Resource and Location Aware Robust, Decentralized Data Management

Dissertation
zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

vorgelegt der
Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von
Dipl.-Math. Elizabeth Ribe-Baumann

Tag der Einreichung: 16.Oktober 2013
Tag der wissenschaftlichen Aussprache: 18.November 2014

Gutachter

1. Prof. Dr.–Ing. Kai-Uwe Sattler
2. Prof. Dr. rer. nat. Jochen Seitz
3. Prof. Dr. Manfred Hauswirth

urn:nbn:de:gbv:ilm1-2014000449

Abstract

Increasingly, large amounts of data are being stored in a distributed manner over wide area networks. Such large scale networks most often employ heterogeneous nodes, with heterogeneity taking the form of available battery power, bandwidth, computing power, and/or network up-times, among other things. As smartphones have become an ubiquitous element of every-day life, the potential for large scale peer-to-peer networks with strong contrasts in peers' capabilities has become undeniable. This scenario is especially relevant for disaster scenarios when access to existing data storage may be unavailable or inaccessible for the broad population, although important information about conditions, hazards, or injured people could be collected by general civilians. But networks built on a mix of smartphones, laptops, and servers are not the only heterogeneous systems with the potential to distributively store large amounts of data: sensor networks and computing clouds also contain nodes with varying characteristics. However, at the core of these distributed, heterogeneous networks is a necessity to accommodate nodes' varying accessibility to resources and to reduce network load by providing short paths between where data is stored and needed, while ensuring a high level of data robustness, or availability. In fact, this resource and location awareness should be used to *increase* robustness.

This work thus focuses on resource and location awareness for robust data management in decentralized, potentially peer-to-peer, networks. In order to facilitate the assessment and comparison of resource and location aware approaches, taxonomies are developed for the classification of how resources and proximity are handled. In addition to robustness, resource awareness, and location awareness, four further requirements are derived from this work's use case scenarios: self-organization, scalability, load balancing, and data consistency. A structured network approach was chosen in order to provide availability guarantees for stored data.

The lack of structured approaches with both resource and location awareness led to the development of two novel distributed hash tables (DHTs) *Resource Based Finger Management* (RBFM) and *Hierarchical Resource Management* (HRM), which are more and less loosely based on the existing DHT Chord and have the same $O(\log N)$ routing complexity. These two DHTs take fundamentally different structural approaches to building an overlay, such that the flat RBFM and the multi-tiered hierarchical HRM provide a foundation with which to examine the suitability of flat vs. hierarchical overlay structures for resource and location awareness. Moreover, HRM uses a variable number of hierarchical layers, facilitating also a comparisons of varying numbers of hierarchy layers. An additional hybrid version of both DHTs as well as a cluster-based version of RBFM for ad hoc networks are also described and used for simulative comparisons. Mathematical analysis and simulative evaluations of the developed DHTs in comparison with naive and location aware approaches demonstrate how resource awareness improve both node lifetimes and lookup success rates twofold when resources are correlated with node failure probabilities. However, a higher number of hierarchy layers causes an increase in overall routing and maintenance load, thus decreasing node lifetimes, so that a lower number of hierarchy layers is beneficial.

DHTs require data replication to ensure that data is not lost when nodes unexpectedly leave the network, which is especially important in the high churn scenarios considered in this work. Thus, a replication technique is tailored to the resource and location aware structure of the proposed DHTs, providing both an increase in resource and location awareness as well as a reduction in the overall replication load. Mathematical analysis demonstrates how the number of replicas necessary to provide a given availability probability is significantly reduced while the remaining replica load is transferred from weak to strong nodes.

Zusammenfassung

Große Datenmengen werden zunehmend auf weite Netze, die oft aus heterogenen Knoten bestehen, verteilt. Dabei kann Heterogenität beispielsweise variable Batteriekapazität, Bandbreite, Rechenleistung oder auch Lebenszeiten beudeuten. Die weite Verbreitung von Smartphones im Alltag birgt ein Potenzial für große Peer-to-Peer Netzwerke, in dem Knoten stark variierende Leistungsfähigkeiten aufweisen. Dieses Szenario ist besonders für Katastrophen-Szenarien relevant, wenn der Zugang zu bereits existierenden Datenspeicherungsmöglichkeiten entweder unerreichbar oder unzugänglich für die Mehrheit der Bevölkerung ist. Gerade in einem Katastrophen-Szenario sammeln Zivilisten allein durch ihren Aufenthalt an weit verteilten Orten ständig eventuell wichtige Information über Zustände, Gefahren und verletzte Personen, die für weitere Verwendung und Koordinierung gesammelt werden sollten. Allerdings sind Netzwerke, die auf Smartphones, Laptops und Servern basieren, nicht die einzigen heterogenen Netzwerke, die große Datenmengen verwalten: Sensornetzwerke und Clouds bestehen ebenfalls aus Knoten mit variierenden Eigenschaften. Im Kern all dieser verteilen heterogenen Netzwerken besteht die Notwendigkeit, den variierenden Zugang der Knoten zu Ressourcen zu berücksichtigen und die Gesamtlast durch kurze Wege zwischen den Speicherungsorten und Anfrageorten der Daten möglichst zu minimieren. All dies ist allerdings einer hohen Verfügbarkeit der Daten unterstellt und es sollte sogar angestrebt werden die Verfügbarkeit der Daten gerade *durch* das Ressourcen- und Lokationsbewusstsein zu verbessern.

Die hier vorgestellte Arbeit konzentriert sich auf die Integration von Ressourcen- und Lokationsinformationen für eine robuste Datenverwaltung in verteilten, vielleicht sogar Peer-to-Peer, Netzwerken. Um verschiedene ressourcen- und lokationsbewusste Ansätze auszuwerten und zu vergleichen, wurden Taxonomien zur Klassifizierung der Nutzung von Ressourcen und Entfernungen entwickelt. Über Robustheit, Ressourcenbewusstsein und Lokationsbewusstsein hinaus wurden vier weitere Anforderungen für die zentralen betrachteten Anwendungsfälle hergeleitet: Selbstorganisation, Skalierbarkeit, Lastbalancierung und Datenkonsistenz. Ein strukturierter Netzwerkansatz wurde gewählt, um Verfügbarkeitsgarantien für die gespeicherten Daten bieten zu können.

Ein Mangel an strukturierten Ansätzen, die sowohl Ressourcen- als auch Lokationsbewusstsein nutzen, führte zu der Entwicklung von zwei neuen verteilten Hashtabellen (weiter DHT genannt): *Resource Based Finger Management* (RBFM) und *Hierarchical Resource Management* (HRM), die mehr oder weniger auf der existierenden Chord DHT aufbauen und ihre $O(\log N)$ Routingkomplexität beibehalten. Diese zwei DHTs nutzen grundsätzlich unterschiedliche Ansätze für den Aufbau des Overlays. So bilden die flache RBFM und die hierarchische HRM eine Grundlage, mit der die Eignung von flachen und hierarchischen Overlaystrukturen für Ressourcen- als auch Lokationsbewusstsein verglichen werden kann. Darüber hinaus verwendet HRM eine variable Anzahl an Hierarchieebenen, womit ein Vergleich zwischen verschiedenen Ebenenanzahlen ermöglicht wird. Weiterhin werden eine zusätzliche hybride Version von beiden DHTs sowie eine cluster-basierte Version von RBFM für ad hoc Netzwerke beschrieben und für simulationsbasierte Vergleiche benutzt. Eine mathematische Analyse und simulationsbasierte Evaluation der entwickelten DHTs zeigen, wie Ressourcenbewusstsein sowohl die Lebensdauer der Knoten als auch die Erfolgsrate der Anfragen gegenüber einem völlig naiven Ansatz und einen nur lokationsbewussten Ansatz um das zweifache erhöht, wenn angenommen wird, dass die Ausfallwahrscheinlichkeit der Knoten mit den Ressourcen der Knoten korreliert. Allerdings wurde festgestellt, dass eine hohe Anzahl von Hierarchieebenen die gesamte Routing- und Wartungslast ebenfalls erhöht und somit die Lebensdauer der Knoten reduziert, sodass eine niedrige Ebenenanzahl von Vorteil ist.

Um zu versichern, dass Daten nicht verloren gehen, wenn Knoten unerwartet das Netzwerk verlassen, benötigen DHTs Replikation. Deshalb wurde eine Replikationsstrategie entwickelt, welche die resourcen- und lokationsbewusste Struktur der vorgestellten DHTs ausnutzt. Diese Strategie erhöht somit das Ressourcen- und Lokationsbewusstsein während die Gesamtlast für der Replikation verringert wird. Eine mathematische Analyse zeigt, wie die Anzahl der benötigten Replikate signifikant gesenkt werden kann um eine vorgegebene Verfügbarkeitswahrscheinlichkeit zu erreichen, während die übrige Ressourcenlast von den schwachen auf den starken Knoten verschoben wird.

Acknowledgements

I'd like to thank first and foremost Stephan Baumann for providing me with hurdles that have encouraged me to grow in ways that I never thought possible, believing that I would find my way when I couldn't see the end of the tunnel, and shouldering triple the load when I could no longer carry my own weight. Although they do not yet understand their sacrifice, I am indescribably indebted to Lukas and Niklas Baumann for their patience, interest, and kind words as I struggled to find the time and energy for them. And a special thank you is due to Prof. Kai-Uwe Sattler, who not only believed that I could finish this dissertation long after I had given up but also shared his awe inspiring refusal to abandon projects or people in whom he has invested time and effort.

And of course, I'd like to thank my family and friends who have been, at times insanely, supportive of both me and my progress on this dissertation. I am aware that my dissertation ambitions would have perished before a single word had been brought to paper without my colleagues and fellow PhD-students-in-suffering, and I am grateful for the solidarity they emanated in excited exchanges of ideas, tedious labor on projects, and whinny banter about the hopeless state of of our work.

Contents

List of Tables

List of Figures

Nomenclature

General Terminology

layer	position within a hierarchy, $\in \{0, 1, \ldots, h_{max}\}$
level	differentiation between peers' resource strengths, $\in \{0, 1, \ldots, l_{max}\}$
bottom layer	hierarchy layer 0
lower layer	hierarchy layers $\{0, \ldots, h_{max} - 1\}$
top layer	hierarchy layer h_{max}
upper layer	hierarchy layers $\{1, \ldots, h_{max}\}$

DHT: Variables and Constants

$B_{x,i}$	node x's i^{th} finger interval $[x_{ID} + 2^{i-1}, x_{ID} + 2^{i})$
c	stretch constant for resource height function $h(x_R) := c \cdot (l_{max} - x_R)$
$d_{key}(x, y)$	clockwise key distance of x and y
$d_{phy}(x, y)$	physical distance of x and y
$d_{res}(x, y)$	resource distance between x and y: $d_{res}(x, y) = d_{phy}(x, y) + x_h + y_h$
$g(\ell)$	finger maintenance interval function
$h(x_R)$	resource height function $h : \{0, 1, \ldots, l_{max}\} \to \mathbb{R}^+$
h_{max}	maximum hierarchy layer
h_{num}	number of hierarchy layers $h_{max} + 1$
κ	denotes a key as needed
k	number of entries stored in prospective links lists per finger interval
k_i	expected number of nodes in finger interval $B_{x,i}$
ℓ	resource level $\in \{0, 1, \ldots, l_{max}\}$ or hierarchy layer $\in \{0, 1, \ldots, h_{max}\}$
l_{max}	maximum resource level
m	size of the binary key: 2^m
N	number of network nodes
ρ	power used for Zipf distributions
p_ℓ	probability that a random node has resource level ℓ. $P(x_R = \ell)$
r	number of entries stored in a node's successor list

x_{ID}	key of node x
x_H	hierarchy layer of node x
x_h	resource height $x_h = h(x_R)$
x_R	resource level of node x
$x.Fkey$	finger range key: first key in finger interval $x.Frange$
$x.Frange$	finger range: farthest finger interval in which x maintains finger
$x.F[i]$	x's layer finger in $B_{x,i}$ (layer x_H)
$x.F[i].node$	node associated with $x.F[i]$
$x.I.closestHigher$	inter-layer link with smallest $d_{key}(x, x.I[\ell])$, layer$> x_H$
$x.I.closestInt$	finger interval $B_{x,i}$ containing $x.I.closestHigher$
$x.I.closestLayer$	$x.I.closestHigher$'s hierarchy layer
$x.I[\ell]$	x's layer-ℓ inter-layer link (direct successor in layer ℓ)
$x.I[\ell].node$	node associated with $x.I[\ell]$
$x.srange$	simple key range $(y_{ID}, x_{ID}]$ with predecessor y
$x.urange$	upper key range $[x_{ID}, z_{ID})$ with upper layer successor z

