITC, and Euro-NF Workshop "Visions of Future Generation Networks" (EuroView2010)*, Würzburg, August 2010.

[LVMT11] Florian Liers, Thomas Volkert, and Andreas Mitschele-Thiel. Scalable network support for application requirements with forwarding on gates. In *11th Würzburg Workshop on IP: Joint ITG and Euro-NF Workshop "Visions of Future Generation Networks" (EuroView2011)*, Würzburg Germany, August 2011.

[LVMT12] Florian Liers, Thomas Volkert, and Andreas Mitschele-Thiel. The forwarding on gates architecture: Merging intserv and diffserv. In *International Conference on Advances in Future Internet (AFIN) 2012*, Rome, Italy, August 2012.

[LVMT13] Florian Liers, Thomas Volkert, and Andreas Mitschele-Thiel. Schlussbericht G-Lab_FoG. Project report, 2013.

[MAB$^+$08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008. URL: `http://doi.acm.org/10.1145/1355734.1355746`.

[MKF+06] Priya Mahadevan, Dmitri Krioukov, Marina Fomenkov, Xenofontas Dimitropoulos, k c claffy, and Amin Vahdat. The internet AS-level topology: three data sources and one definitive metric. *SIGCOMM Comput. Commun. Rev.*, 36(1):17–26, January 2006. URL: `http://doi.acm.org/10.1145/1111322.1111328`.

[ML02] Sanjeev Mervana and Chris Le. *Design and Implementation of DSL-Based Access Solutions*. Cisco Press, Indianapolis, USA, 2002.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '01, pages 346–, Washington, DC, USA, 2001. IEEE Computer Society. URL: `http://dl.acm.org/citation.cfm?id=882459.882563`.

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=646334.687801`.

[MP02] David E. McDysan and Dave Paw. *ATM & MPLS Theory & Application: Foundations of Multi-Service Networking*. McGraw-Hill/Osborne, 2002.

[MR08] Paul Müller and Bernd Reuther. Future internet architecture - a service oriented approach (Future Internet Architecture - Ein serviceorientierter Ansatz). *it - Information Technology*, 50(6):383–389, 2008.

[MW00] Jeonghoon Mo and J. Walrand. Fair end-to-end window-based congestion control. *Networking, IEEE/ACM Transactions on*, 8(5):556–567, 2000.

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474 (Proposed Standard), December 1998. Updated by RFCs 3168, 3260. URL: `http://www.ietf.org/rfc/rfc2474.txt`.

[Ope11] Openflow switch specification, February 2011. URL: `http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf`.

[Ora] Oracle. Java object serialization specification. URL: `http://docs.oracle.com/javase/6/docs/platform/serialization/spec/serialTOC.html` [retrieved 04/12/2013].

[Osd12] Manuel Osdoba. Evaluierung eines hierarchischen routingsystems im kontext des future internet ansatzes "forwarding on gates". Master's thesis, Technische Universität Ilmenau, Ilmenau, July 2012.

[Per01] Radia Perlman. Myths, missteps, and folklore in protocol design. In *Proceedings of USENIX*, 2001.

[Per10] C. Perkins. IP Mobility Support for IPv4, Revised. RFC 5944 (Proposed Standard), November 2010. URL: `http://www.ietf.org/rfc/rfc5944.txt`.

[Pos81] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. Updated by RFCs 1349, 2474, 6864. URL: `http://www.ietf.org/rfc/rfc791.txt`.

[Pou] Pouzin Society. Homepage Pouzin Society. URL: `http://pouzin.pnanetworks.com/home.html` [retrieved 04/13/2013].

[Pou74a] Louis Pouzin. Cigale, the packet switching machine of the cyclades computer network. In *IFIP Congress*, pages 155–159, 1974.

[Pou74b] Louis Pouzin. A proposal for interconnecting packet switching networks. In *EUROCOMP*, pages 1023–1036, Brunel University, May 1974.

[PPJ11a] J. Pan, S. Paul, and R. Jain. A survey of the research on future internet architectures. *Communications Magazine, IEEE*, 49(7):26–36, 2011.

[PPJ11b] Subharthi Paul, Jianli Pan, and Raj Jain. Architectures for the future networks and the next generation internet: A survey. *Comput. Commun.*, 34(1):2–42, January 2011. URL: `http://dx.doi.org/10.1016/j.comcom.2010.08.001`.

[Pre03] Prian J. Premore. *An analysis of convergence properties of the Border Gateway Protocol using discrete event simulation*. PhD thesis, Dartmouth College, Hanover, New Hampshire, May 2003.