Replication: Variables and Constants

ε-replicas	replicas chosen to fulfill availability probability requirement
local-replicas	replicas additionally chosen to ensure local copies by querying nodes
d	data object
ε	target availability probability
$\Lambda(d)$	set of nodes which have initiated a lookup for d
$\Sigma(d)$	set of nodes on which data object d is replicated
$input(d)$	node which initiated put request for d
$owner(d)$	node responsible for d's key
π	proximity constant - determines local fraction of network
ϕ	number of ε-replicas
$\phi_{min,\ell}$	for HRM, minimum number of ε-replicas for $owner(d)_R = \ell$ that fulfill availability requirement
ϕ_{min}	for RBFM, minimum number of ε-replicas that fulfill availability requirement
r	estimated radius of network

Approach Abbreviations

HRM2:0-123 w/RBFM:L90	HRM2:0-123 which uses RBFM in the upper hierarchy layer
HRM	Hierarchical Resource Management - novel, hierarchical resource and location aware DHT

HRM2:0-123 .... HRM with $h_{num} = 2$ and res.level 0 nodes in hier.layer 0

HRM2:01-23 .... HRM with $h_{num} = 2$ and res.level 0 1 nodes in hier.layer 0

HRM4 .... HRM with $h_{num} = 4$

RBFM .... Resource Based Finger Management - novel flat location and resource aware DHT

RBFM:Q90 .... flat location and resource aware DHT, quadratic maintenance patterns, stretch factor 90

Chapter 1

Introduction

As the world becomes increasingly digitally archived, the enormous quantity of digital information generated and stored every day also increases. Information generated by companies, government offices, academic organizations, sensor networks, web surveillance, media production and exchange, social networks, and everything web 2.0 must be stored in a highly scalable fashion that accommodates its incredibly fast generation. Much of this data is stored in private data centers or on rented space in data clouds, with attention paid to high availability and low cost, among other things. Yet, decentralized data management schemes have become important for managing huge amounts of data without the bottlenecks associated with central coordination and for situations in which the central coordination of data is unfeasible or undesirable, due to either a lack of connectivity, monetary means, or trust. Peer-to-peer ad hoc networks, for example, in which peers can only communicate on a local level to peers within wireless communication range, cannot store gathered information on a central server to which they do not have communication access. Likewise, a network of users with local storage capacities may be interested in saving large amounts of data but unwilling to pay for outsourced storage. Or a network of sensors might have access to large scale storage, but not be willing to save sensitive or unfiltered data on an external central system.

However, these decentralized solutions are often geared towards heterogeneous networks where nodes have varying access to resources such as energy, computing power, or bandwidth. This work concentrates on methods for ensuring the robustness of highly scalable *distributed* storage of data on heterogeneous networks where nodes have varying availability to resources. Here, robustness is understood as high data availability which is affected by many factors such as the system's self-organization, scalability, load balancing, data consistency, resource awareness, and location awareness. While many of the use cases motivating this work employ heterogeneous (mobile) peer-to-peer networks which offer large, unharnessed storage capacities, this work's central goal to increase robustness with minimal resource usage also caters to the needs of other scenarios such as energy-efficient storage within a cloud.

1.1 Motivation

Consider a heterogeneous network of nodes on potentially mobile devices that would like to store a large amount of data but either lack the required communication to a central server, the means with which to pay for outsourced data storage, or the willingness to store data on an external system. These nodes could, for example, comprise a sensor network having strongly restricted battery and computing power, or an ad hoc network of mobile users after a disaster with restricted battery power and bandwidth. Alternatively, the nodes could compose a cloud responsible for storing a large amount of data within a data center, with restricted bandwidth and varying available computing powers. As mobile devices have become more prevalent, their users are both generating more data and better utilizing their computing capacities. While mobile applications for smartphones are now just as much a part of day-to-day life as the wireless internet connections that make interactive applications possible, the mobile devices' collective storage potential is yet to be harnessed. Yet, it is easy to imagine that, should it suddenly become necessary for large amounts of data to be stored in an ad hoc fashion, these smartphones and other mobile devices might join forces to provide a source of highly scalable, distributed data storage.

However, a distributed data system also comes with additional costs for maintenance, data updates, and queries:

- nodes must establish and keep track of their links and routing paths within the network,
- nodes must migrate data when nodes join and leave the networks,
- the correct node must be found for each data update, and when replicas are present, multiple nodes are involved in the process,
- when replicas are present, nodes must coordinate the consistency of queried data, and
- queries must be routed to the owner(s) of the queried data.

These additional costs cause higher use of mobile devices' varying resources such as battery power, bandwidth, and computing power. Resulting overburdened nodes can lead to node inaccessibility, high delay times, and node failures, which in turn lead to lower data availability, higher data loss, and lower cumulative storage space, i.e. lower *robustness*. We might say that the system is only as reliable as its weakest node's ability to complete its assigned tasks. The reliability and applicability of such a system depends strongly on its ability to allocate load in a resource sensitive manner by shifting load to nodes with stronger resources.

On the other hand, location awareness is also necessary in a mobile network that might depend on ad hoc routing for communication. Long distance hops require more energy than short hops, and long distance paths require multiple hops between end nodes, using more energy, bandwidth, *and* computing power. Thus, long distance searches inevitably use more resources than short distance searches. By placing data closer to where it is actually needed and routing queries on shorter, more efficient paths, unnecessary resource usage, network load, and delays could be avoided.

So while resource awareness leads to a more balanced usage of resources, location awareness leads to an overall conservative usage of resources. Applied together, they offer the potential to overcome the challenges facing mobile devices due to the additional costs associated with maintaining a distributed data system.

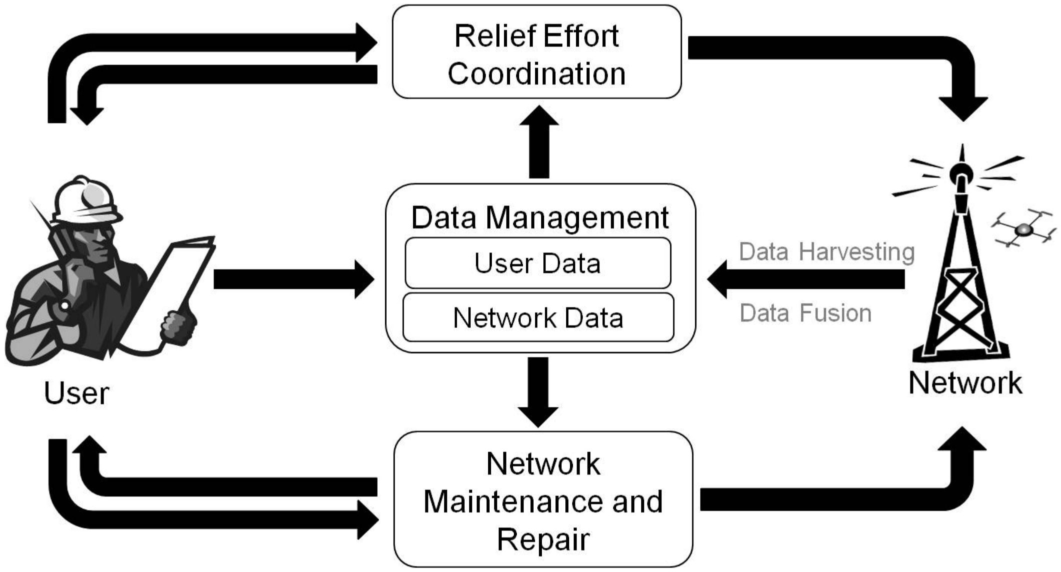


Figure 1.1: In a disaster scenario, data may be generated and used by both human users and automated network processes.

1.2 Use Cases

The task of balancing load in a distributed data system based on nodes' resources is by no means restricted to scenarios with mobile devices. The following outlines four plausible scenarios which could all reap benefits from resource and location awareness while demonstrating heterogeneity in the form of computing power, available bandwidth, power usage, available software, or system uptimes. The central use case for this work is of a heterogeneous network of mobile devices in a disaster scenario and is described first and in most detail. This use case targets a simple distributed data management application in which each participating node can enter data into and lookup data in the system. The following three use case scenarios are based on today's heterogeneous world wide web, on which web 2.0 applications requiring data storage are built; sensor networks, which store filtered sensor data within the network for later processing; and a heterogeneous cloud network, in which nodes know and trust one another while maintaining data with equal interest. Thus, each use case scenario presented uses a different network constellation with a specific application.

1.2.1 Disaster Scenario

During the response and recovery phases after a disaster, various working levels produce and require an immense amount of information. The most visible layer is formed by the front-end users, or helpers, who are ideally equipped with mobile devices that serve their communication and information needs. The services that these helpers use access and provide global information about, for example, participants, environmental states, or routing. In order to ensure efficient disaster management, this data must be reliably available to a quickly growing group of front-end users, despite possible breaks in network links or node failures due to, for example, power loss or mobile devices that have moved out of broadcast range. Users may even be

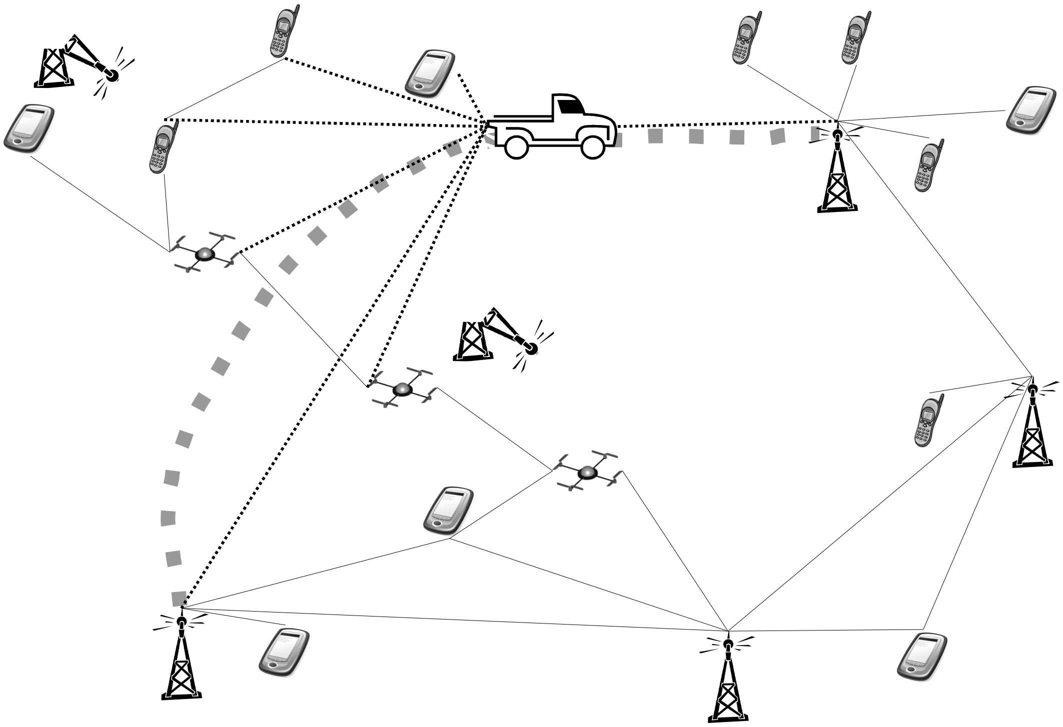


Figure 1.2: Example of a disaster scenario in which varying actors are responsible for collecting, storing, and routing data. Wireless connections are shown as lines, and the truck's changing connections as it passes through its trajectory are shown dashed.

employing a delay-tolerant network for joining network partitions, with mobile users transporting messages between partitions or single users outside of communication range [BRB10]. In this case, placing copies of data physically close to where it is needed could prevent the use of the delay tolerant network when queried data can be found within the local partition (see Figure 1.3). In any case, the information management system must be designed robustly for this highly dynamic situation.

Further down, a network of base stations enables wireless communication between front-end users, the information management system, and the outer world. This base station network is also highly dynamic and its management requires similarly reliable storage, retrieval, and aggregation of real-time data to accommodate tasks such as spectrum sensing and spectrum aware routing for cognitive radios. Imagine, for example, that the information management system is providing crucial routing information for base stations and users (e.g. for spectrum aware routing or as a DNS server). Clearly, if this information cannot be retrieved as needed, then communication fails.

On the other hand, civilians with personal mobile devices are constantly gathering first-hand information about the current situation. If this data could be saved digitally in a reliable, distributed fashion among the available private mobile and stationary devices, it could potentially be used to help find hurt or needy people; identify blocked roads, collapsed buildings, and broken utilities lines; locate hospitals with capacities; or assign displaced citizens to shelters. While the effective use of such data is a topic in and of itself, which is not approached here, the sheer feasibility of its storage is a first step in that direction.

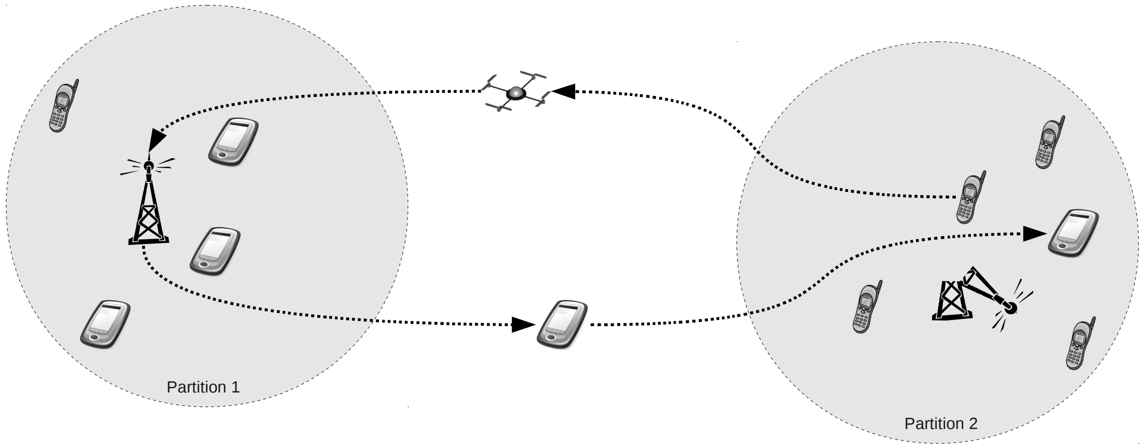


Figure 1.3: Two network partitions are joined to a single delay tolerant network with the help of users (a quadrocopter and a PDA) that move between the partitions. Information updates and retrievals can thus be performed but with longer delays.

Figure 1.2 demonstrates how such a network with varying participants might look in a disaster scenario. In this example, all nodes (i.e. cell phones, PDAs, base stations, quadrocopters, and the truck) are jointly responsible for collecting, storing, and routing data. Cell phones are depicted for civilians and PDAs for recovery workers. The two broken base stations' coverage is being compensated for by using mobile base stations, both statically placed (quadrocopters) and constantly moving along a trajectory (truck). Data is routed along wireless links, some of which (dashed) are only established for short periods of time. While all nodes collect and jointly store information, they have varying availability to resources such as power and bandwidth and must deal with a changing, perhaps partitioned, network.

A single break in the data lifecycle as shown in Figure 1.1 can debilitate the cumulative management efforts. A disaster scenario with its broken power and communication infrastructures has its own special challenges with regard to resource and location awareness, whether data is being stored and retrieved for relief workers, the underlying network infrastructure, or civilian observations:

Strongly restricted energy: Since power supply infrastructures may be broken, nodes can be expected to have very low energy availability. Nodes that do have access to power sources may lose it at regular intervals, resulting in long-term node failures once their energy has been depleted. Since this potentially holds for the entire network, the more energy that is collectively used, the less storage space is collectively available.

Strongly restricted bandwidth: With communication infrastructures damaged, bandwidth is either strongly reduced or nodes become reliant on an ad hoc routing network. Either way, the network has very high relative load and quickly becomes overburdened, leading to long delays and dropped packages. Reducing the communication overhead on the underlay is thus a central challenge that depends strongly on location awareness in ad hoc networks.

Location-sensivite data: Much data is location specific, and more likely to be inserted into the system or queried from nodes in a specific area. Storing and replicating this data

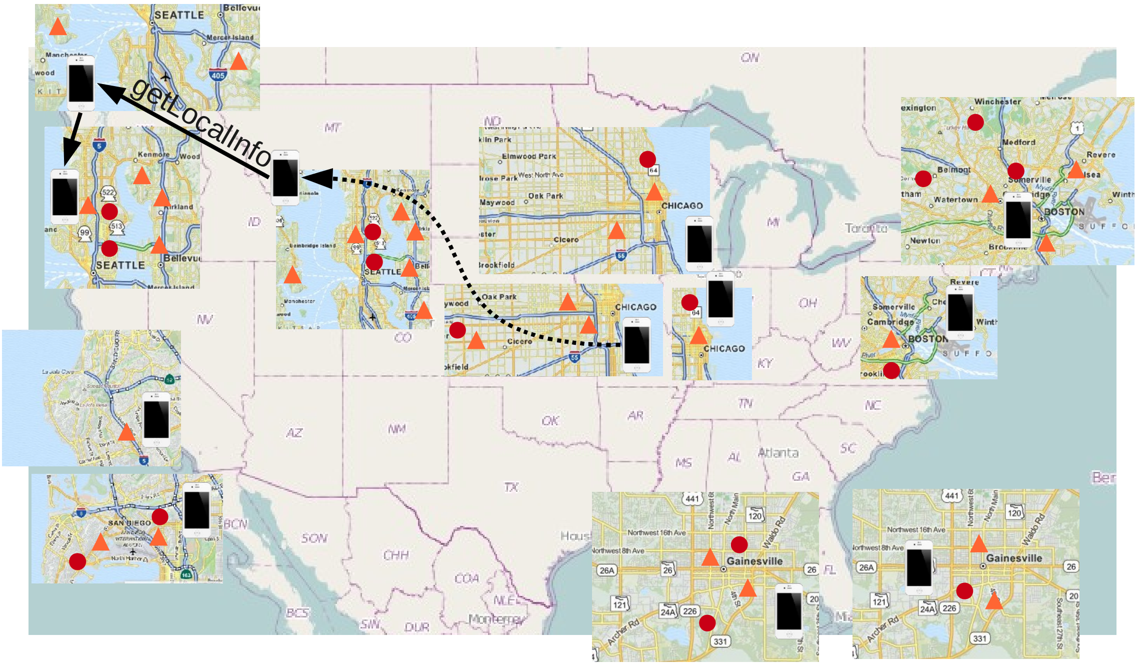


Figure 1.4: Distributed map maintenance for events and restaurant reviews (red dots and orange triangles). Each user maintains a list of information that is potentially interesting in its area. When a user would like information about a given region, for example when it moves into a new area, it queries the network for a cumulated view of stored information.

geographically near to where it is needed and ensuring efficient lookup routing would both reduce network traffic and increase data availability in case of partitioning.

Network partitioning: Since geographic-based network partitioning may be expected due to further infrastructure losses or natural occurrences, location-aware routing is also important to enable lookup routing after network partitions. Replication that anticipates the dynamic changes in the network can also ensure data availability despite partitioning.

1.2.2 Distributed Map Management

Many mobile device users already use distributed applications or contribute to the web 2.0 through platforms such as facebook [fac13] and Yelp [Yel13]. Users are increasingly interested in storing and finding specific information from mobile users, such as product reviews, restaurant reviews for a given area, or event advice for a given city. This inevitably leads to a huge quantity of data, which could be stored in a distributed, location aware fashion on mobile devices and queried from the network on demand. For example, interesting events and restaurants in large cities might be stored collectively in the distributed data management network along with their locations and reviews. When a visitor to Seattle queries the network, the network searches for events, restaurants, and respective reviews on active mobile devices within the city, perhaps using given user preferences, which are then visualized on a map. The structured distributed data management system ensures that data is found from all nodes having participated in and entered information into the system, and not only those that are currently online [1]. Figure 1.4 illustrates how users each have different information which they store locally and can be combined when queried.

However, mobile users will only be interested in participating in data storage and retrieval if their mobile devices are not overrun by the accrued load. In this case, a natural load balancing may be achieved by distributing the information in a location aware fashion to the physical area it pertains to. Areas with much information most likely have many participating nodes within said area inputting data. The same is likely of areas with high query rates. On the other hand, weak nodes with low battery power or bandwidth availability should be used only moderately, while laptops or servers with more resources and longer uptimes should be more heavily utilized. Hence, both resource and location awareness play important roles in the success of the network.

1.2.3 Sensor Networks

Sensor networks may employ static or moving sensor nodes with hard wired or dynamic overlays between the nodes. A sensor network may be responsible for streaming or forwarding sensed data to a sink, distributing data or detected changes among the sensor nodes, jointly processing sensed data, or distributively storing processed data in the sensor network for a specified amount of time. Sensor networks with large numbers of extremely small sensors, for example dust sensors, often operate on battery power and are based on wireless communication. Such networks can be used, for example, as survaillence of earthquake activities using a mesh network distributed across city rooftops as in EDIM (Earthquake Disaster Information System for the Marmara Region, Turkey) as coordinated by the University of Karlsruhe [WEZ$^+$10]; or to closely monitor ground temperatures within a natural region prone to forest fires.

Wireless communication most often happens on an ad hoc basis, with sensor nodes communicating directly with one another, but there are many different approaches for building and routing over an ad hoc network, from cluster based schemes to emergent schemes such as ant-based routing. Thus, sensor nodes demonstrate restricted resources including battery power, bandwidth, and computing power that, if overused or misused with respect to their underlying ad hoc network, can quickly lead to node failures, network delays, lost data, or failed processes. Furthermore, the network can improve its resource usage and data availability by acting in a location aware fashion, reducing the amount of hops that single tasks require by, for example, storing data inside the wireless sensor network nearby to where it will be needed or jointly processing data on nodes that are within direct communication range of one another. Location awareness, however, may depend strongly on the implemented underlay structures. A cluster-based approach, for example, would profit most from location awareness that takes cluster structures and cluster heads into account, as two nodes that are physically close but belong to different clusters may be logically distant within the underlay.