[Pre11] Pressestelle TU Ilmenau. TU Ilmenau präsentiert auf der CeBIT 2011 Flugroboter für Katastrophenszenarien, February 2011. URL: `http://www.pressebox.de/pressemitteilung/technische-universitaet-ilmenau/TU-Ilmenau-praesentiert-auf-der-CeBIT-2011-Flugroboter-\fuer-Katastrophenszenarien/boxid/407297`.

[RB11] Liz Ribe-Baumann. Combining resource and location awareness in DHTs. In R. Meersman, T. Dillon, and P. Herrero, editors, *OTM 2011, Part I, LNCS 7044*, pages 385–402. Springer-Verlag, 2011. URL: `http://www.dbis.prakinf.tu-ilmenau.de/publications/files/DBIS:Rib11.pdf`.

[RCCD04] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering. IPv6 Flow Label Specification. RFC 3697 (Proposed Standard), March 2004. Obsoleted by RFC 6437. URL: `http://www.ietf.org/rfc/rfc3697.txt`.

[Rei07] A. Reitzel. Deprecation of source routing options in IPv4. Internet Draft, August 2007. URL: `http://tools.ietf.org/html/draft-reitzel-ipv4-source-routing-is-evil-00`.

[RIN] Rina webpage. URL: `http://csr.bu.edu/rina/index.html` [retrieved 04/25/2013].

[Ros90] Marshall T. Rose. *The open book: a practical perspective on OSI*. Prentice Hall, Englewood Cliffs, NJ, 1990.

[RTF$^+$01] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta. MPLS Label Stack Encoding. RFC 3032 (Proposed Standard), January 2001. Updated by RFCs 3443, 4182, 5332, 3270, 5129, 5462, 5586. URL: `http://www.ietf.org/rfc/rfc3032.txt`.

[Sal82] Jerome H. Saltzer. On the naming and binding of network destinations. In *Local Computer Networks*, pages 311–317, Amsterdam, 1982. Also published as RFC 1498.

[SDW92] W. Timothy Strayer, Bert J. Dempsey, and Alfred C. Weaver. *XTP: The Xpress Transfer Protocol*. Addison-Wesley Pub, August 1992.

[SKM12] A. Siddiqui, R. Khondoker, and P. Muller. Template based composition for requirements based network stacks. In *Telecommunication Networks and Applications Conference (ATNAC), 2012 Australasian*, pages 1–6, 2012.

[SM05] V. Srivastava and M. Motani. Cross-layer design: a survey and the road ahead. *Communications Magazine, IEEE*, 43(12):112–119, 2005.

[SM12] A.A. Siddiqui and P. Mueller. A requirement-based socket API for a transition to future internet architectures. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 340–345, 2012.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984. URL: `http://doi.acm.org/10.1145/357401.357402`.

[SS05] Yuval Shavitt and Eran Shir. DIMES: let the internet measure itself. *SIGCOMM Comput. Commun. Rev.*, 35(5):71–74, October 2005.

[SS06] B. Sardar and D. Saha. A survey of TCP enhancements for last-hop wireless networks. *Communications Surveys Tutorials, IEEE*, 8(3):20–34, 2006.

[SSBK03] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H. Katz. OverQoS: offering internet QoS using overlays. *SIGCOMM Comput. Commun. Rev.*, 33(1):11–16, January 2003.

[SSF] SSF Research Network. Homepage Scalable Simulation Framework (SSF). URL: `http://www.ssfnet.org/homePage.html` [retrieved 04/12/2013].

[Sta93] William Stallings. *Networking Standards - A Guide to OSI, ISDN, LAN and MAN Standards*. Addison-Wesley, 1993.

[SZ99] Ion Stoica and Hui Zhang. Providing guaranteed services without per flow management. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 81–94, New York, NY, USA, 1999. ACM. URL: `http://doi.acm.org/10.1145/316188.316208`.

[Tan03] Andrew S. Tanenbaum. *Computer Networks*, volume 4. Pearson Education Inc., 2003.

[TBD$^+$11] Joe Touch, Ilia Baldine, Rudra Dutta, Gregory G. Finn, Bryan Ford, Scott Jordan, Dan Massey, Abraham Matta, Christos Papadopoulos, Peter Reiher, and George Rouskas. A dynamic recursive unified internet design (DRUID). *Comput. Netw.*, 55(4):919–935, March 2011.