1.2.4 Heterogeneous Cloud

Take for example a cloud environment, in which nodes have varying computing power, energy consumption, and network connections (see Figure 1.5). Efficient and reliable nodes with high

[1] This use case idea originated from Béatrice Finance from PRiSM Laberatory-University of Versailles.

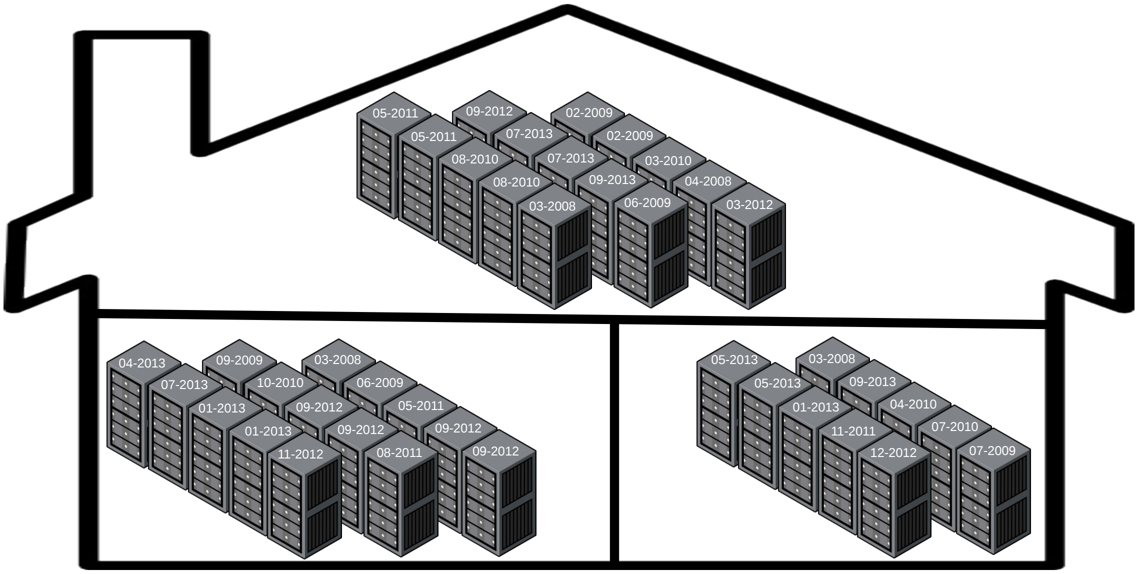


Figure 1.5: Server racks in a cloud have varying physical distances to one another depending on their locations, even if they are all located within a single building. They also have varying strengths and reliabilities depending on, among other things, their load and ages.

computing power should obtain more load than inefficient nodes with low computing power. Furthermore, fewer replicas are required to achieve the same data availability when they are placed on reliable nodes. On the other hand, routing hops should follow low-latency links in order to reduce cross-rack and cross-network traffic, shorten lookup times, and reduce energy costs. Thus, links to nearby nodes within the same rack or regional area should be preferred.

Or consider a second example in which widespread wireless nodes run on battery power (e.g. smart phones which are recharged at regular, dependable intervals) and maintain a distributed hash table using an intact infrastructure: Nodes with high power availability can handle more load than nodes with very limited power availability while latencies between nodes vary greatly depending on nodes' up and down links. While in both scenarios all nodes may cooperatively share the storage load, reducing the maintenance and routing load on weaker nodes would improve the networks' performance. Moreover, in the first case, minimizing inefficient node's maintenance and routing load can help reduce the overall network costs, while in the second case, minimizing low power nodes' maintenance and routing load can help to lengthen nodes' lifetimes between recharging. A homogeneous use of the nodes would ultimately result in shorter battery lifetimes and thus higher failure rates of low power nodes, thus effecting both the network's robustness and overall storage capacity.

1.3 Requirements

While our main goal remains robustness, these use cases provide us with a list of specialized requirements that influence robustness for the distributed data management system and go beyond location and resource awareness. These requirements are explained before taking a more in depth look at the definition issues surrounding location and resource awareness. As we will see in Chapter 4, each of these requirements can be assessed from varying perspectives using numerous measures. The main goal from which all others stem remains:

Robustness. Data that has been input into the system needs to be available at any given time. Thus, robustness can be equated to **data availability**. This means that data can be found when queried, even if there is a very high query rate or the data's original owner has failed. To ensure robustness, the network must remain intact, the network must be able to efficiently store and query for information, enough nodes must be present to cover the required storage capacity and query load, and nodes must be able to handle node failures and data handoffs.

Note that while each of the following six specific requirements has a unique specialized goal, the means towards reaching these goals often overlap, and they all serve the purpose of improving the central goal: overall robustness.

(i) **Self-organization.** The centralized coordination of a quickly growing network based on a large number of users inputting data into the system and storing and querying that data would inevitably lead to a bottleneck. In order to facilitate the system's growth without a bottleneck, decentralized self-organization is necessary.

(ii) **Scalability.** Self-organization alone does not ensure the system's scalability. Nodes may be joining the network and information may be input into the system at a high rate, and the system needs to be able to handle these growth dynamics.

(iii) **Load Balancing.** Load needs to be distributed amongst nodes in a manner that avoids both bottlenecks and underused nodes. When, for example, load is distributed primarily within a single given geographic region where queries originate, then nodes within that region become overused with long delay and high failure rates while nodes in other regions have little to no load.

(iv) **Data consistency.** Data replication is essential in ensuring robustness, but replications need to be held up to date and consistent to ensure that queries return the correct information.

(v) **Resource awareness.** The heterogeneous, possibly mobile nodes should be handled with respect to their varying resource availabilities. Ignoring nodes' strengths or weaknesses may cause nodes to be overburdened or underused, resulting in poor load balancing and possibly causing node failure, thus reducing the overall robustness of the system.

(vi) **Location awareness.** Location awareness aims at integrating nodes' locations into where and how information is stored and how queries are routed within the network. It ultimately leads to lower network traffic and can be used to save information close to where it is needed, also improving the data's availability in the case of node failures or network partitioning.

Robustness along with the requirements (i)-(vi) provide the basis for this work's research goals, which can be summarized as the *development and comparison of system solutions that suitably fulfill these requirements*. Naturally, foundational system decisions are made on the basis of these requirements.

Requirement Characterization

While the goal of this work is to develop and compare system solutions that fulfill the robustness requirement by fulfilling requirements (i)-(vi), the possible systems cannot take the form of a mathematical optimization problem that can simply be solved for the best solutions. However, as we will see in later chapters, a specific system often has settings and parameters that can be used to alter the degree to which various requirements are fulfilled and which can be used to formulate mathematical optimization problems. But first, we must find systems that fulfill all of these requirements to some degree.

While each system can be broken into various design aspects that work together to create the overall system's behavior, note that a system can display load balancing, data consistency, resource awareness, or location awareness even when the respective requirement is not incorporated into every design aspect. For example, an effectively load balanced system might have both imbalanced routing and maintenance load, with routing being performed primarily by one group of nodes and maintenance performed by yet another group of nodes. Or a resource aware system may only have resource awareness integrated into its routing, and not into its data distribution or maintenance.

Integration	Requirement
Prerequisite for all design issues	(i) self-organization
	(ii) scalability
Overall system goals, vary for design issues	(iii) load balancing
	(iv) data consistency
	(v) resource awareness
	(vi) location awareness

Table 1.1: Requirements must be met in different manners.

On the other hand, the self-organization and scalability requirements can only be fulfilled when every aspect of the system is self-organizing and scalable. For this work, if a single design aspect proves not to be self-organizing, i.e. requires central coordination, then the overall system cannot be considered self-organizing. Likewise, a single non-scalable aspect leads to a bottleneck, impeding the overall scalability. Thus, in Chapter 3, where limitations and approaches are discussed, requirements (i) self-organization and (ii) scalability are treated as *prerequisites* for each design aspect while the remaining requirements (iii)-(vi) are treated as *overall system goals* (see Table 1.1).

Chapter 2

Concepts: Resource and Location Awareness

As we will see in the related work discussed in Chapter 4, resource and location awareness have been treated in varying fashions in structured peer-to-peer networks and have seldom been considered together. Unfortunately, the lack of a common foundation makes comparing the characteristics, goals, applicability, and effectiveness of various approaches difficult. Moreover, resource awareness is generally not considered directly in evaluation metrics, hindering the evaluation of the effects of resource awareness on system behavior. In order to facilitate such comparisons, taxonomies for the treatment of resources and locations are introduced in this chapter. These taxonomies do not address how approaches use these resources and locations to achieve resource and location awareness, as discussed in Chapter 4, but rather what and how resources and locations can be interpreted. With the help of these taxonomies, the specific interpretations used throughout this work are then discussed and a basic resource and location framework is established.

2.1 Resources Taxonomy

As previously discussed, resources are understood in a broad sense, ranging from battery power to computing capacities to bandwidth, but always express some heterogeneity among nodes. Yet, how effectively this heterogeneity can be incorporated into a system's design depends on how nodes' resources are defined and observed. Since this clearly varies widely for specific cases, a more specific characterization is necessary to understand the concept and use of "resources." The following five dimensions enable the characterization of how nodes' resources are treated:

R.1 which node characteristics are represented by observing resource availability levels,

R.2 what is interpreted as resources,

R.3 how is the availability of a node's resources determined,

R.4 what scale is used to express this availability,

R.5 how are these resources notationally expressed.

Dimension	Example
R.1 Characteristics reflected by resources	connection strength to network
	available/existent load
	suitability for specific tasks
	time-to-live
R.2 Interpreted as resources	combination of:
	bandwidth, battery power, computing power,
	available software, connectivity to server
R.3 Determining resource availability	global/centralized assignment
	local raw resource measurement values
	local values compared to neighborhood values
	local values compared to global/static values
R.4 Scale	binary $\{0,1\}$
	finite/infinite discrete levels $\{0,1,\ldots,l_{max}\}$ or $\mathbb{N}$
	continuous range $\subseteq \mathbb{R}^{+}$
R.5 Notation	function
	scalar
	tuple

Table 2.1: Concrete examples for resource dimensions.

Table 2.1 offers concrete possibilities for these dimensions, but these lists are not necessarily exhaustive. Note that the various dimensions are independent of one another. We might, for example:

(R.1) wish to represent a node's suitability to run a given program and its corresponding time-to-live by observing its resources while having

(R.2) nodes with battery power and an available software package considered as resources whose availability are

(R.3) determined compared to a given global power scale and the local presence of the software on

(R.4) continuous and binary scales, respectively, but

(R.5) expressed as a scalar by multiplying both values.

Note furthermore that while R.4 and R.5 only represent the mathematical framework in which resources are interpreted and saved, they have significant influence in how effective resources are used. The scale determines at what granularity resources are observed, and thus how much information is made accessible. For example, whether battery power is observed merely as “strong” and “weak” (binary), on a scale of integers between 1 and 4, or in the continuous interval $[0, 1]$ strongly influences how an implementation can treat and differentiate between nodes of varying battery strength. On the other hand, when multiple resources are monitored, the notation influences how the varying resources can be used. For example, if battery power and bandwidth are both of interest, they might be combined into and stored as a scalar sum $quality = power + bandwidth$, used as parameters for a function that expresses node quality in over a node-uptime variable $quality = f_{power,bandwidth}(uptime)$, or stored separately in a tuple $quality = (power, bandwidth)$.

Clearly, these dimensions will strongly influence the effects of resource integration in a system: the better the chosen resources’ calculated notations (a combination of dimensions R.3, R.4, and R.5) reflect the target characteristics in R.1, the more reliably they can be used to the desired effect within the system. In our case, where an improvement in the overall data availability robustness is the main goal, R.1 should also be correlated with individual nodes’ robustness. However, node robustness is inevitably interpreted differently for various scenarios. Take for example a mobile network: time-to-live plays an important role if there are many nodes with short lifetimes while connection strength is more important if nodes have long lifetimes but often loose connectivity on the outskirts of the network. Thus, the goals of this work provide few constraints for R.1 alone, and likewise leave the other dimensions largely without constraints, as it is more important *how* the dimensions work together.

2.2 Location Taxonomy

Of course, the effects of location awareness also hinge on how location and distance is defined and used. While location may be used directly for tasks such as data placement, the corresponding distance may be used to establish links to nearby nodes or to chose nearby routing hops. However, location itself can range from direct physical location to a relative location with limited similarity to the physical world based merely on the round trip times to other nodes. Location can also be characterized via multiple dimensions, so that its working definition depends on:

L.1 what we understand location to express,

L.2 how location is notationally expressed,

L.3 how location is determined,

L.4 how location is used to calculate distances, and

L.5 what we understand distances to express.

Possible values for these dimensions are listed in Table 2.2, although, again, this list is not necessarily exhaustive. The first and fifth dimensions determine how the location information can be meaningfully used within a system, and the accuracy with which L.2, L.3, and L.4 reflect L.1 in L.5 will play a role in determining how effectively the location information can be used to increase robustness.

Dimension	Example
L.1 Location expresses	(i) actual physical location
	(ii) relative location in respect to landmarks or nodes
L.2 Location notation	coordinates
	bins
	(links within) a graph
L.3 Location determined	GPS coordinates - (i)
	triangulation to nodes with GPS coordinates - (i) or (ii)
	latency between node pairs - (ii)
	graph based information - (ii)
	closest found node's bin - (i) or (ii)
L.4 Distance calculated	Euclidean distance
	Shortest distance in graph
	Manhattan distance
L.5 Distance expresses	round trip latency between two nodes
	number of hops between two nodes
	number of ISP's traversed between two nodes
	physical distance between two nodes

Table 2.2: Concrete examples for location dimensions. For "location determined" values, the meaningful corresponding values for "location expresses" are listed.

For example, from the perspective of location aware routing load and replication, if the distance expresses:

- the number of hops between two nodes, then the system may decrease the overall network traffic (by reducing the number of hops used per task) and to place data within a few hops of where it is needed;
- the number of ISP's traversed between two nodes, then the system may decrease the cross-network traffic (by reducing the number of hops between ISP's for tasks) and place data within the ISP where it is needed;
- the physical distance between two nodes, then the system may decrease the total physical distance traveled by lookups (thus decreasing the number of underlay hops for tasks) and place data physically close to where it is needed.

Analogous to resource awareness, our scenarios and the robustness requirements set few conditions on the values assumed for these dimensions, requiring only that the resulting distances effectively reflect the desired behavior between nodes.

2.3 Taxonomy Decisions

This work attempts to take a broad approach to resources and location, as is the case with much of literature as discussed in Chapter 4 but some restrictions are made within the taxonomy. These restrictions provide a foundation for this work with which the applicability of existing approaches can be assessed, new approaches can be developed, and analysis and evaluations can be performed.

R.1 Nodes' resource availabilities reflect some availability-related characteristic. For example, this could be node load if nodes become unavailable with a certain load threshold or connection strength if nodes with weaker connections are often unavailable. The primary use case scenario with mobile nodes with limited battery power corresponds to time-to-live. This assumption is a necessary link between the discussed use case scenarios and the work that follows.

R.2 The resources used to reflect availability are also unrestricted, although the use cases refer primarily to battery power and bandwidth.

R.3 Nodes determine resource availability locally, but their values should be comparable on a global scale. This is necessary in order to uphold the scalability and self-organization requirements. In order to achieve a global scale, each node must know approximately how much battery-power or bandwidth is associated globally with a given availability.

R.4 A finite set of discrete values $\{0, 1, \ldots, l_{max}\}$ is used for nodes' resource levels. This provides a large degree of differentiation with an arbitrary number of levels. While this does not offer as much variation as a continuous range, it eases analysis and evaluation by providing groupings of nodes over which measures can be assessed.

R.5 This work focuses on the scalar notation, exploring the possibilities for integrating resource and location awareness for this most simple and popular notational form. Based on this work, future work could then focus on the more complex tuple or function notations.

L.1 Location expresses actual physical location, such that any two nodes can determine their location with respect to one another. Otherwise, nodes cannot decide if they are physically close to arbitrary other nodes, which diminishes the system's flexibility.

L.2 Location is notated using coordinates, thus facilitating accurate distance approximations between nodes that reflect physical distance. This supports the use of actual physical positions. Note that bins or graphs could indeed be derived from physical locations with coordinates.

L.3 Location can be determined in any way that reflects actual physical location.

L.4 The Euclidean distance is used to reflect physical distance. While this distance is not necessary, it is the most common when using coordinates.. A distance must be established in order to perform the later analysis in Chapters 6 and 10.

L.5 It is only assumed that it is somehow advantageous for nodes to be linked to, route along, and have needed information stored at nodes within a small distance. Location awareness could otherwise be considered superfluous. Thus, distance expresses physical distance but may interpreted to reflect the round trip latency or number of hops between two nodes.

2.4 Network Assumptions

We assume that each node x has sufficiently correct two dimensional virtual (i.e. not necessarily geographical) network coordinates (x_1, x_2). These may be obtained, for example, by nodes measuring their latencies to other nodes and adjusting their coordinates according to a mass-spring system such as in Vivaldi [DCKM04]. Any two nodes x and y can determine their *physical distance* by calculating the Euclidean distance between their network coordinates:

Definition 2.4.1 (Physical Distance.)**.**

$$d_{phy}(x, y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}.$$

Although similar networks could be built using other coordinate dimensionality and distance metrics, this constellation is chosen for simplicity. Similarly, coordinates are treated statically in analysis and evaluation to reduce the number of variables.

For analysis purposes, a uniform distribution of nodes within a disc-space with a fixed diameter was chosen to simplify mathematical comparisons. Simulated evaluations use both these randomly chosen coordinates and coordinates from a set of over 200,000 node positions provided by the used peer-to-peer simulation framework OverSim [IB09].

2.4.1 Resource Availability

There are a fixed number of resource availability levels $l_{max} + 1$. Each node x knows its dynamic resource availability level that is expressed as an integer value

$$x_R \in \{0, 1, \ldots, l_{max}\}$$

for the fixed maximum level l_{max}. The lowest possible resource level (while still operational) is $x_R = 0$ while $x_R = l_{max}$ implies *unbounded* resources. This assumption must be handled with care, for each node still has a maximum load that it can handle and dictated by, for example, CPU or bandwidth. Thus, an upper bound for load per time frame must be considered for these strongest nodes. The resource levels are globally defined so that a given resource level on differing node types is comparable, i.e. nodes with identical resource levels have comparable available resources, but a node's own resource level is only locally known. Considering for example power availability with $l_{max} = 3$, that could mean that a handheld operating on battery power may have resource level two when fully charged, but a cell phone with a weaker battery may only reach a resource level one when fully charged, and only nodes hooked up to constant power supplies can be considered level three. Note that nodes can switch between resource levels

When analysis or evaluation require a distribution of nodes' resource levels, a Zipf distribution is assumed. This reflects trends for node lifetime found in peer-to-peer networks, where node lifetime tends to follow a heavy-tailed Pareto distribution (the continuous counterpart of the

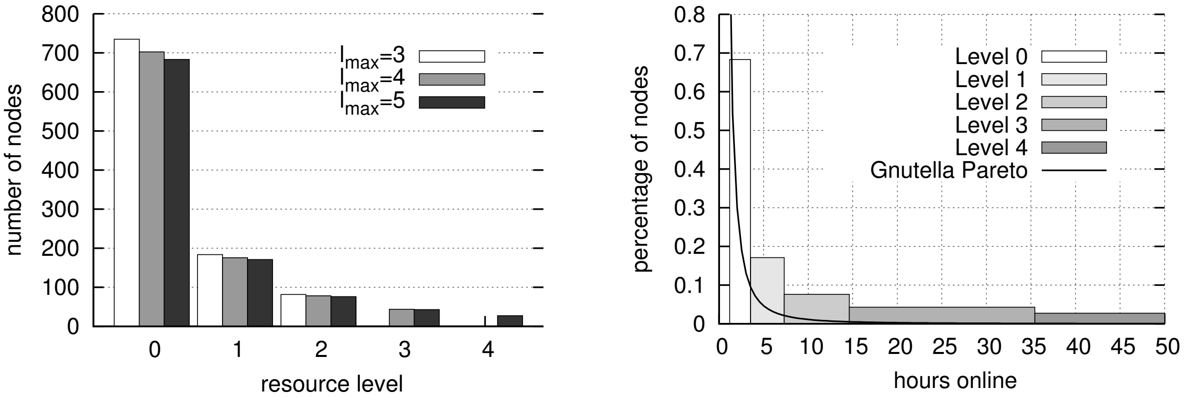


Figure 2.1: Left: The quadratic Zipf distribution for 1000 nodes and 3,4, and 5 resource levels. Right: The Pareto distribution found by Bustamante and Qiao for Gnutella nodes' lifetimes [BQ04]. The lifetime-groupings which correspond to the number of nodes in each resource level for 5-level quadratic Zipf-distribution at right are shown as a bar graph.

Zipf distribution) [BQ04,SGG02]. While Kassinen et al. have used an exponential distribution to model mobile node lifetimes in peer-to-peer systems [KHKY09], Ou et al. showed that the choice between a Pareto distribution, exponential distribution, or Weibull distribution did not significantly effect the performance of their simulated P2P network [OHY09].

The probability that a random node x has resource level $x_R = \ell \in \{0, 1, \ldots, l_{max}\}$ depends on the power ρ of the Zipf-distribution:

$$p_\ell := P(x_R = \ell) = \frac{1}{(\ell+1)^\rho} \cdot \frac{1}{\sum_{j=0}^{l_{max}} 1/(j+1)^\rho}. \tag{2.1}$$

The power of the Zipf distribution is not set for the analysis, but for examples and evaluation where a concrete distribution is needed, a power of two Zipf distribution is used. The corresponding number of nodes per resource level in with a power of two Zipf distribution with a total of 3, 4, and 5 resource levels are shown in Figure 2.1, as is the Pareto distribution's probability density function (pdf) with the corresponding cumulative distribution function

$$F_X(x) = \begin{cases} 1 - (\frac{1.1924}{x})^{1.0607}, \, x \geq 1.1924 \\ 0, \, x < 1.1924 \end{cases} \tag{2.2}$$

found by Bustamante and Qiao [BQ04] to express node lifetimes in the peer-to-peer file sharing system Gnutella [AH00]. The bar graph behind the Pareto pdf shows how nodes' lifetime lengths would have to be assigned to resource levels for $l_{max} = 5$ in order to assign the nodes to resource levels that yield the shown Zipf distribution of 1000 nodes. For example, nodes with lifetimes between the lowest lifetime 1.924 hours and approximately 3.5 hours would be assigned to resource level 0 to yield an expected 683.2 nodes, while nodes with lifetimes between 3.5 and 7.3 hours yield an expected 170.8. Note how, for this quadratic Zipf distribution, the size of the corresponding hour-windows increase along with resource levels. This characteristic has been reflected in simulation setups, as described in Chapter 7.