[TG01] Christian Tschudin and Richard Gold. Selnet: A translating underlay network. Technical Report 2003-020, Uppsala University, Uppsala, Sweden, November 2001.

[TGD$^+$11] Eleni Trouva, Eduard Grasa, John Day, Ibrahim Matta, Lubomir T. Chitkushev, Steve Bunch, Miguel Ponce de Leon, Patrick Phelan, and Xavier Hesselbach-Serra. Transport over heterogeneous networks using the RINA architecture. In *Proceedings of the 9th IFIP TC 6 international conference on Wired/wireless internet communications*, WWIC'11, pages 297–308, Berlin, Heidelberg, 2011. Springer-Verlag.

[TP08] D. Joseph Touch and Venkata K. Pingali. The RNA metaprotocol. In *Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference on*, pages 1–6, 2008.

[Tro09] Dirk Trossen. Invigorating the future internet debate. *SIGCOMM Comput. Commun. Rev.*, 39(5):44–51, October 2009. URL: `http://doi.acm.org/10.1145/1629607.1629617`.

[TSS+97] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35:80–86, 1997.

[Tsu92] Paul F. Tsuchiya. Pip: The 'P' internet protocol. Internet, May 1992. work in progress.

[TW11] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*, volume 5. Pearson Education Inc., 2011.

[VL12] Thomas Volkert and Florian Liers. Video transcoding and rerouting in forwarding on gates networks. In *12th Würzburg Workshop on IP: ITG Workshop "Visions of Future Generation Networks" (EuroView)*, Würzburg Germany, August 2012.

[VLBA12] Thomas Volkert, Florian Liers, Martin Becke, and Hakim Adhari. Requirements-oriented path selection for multipath transmission. In *12th Würzburg Workshop on IP: ITG Workshop "Visions of Future Generation Networks" (EuroView)*, Würzburg Germany, August 2012.

[VMEK+09] L. Volker, D. Martin, I. El Khayaut, C. Werle, and M. Zitterbart. A node architecture for 1000 future networks. In *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*, pages 1–5, 2009.

[VMT12] Thomas Volkert and Andreas Mitschele-Thiel. Hierarchical routing management for improving multimedia transmissions and QoE. In *Proceedings of 13th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, San Francisco, California, USA, June 2012.

[VMW$^+$09] L. Völker, D. Martin, C. Werle, M. Zitterbart, and I. Khayat. Selecting concurrent network architectures at runtime. In *Proceedings of the IEEE International Conference on Communications (ICC 2009)*. IEEE, June 2009.

[VOBMT13] Thomas Volkert, Manuel Osdoba, Martin Becke, and Andreas Mitschele-Thiel. Multipath video streaming based on hierarchical routing management. In *Proceedings of 27th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Barcelona, Spain, March 2013.

[VPMK04] D. Vali, S. Paskalis, L. Merakos, and A. Kaloxylos. A survey of internet QoS signaling. *Communications Surveys Tutorials, IEEE*, 6(4):32–43, 2004.

[VTRB97] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC 2136 (Proposed Standard), April 1997. Updated by RFCs 3007, 4035, 4033, 4034. URL: `http://www.ietf.org/rfc/rfc2136.txt`.

[WCBH$^+$08] Oliver P. Waldhorst, Christian C. Blankenhorn, Dirk Haage, Ralph Holz, Gerald G. Koch, Boris Koldehofe, Fleming Lampi, Christoph P. Mayer, and Sebastian Mies. Spontaneous Virtual Networks: On the road towards the internet's next generation. *it - Information Technology Special Issue on Next Generation Internet*, 50(6):367–375, December 2008.

[Weg10] Bettina Wegner. CeBIT: TU Ilmenau treibt Internet der Zukunft voran, February 2010. URL: `http://idw-online.de/pages/de/news357291`.

[WHKL08] Gerd Wütherich, Nils Hartmann, Bernd Kolb, and Matthias Lübken. *Die OSGi Service Platform: Eine Einführung mit Eclipse Equinox*. dpunkt Verlag, 2008.

[YCB07] Xiaowei Yang, D.. Clark, and A.W. Berger. NIRA: A new inter-domain routing architecture. *Networking, IEEE/ACM Transactions on*, 15(4):775–788, 2007.

[Zit08] Jonathan L. Zittrain. *The Future of the Internet and How to Stop It*. Yale University Press, New Haven & London, 2008. URL: `http://futureoftheinternet.org/static/ZittrainTheFutureoftheInternet.pdf`.