2.4.2 Node Failure

Node failure is approached from three different perspectives for the mathematical analysis and simulative evaluation in order to assess all of the Chapter 1 requirements: a massive simultaneous failure, a constant drain scenario, and a location-based scenario. For the massive failures, nodes with higher resource levels have lower failure probability (based on a scenario with nodes with varying time-to-live or battery power), with 30% to 50% of nodes leaving simultaneously and ungracefully. This scenario is meant to assess the network's ability to route lookups and find replicas after massive unexpected failures, thus directly evaluating the robustness and data consistency requirements and indirectly evaluating resource awareness. With resource awareness, the system should recognize which nodes are in higher danger of failing and prepare accordingly.

For the analysis and evaluation with the massive failure model, a probability distribution function for nodes' conditional failures $P(F_x|x_R = \ell)$, given the event $x_R = \ell$. This means that a node x's failure probability depends on its resource level x_R. Based again on Bustamante and Qiao [BQ04], a Zipf-distribution is used and it is assumed that the conditional probability is proportional to $P(R = \ell)$, i.e. $P(F_x|x_R = \ell) = \alpha \cdot P(x_R = \ell)$ for some α. Assuming that there are an expected γ simultaneous network failures, e.g. $\gamma = 500$ out of $N = 1000$ nodes fail, then the probability that a single random node will fail is γ/N. This failure probability can be used to derive α:

$$\begin{aligned}\frac{\gamma}{N} = P(F_x) &= \sum_{j=0}^{l_{max}} P(F_x \text{ and } x_R = j) \\ &= \sum_{j=0}^{l_{max}} P(F_x|x_R = j)P(x_R = j) \\ &= \sum_{j=0}^{l_{max}} \alpha P(x_R = j)P(x_R = j) \\ &= \alpha \sum_{j=0}^{l_{max}} P(x_R = j)^2. \end{aligned} \tag{2.3}$$

Thus, for the conditional probabilities $P(F_x|x_R = j) = \alpha P(x_R = j)$ the constant α is:

$$\alpha = \gamma / \left(N \sum_{j=0}^{l_{max}} P(x_R = j)^2 \right).$$

On the other hand, the constant drain scenario assesses how long a network can "survive" without the help of new peers. A network is assumed "dead" after a fixed percentage of nodes have failed, for example 50% or 70%. This is a simplified churn model, in which nodes only fail (i.e. new nodes are not added), and failure is caused by node activities which drain a node's resources until they are depleted. Thus, the above conditional failure probabilities are not applicable in this scenario, since a node's failure depends not only on its resource level but also on how heavily it is used. This failure model is rooted in the use case scenario with nodes running on varying levels of battery power (i.e. resource levels) and drained with each sent and received message. Asymmetrical drain patterns are used, with a send costing more than a receive, but with drain levels constant and do not vary between the resource levels. However, the top resource level is the exception, and since it is considered to have undepletable resources, it is not drained at all (i.e. they do not fail). The selected resource and drain values are kept abstract to serve only in comparing various approaches, and thus do not attempt reflect real world battery runtimes. By observing the system's lifetime and resource usage patterns as the system adapts to node failure, the applicability of various approaches to large resource-limited networks can be assessed. Thus, it directly evaluates the scalability, load balancing, and resource awareness requirements.

The location-based failure model aims at testing and comparing systems' location awareness. Recall that the objectives of location awareness include using location information to improve data availability in the face of node failures, reduce overall load, and decrease lookup times by placing data close to where it is needed. Much of this can be tested in static networks or networks with normal churn. By additionally triggering widespread failures within physical clusters, as might be the case in a disaster scenario, the additional robustness added by the location awareness can be tested. For this failure model, nodes fail with a probability that depends on their distance to a fixed point. Thus, nodes close to the failure hub have a very high failure rate while far nodes are left mostly unscathed. By observing the data availability and lookup latency of data of specific interest to the affected area, the location awareness, data consistency, and load balancing of various approaches can be compared.

Chapter 3

Peer-to-Peer Approaches, Limitations, and Contributions

This chapter explores the possibilities for developing a distributed data management system that fulfills the robustness requirements from Section 1.3, highlights limitations and challenges that must be faced in doing so, outlines this work's specific goals based on the limitations and challenges, and briefly describes the contributions of this work. Open problems associated with these goals and contributions are discussed in Chapters 4 and 8 after an intensive overview of existing approaches and solutions. Thus, the contributions described here are meant merely as an orientation point for the reader.

While not all of the use cases necessarily implement peer-to-peer systems, they are assumed to be the basis of this work. In order to achieve the self-organization requirement, nodes are assumed to cooperate to collectively store, maintain, and query a large amount of data, with each node taking on both a client and a server role. In its client role, a node can input information into and query the system. In its server role, a node maintains, updates, and answers queries on data itself. The following section thus explores the term peer-to-peer and the characterization of peer-to-peer systems in order to provide a basis for further design decisions.

3.1 Peer-to-Peer Systems

There have been many definitions of peer-to-peer systems since the turn of the century, including the following much cited definition from Androutsellis-Theotokis and Spinellis [ATS04] which is based on two characteristics that they ascribe to peer-to-peer systems: the direct sharing of resources (i.e. without an intermediary node/server) and the ability to adapt to network dynamics (e.g node and connection failures).

> Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.

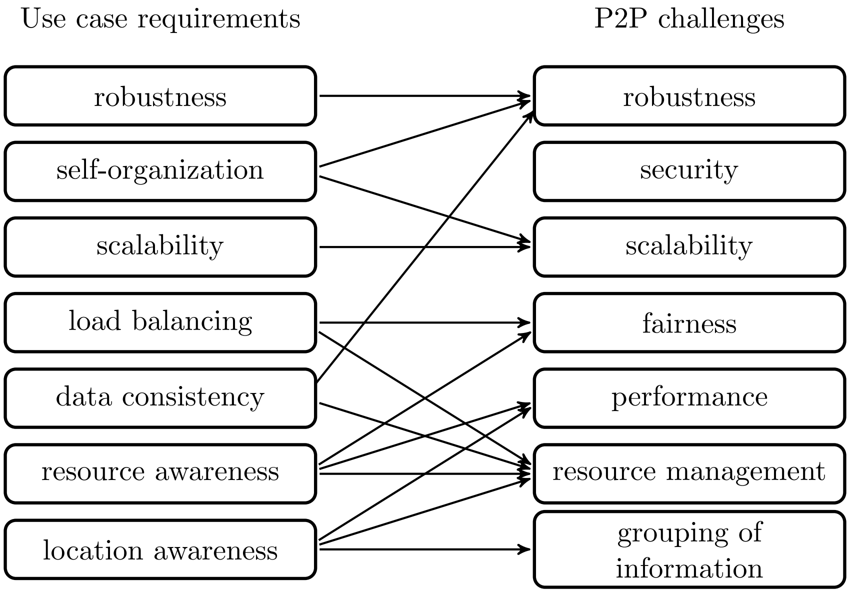


Figure 3.1: Mapping of general peer-to-peer challenges to specific use case requirements.

This definition excludes systems that rely on centralized servers, as does the approach taken in this work. Furthermore, it stresses the importance of sharing resources, while this work aims at examining and optimizing how those resources other than content and storage space are used. The main challenges [KKHY13] or attributes [ATS04] associated with peer-to-peer networks in literature include:

Robustness may be interpreted in many ways. Risson and Moors, for example, consider robustness as a combination of dependability in the face of failure [LZZ+04] and adaptability to changing resources, identities, queries, and application requirements [RM06]. Androutsellis-Theotokis and Spinellis meanwhile group robustness into security as “availability and persistence”, such that data is accessible to users when required despite failures and system changes [ATS04].

Resource management capabilities were reduced by Koskela et al. to “suitability for battery-powered devices” [KKHY13] while Androutsellis-Theotokis and Spinellis focus primarily on data as a resource [ATS04]. These resources are managed, for example, by adapting message patterns to reduce power usage or allowing the removal or documents and adaption of storage space.

Fairness involves each node obtaining a level of load that corresponds to its capabilities. Fan et al. state that “To make the system scalable, peers need to contribute (via uploading to other peers) in order to receive service.” [FLC09]

Performance stands, as a trade-off, in direct relation to fairness [FLC09]. It generally refers to the ability to complete tasks within an acceptable time frame, such as data storage and lookups.

Scalability refers to the system’s capability to uphold its performance and robustness despite an increase in the number of nodes.

Security is a very broad challenge that includes both integrity and authenticity; privacy and confidentiality; and availability and persistence (thus overlapping with robustness) [ATS04]. It thus deals with the accuracy and completeness of data and its origin as well as the protection of data from unauthorized use.

Grouping of information was identified by Androutsellis-Theotokis and Spinellis as an important new area of peer-to-peer research. Data can be grouped in various fashions, from semantic grouping to location-based grouping [ATS04].

Typical measures which are used to evaluate these challenges in implemented systems are discussed with respect to the literature in Chapter 4. The specific requirements established in Chapter 1 can be mapped to the general peer-to-peer challenges as shown in Figure 3.1. While robustness and scalability can be uniquely mapped, the other requirements can be associated with multiple challenges. Note that security is the sole challenge that is not addressed in this work, although it is certainly important for any distributed data management system.

These peer-to-peer challenges have been addressed in a multitude of fashions, and various taxonomies have been developed in order to classify peer-to-peer solutions. Risson and Moors, for example, introduced a taxonomy with categories for four research directions in their research survey on robust peer-to-peer networks: search, storage, security, and applications [RM06]. A taxonomy introduced by Brands and Karagiannis [BK09] that was based largely on Androutsellis-Theotokis and Spinellis survey on peer-to-peer content distribution technologies [ATS04] and was later extended by Koskela et al. as shown in Table 3.1. Considering these categories with respect to the requirements established in Chapter 1, relevant characteristics are then explained and characterized for this work's use cases, further summarized in Table 3.2.

Category	Options	From
Type of service	communication; collaboration; backup distributed computing; content distribution; streaming; data management; information retrieval	[ATS04]
Index centralization	partially centralized; distributed; hybrid; local	
Network structure	structured; unstructured; unstructured and structured	
Security mechanisms	authentication; authorization; access control; encryption; anonymity; accountability	
Caching, replication, and migration	passive/active/cache-based replication; introspective/dynamic replica management	
Deployment	deployed; undeployed	[KKHY13]
Standardization	standardized; proprietary	
Number of overlays	single overlay; multi-overlay	[BK09]

Table 3.1: P2P taxonomy categories.

Type of service. The types of peer-to-peer services have evolved over the past decade, and these categories are similar to those suggested by Vu et al. in their recent book on peer-to-peer computing [VLO10]. Content distribution includes the classic wide-spread file sharing as well as web caching, high/multi-dimensional data sharing, and publish/subscribe applications. File sharing networks are without a doubt the most well known peer-to-peer application, including Napster, Freenet [CSWH01a], Gnutella [AH00] and Oceanstore [KBC$^+$00]. Peer-to-peer web caching saves (popular) internet content at peers within an intranet through which requests can be routed to fetch cached content, while high/multi-dimensional data sharing introduces a more complex file sharing, indexing data according to multiple attributes or dimensions, for example using feature vectors that describe the content of a file via the most frequently used words. Information retrieval, although similar, can be viewed as a separate category of applications that focus on the content-based extraction of information, from which the most visible application as web search engines such as YaCy [Yac13] or FAROO [Far13] that makes use of distributed crawlers. Data management, on the other hand, can be viewed as an extension of content distribution or file sharing, but refers less to the storage of unpredictable, unstructured data than as to the structuring and querying of data over a peer-to-peer infrastructure. Examples include Local Relational Model (LRM) [BGK$^+$02] for modeling distributed data the distributed query engine PIER [HHL$^+$03]. Backup systems such as pStore [BBST01] have more sophisticated versioning and backup mechanisms than file sharing systems and are not intended for general file sharing. Of these data-centric categories, the use case scenarios from Chapter 1 clearly correspond to a content distribution application, or more specifically, file sharing.

Meanwhile, of the remaining categories, the collaboration applications such as the Java software development tool JBuilder [JBu13] or the project-oriented Collanos Workplace [Col13] are inherently peer-to-peer on some level, since their main goal is the cooperation of peers on joint projects. And peer-to-peer communication applications such as Skype [BS06] and Jabber [Jab13] have found wide-spread use, similar to streaming media such as P2PTV [dSLMM08] for television and Spotify [KN10] for music. Distributed computing applications vary widely, from the bitcoin [RH13] currency that can be earned by participating in mining processes, to the numerous applications that use the Berkeley Open Infrastructure for Network Computing (BOINC) middleware system on which the original SETI@home (Search for Extra-Terrestrial Intelligence) project of “volunteer computing” was initiated.

Indexing structure. A system’s indexing structure is considered hybrid decentralized if there is a single server which is responsible for maintaining index lists of where information is stored or resources are located, but nodes communicate directly to perform tasks/retrieve information. In order to locate a suitable peer for a given task, the centralized index server must first be queried. A distributed (or decentralized) indexing structure is independent of centralized servers such that each node is capable of locating suitable peers, while a partially centralized index involves dynamically assigned super-peers which act as index servers. When local indexes are used, peers index only their own data and some form of flooding must be used to locate other peers or the data on other peers. Androutsellis-Theotokis and Spinellis note that peer-to-peer networks are by definition decentralized, but in reality use varying degrees of decentralization, necessitating this differentiation [ATS04]. Since hybrid systems suffer from a single point of failure and loss of scalability due to their bottleneck, the established requirements restrict the indexing structure to decentralized and partially centralized systems only.

Category	Options
Type of service	content distribution
Index centralization	partially centralized; distributed
Network structure	structured
Security mechanisms	*
Caching, replication, and migration	*
Deployment	*
Standardization	standardized
Number of overlays	*

Table 3.2: P2P taxonomy for the approaches examined here, * denotes an arbitrary value for the given category.

Network structure. Every peer-to-peer network is based on an overlay, or graphical representation of the logical connections between peers. In unstructured networks, these logical connections are formed ad hoc, with links added in an unpredictable fashion as peers enter the network. In content distribution networks, the location at which a piece of data is stored is independent from the overlay's (data) structure, and is therefore not allocated to a globally determinable set of nodes. Rather, data is stored, for example, on the node x that it originates from; a number of nodes nearby to x; or to a strong storage node with similar data. Thus, searching for data in an unstructured system involves exhaustively querying nodes using some variation of flooding, since there is no way to know exactly where a piece of data is located. Unless every node in the system is queried, there is no way to guarantee that data present in the system will be found within a given number of hops or successfully found at all (see Figure 3.2 for an example of flooding that is terminated after three hops). Furthermore, if data is replicated, data updates are difficult and not typically dealt with, so that even if a query returns a result, there is no guarantee that the result is up to date. Unstructured distributed data management systems are often built on and reflect many of the characteristics of mobile ad hoc networks (MANETs), with adept location-awareness and shortest-path searches. They are considered well suited for networks with very high levels of churn [ATS04], and have thus been implemented in systems such as Napster and Gnutella [AH00].

In structured networks, on the other hand, links are formed based on deterministic rules and each piece of data is deterministically assigned to nodes based on the overlay's logical structure. These assignments are simply mappings from data onto nodes and each node contains part of a distributed routing table based on this mapping. Queries systematically traverse the overlay with the help of these routing tables so that extant data can always be retrieved within a bounded number of overlay hops. While structured networks have more complicated maintenance, they profit from better scalability and lookup guarantees. Due to the definition of overall robustness as data availability in Section 1.3, any system must provide data retrieval guarantees with lookup bounds in order to satisfy the robustness requirements and be applicable for this work. Therefore, unstructured systems are ruled out at the cost of native location-awareness and lightweight overlay maintenance, and this work concentrates solely

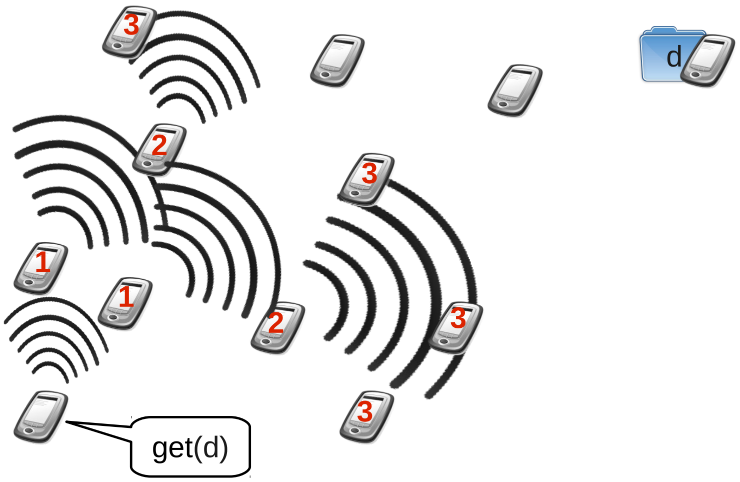


Figure 3.2: A lookup on an unstructured network often requires some kind of flooding to search for data at an unknown position. Here, data “d” is search for, but it is still not found after three flooding hops.

on structured peer-to-peer systems which are almost always implemented as distributed hash tables (DHTs), which are described in the following section.

Standardization. While there are numerous peer-to-peer systems for content distribution, only those which are freely available and whose protocols are freely documented for research and further development are considered here. While proprietary applications may have sophisticated solutions to problems such as battery conservation, they offer little research value without the necessary transparency.

Number of overlays. This category refers to the number of independent overlays which are brought together to form a single, joint peer-to-peer network. These overlays do not necessarily share the same keyspace or structure, but jointly run applications. The established requirements place no restriction on this category, but the developed novel overlays in Chapter 5 are based on single overlay systems for simplicity’s sake.

3.2 Distributed Hash Tables

Structured peer-to-peer systems, comprised to date of almost only Distributed Hash Tables (DHTs) (Mercury is one rare example of a non-DHT structured system [BAS04]), have been implemented for a range of applications from peer-to-peer file-sharing to instant messaging. DHTs map both nodes and data into a joint keyspace so that data can be found in a deterministic fashion using key-based routing. In contrast to unstructured systems, DHTs ensure that lookups of stored data are successful within a given number of hops, provided the network is intact. DHTs typically spread information randomly and uniformly over participating nodes, which self-organize into a logical key-based overlay network according to which nodes have knowledge of each other. DHTs in their original form have a relatively good degree of robustness and fulfill many of the requirements outlined in Section 1.3, including self-organization, scalability, load balancing, and replication protocols that ensure data consistency. And while

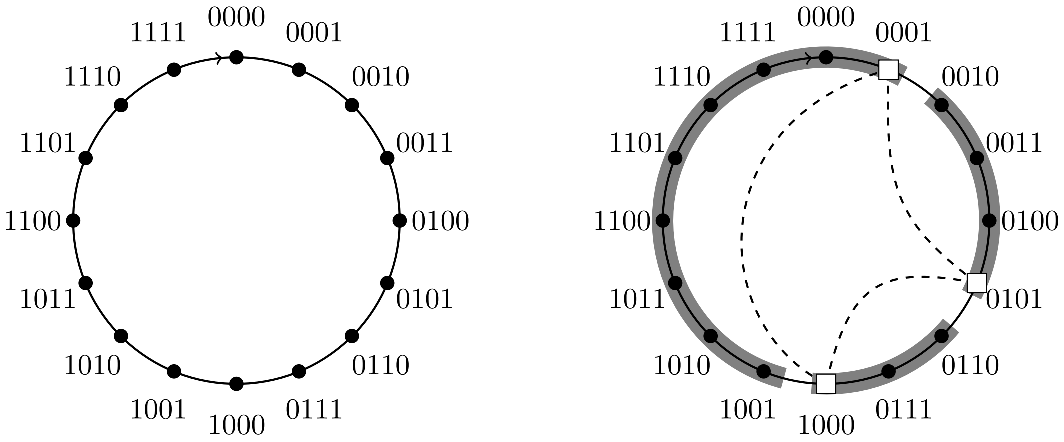


Figure 3.3: Left: all keys from the wrap-around binary space $0^4 \ldots 1^4$ arranged as a ring. Right: Three nodes, shown as squares, have chosen nodeIDs which determine their position in the keyspace. Possible keyranges for which they are responsible are backdropped in gray, in this case using a distance measure of clockwise key distance. Dashed links represent nodes' "nearest neighbors" as might be defined in a ring-topology.

work has been done to improve their location awareness and to handle heterogeneous networks, there is little at the crosssection that incorporates both location *and* resource awareness to increase the system's robustness in dynamic scenarios such as in Section 1.2. This work picks up here and aims at improving the robustness of DHTs on heterogeneous networks with nodes of varying resource availability and with consideration to the outlined requirements (0)-(vi).

A distributed hash table is essentially a hash table with (key, value) pairs (κ, ν) that is broken up to be stored on many different nodes using a mapping of keys to nodes. The basic functions called by users of a DHT are the store operation put(κ, ν), which stores value ν at key κ, and the lookup operation get(κ), which returns the value(s) stored at key κ. DHT design issues include:

(keys) the choice of the keyspace,

(map) the allocation of keys to nodes,

(links) the overlay network generated by logical links between participating nodes,

(routing) the routing of lookups over the overlay network,

(maintenance) the maintenance of the overlay network links,

(churn) the handling of node joins and fails, and

(replication) replication to increase data availability and ensure data consistency.

For each $(\kappa,\ \nu)$ pair, the key is taken from a defined keyspace: for example the wrap-around binary space $0^m \ldots 1^m$ as shown in Figure 3.3. This key value is a piece of data to be stored, possibly ranging from complete files to the address(es) of the node(s) that store the particular data referred to by κ. Typically, each node chooses a nodeID from the keyspace and a system-wide known mapping of individual keys to nodeIDs then partitions the keyspace such that

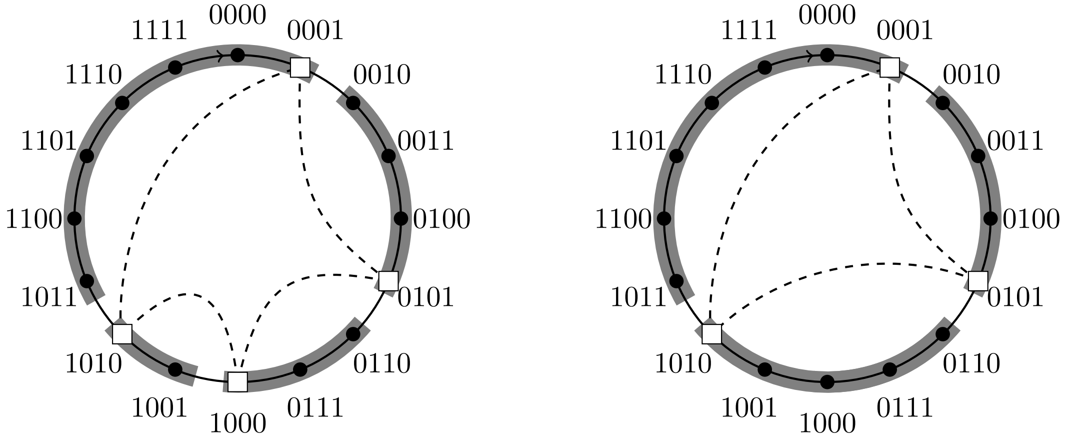


Figure 3.4: Starting from the situation in Figure 3.3. Left: a node with nodeID 1010 joins the network, data must only be migrated between nodes 0001 and 1010. Right: Node 1000 leaves the network, data must only be migrated between nodes 1000 and 1010.

each node recognizes in a self-organized manner the keyrange for which it is responsible (the gray underlayed areas in Figure 3.3 show one example of such a mapping). This mapping employs consistent hashing to ensure that a single node's join or failure only effects a small number of the total nodes, preferably producing significant changes in the routing tables and key values of only one or two neighboring nodes. Figure 3.4 demonstrates how the key values assigned to nodes may change using consistent hashing for node joins and failures. Indeed, consistent hashing is one of the most important characteristics of DHTS, as it is contributes greatly to scalability and self-organization. NodeIDs are chosen in a variety of fashions, most often randomly or as the SHA-1 hash of a node's IP address but also in direct correlation to a node's location. As we will see in Chapter 4, most of the standard approaches (e.g. Chord, Pastry, P-Grid, end Kademlia) do use the m-bit binary wrap-around keyspace, pick nodeIDs randomly, and store (κ, ν) pairs on the node whose nodeID is closest to κ based on a protocol-specific distance (e.g. clockwise distance for Chord, Pastry, and P-Grid; XOR distance for Kademlia).

Using these nodeIDs, nodes then establish logical links to each other on which network messages are routed. These overlay links are established depending on the network topology prescribed by the given DHT and together form the network overlay. While the most popular topologies are rings, prefix trees, and binary trees, these and other topologies have many similarities and often establish links to identical neighbors. Figure 3.3 shows how links might logically be established to nodes' nearest neighbors in a ring-based topology, while Figure 3.5 demonstrates how links to nearest neighbors might be logically determined for a binary-tree based approach. Since overlay links are logical, they most often require multiple physical underlay hops, often on nodes that do not belong to the DHT. Network messages, such as lookup and maintenance messages, are routed on the overlay according to the routing protocol specified by the DHT. All DHTs use key based routing, with many using either greedy routing or digit fixing on the keys, while specialized DHTs also use other paradigms such as geographic or hierarchical routing. Both iterative and recursive routing approaches are used. Iterative routing involves the initiating node sending out search messages for a key, receiving answers about additional closer nodes, and repeatedly sending out search messages to closer

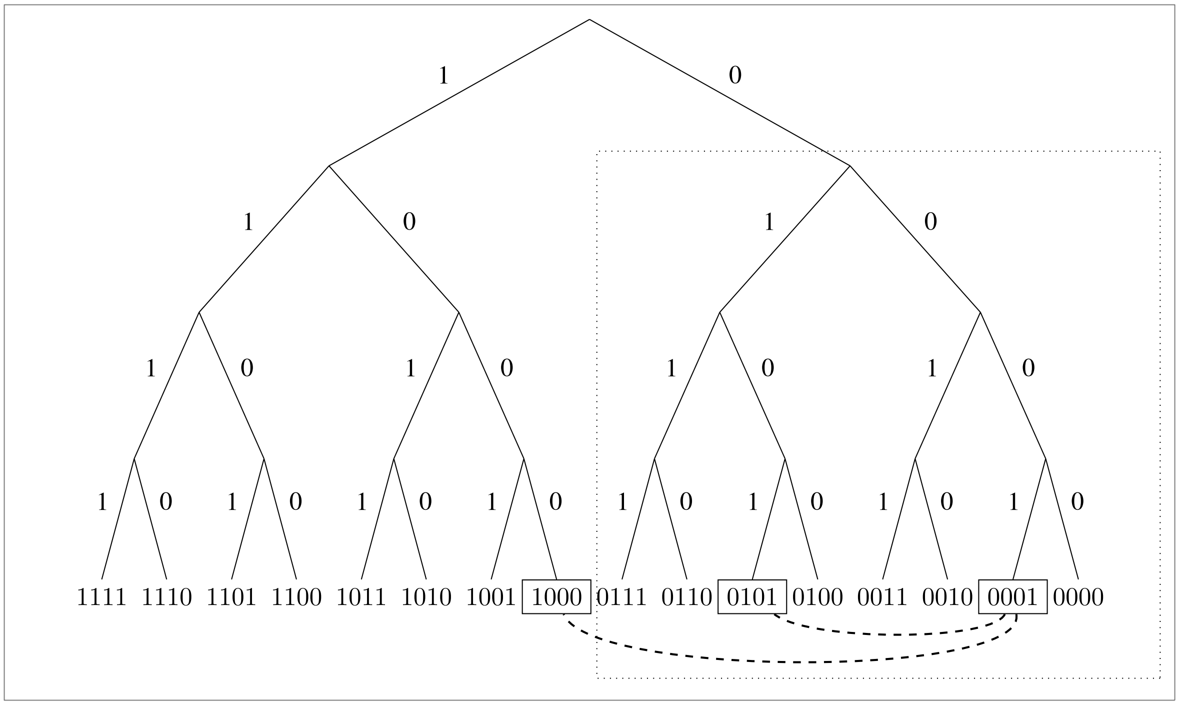


Figure 3.5: The same three nodes with nodeIDs 1000, 0101, and 0001 are used for this binary-tree based example. Each node now looks for a nearest neighbor in the smallest possible subtree to which it belongs, for 0101 and 0001 shown dotted and for 1000 shown in gray. Node 1000 has a choice of two links, and has chosen 0001 in this example.

nodes until the destination has been found and the lookup message is delivered directly by the originating node. In recursive routing, the originating node forwards the lookup message directly to some neighbor node which then forwards the message in turn to on of its neighbor nodes, until the destination node receives the message. The basic routing protocol used by a node receiving a lookup or search message is summarized in Algorithm 1. Both a low number of links per node, i.e. low degree, and a low routing complexity are DHT design goals, but they typically pose a tradeoff: the lower the degree, the higher the routing complexity. Upper bounds on the routing complexity play a central role in comparing DHTs and are often around $O(\log N)$ for networks with N nodes, while nodes' degrees range from constant to slowly growing such as $O(\log N)$ or $O(\sqrt{N})$.

Most DHTs take basic steps to replicate data in order to handle unexpected node failures, often replicating data to several of the closest nodes in the keyspace. More complex replication schemes are necessary for networks with high churn or update rates, and are often presented independently from the system. The steps to discover failed nodes and partitions and then reestablish links to rebuild the overlay are outlined for a DHT regardless of its replication schemes.

So typically (as seen in Algorithm 2), to join a DHT, a node x first chooses a nodeID from the defined keyspace and then contacts some bootstrap node in the DHT with a join request and its nodeID x_{ID}. This request is routed to the node's immediate overlay neighbor(s) (often its immediate neighbors in the keyspace), which then adjust their routing tables, perform data migration with x, and perhaps share network information with x. Once x has established the

Algorithm 1 Lookup routing procedure at local node thisNode.

```
procedure routeTo(msg, key, fromNode)
    if key.isInKeyrange(thisNode) then processLocally(msg)
    else
        for neighborNode in routingTable do
            if key.isInKeyrange(neighorNode) then nextHop=neighborNode
            end if
        end for
    end if
    if nextHop undefined then nextHop = findBestNextHop(routingTable, msg, key)
    end if
    if nextHop undefined then dropMessage(msg)
    else if usingIterativeRouting() then replyToSearch(nextHop, fromNode)
    else if usingRecursiveRoutinge() then forward(msg, key, nextHop)
    end if
end procedure
```

Algorithm 2 Local node thisNode joins via bootstrapNode.

```
procedure nodeJoin(bootstrapNode)
    thisNode.nodeID = choseID(keyspace);
    sendJoinMessage(bootstrapNode, thisNodeInfo)
    processJoinResponse()
    if notEmpty(neighborList) then
        startDataMigration()
        gatherNetworkInfoFromNeighbors()
        for keyrange from overlayLinkKeyranges do
            sendLinkRequest(requestMsg, keyrange)
        end for
    else rescheduleNodeJoin(delay)
    end if
end procedure
```

necessary overlay links to other nodes, it is fully functional in the DHT. Any node in the DHT can initiate a put(κ,value) or get(κ) lookup for any key κ. For example, to find a piece of data associated with a filename, a hash of the filename can first be calculated to obtain the respective key. The lookup is routed to the node responsible for κ either iteratively or recursively. Once a lookup has reached the responsible node, this node either saves the value in its local storage as a (key, value) pair or returns the requested data associated with κ to the node that initiated the lookup. Thus, any node participating in the DHT or client that has rights on a participating node can initiate the storage or request of data in the DHT, which is performed in a scalable fashion with guarantees on the number of overlay hops necessary to reach the responsible node.

Since they were first published around 2000, DHTs have received much attention and have shown themselves to be efficient, reliable approach for storing large amounts of data in a distributed fashion. With a quickly growing network, a DHTs slowly growing lookup latency, load balancing, and low costs for adding new nodes provide the needed scalability. However, well established DHTs such as the Content Addressable Network (CAN) [RFH$^+$01] and Chord [SMK$^+$01] were designed to route efficiently on homogeneous networks without location information. Thus, most of the early DHTs do not inherently differentiate between nodes' characteristics (for example, group associations or resources), nor do they take location information into account when routing lookups or placing data. This causes unnecessary lookup delays and cross-network traffic, wasting network resources such as bandwidth and node power, and leads to overburdened weak nodes and underused strong nodes. Furthermore, they are designed for relatively static networks so highly frequent failures and network partitioning compromise data availability. These shortcomings are especially detrimental in a wireless environment, which thus requires additional design consideration for resources and location.

3.3 DHT Design

Having restricted the problem of fulfilling the system requirements from Section 1.3 to the use of DHTs, the DHT design issues from Section 3.2 must be considered in respect to these requirements (data availability, self-organization, scalability, load balancing, data consistency, resource awareness, and location awareness). Cumulatively, these design aspects must yield a system that provides robust, distributed data management on highly heterogeneous networks of nodes that are restricted by their resource availability and display some sort of location sensitivity. Recall that a network's resource sensitivity may be observable through nodes' varying power availability, bandwidth, computing power, or available software, and a network may demonstrate location sensitivity by displaying varying latency times between various nodes, more physical hops and incurred network load between varying node pairs, or more routing failures to nodes from other internet service providers.

While DHT's fulfill not only the prerequisites of data availability, self-organization, and scalability but also the more flexible load balancing and data consistency requirements, they do not natively address both resource and location awareness. In fact, resource and location awareness often come at the cost of scalability and load balancing. However, as we will see in Chapter 4, some DHTs do integrate either resource *or* location awareness to some degree. The following two sections provide examples of how resource and location awareness could possibly be integrated into the various DHT design aspects. Recall that, as established in Sections 2.1 and 2.2, we have few restrictions on the concrete definition of the resource and location dimensions (R.1)-(R.5) and (L.1)-(L.5) other than an effective reflection of what is expected through (R.1), (L.1), and (L.5) by the respective other dimensions.

3.3.1 Resource Awareness

Each of the DHT design issues from Section 3.2 can be adjusted for resource awareness, although their practicality depends on the given scenario, for example:

- **(keys) and (map)** the keyspace can be chosen as the same space in which the resources are notated and then used to define nodes' keys, resulting in nodes with similar resource availabilities being near to one another in the keyspace,

(links) links can be generated based on nodes' resources, such that only nodes with similar resources are linked or each node has links to each of various resource types,

(routing) routing can be designed with a preference for strong nodes or paths that use nodes with similar resources,

(maintenance) maintenance frequency can be adjusted to be more frequent for weak nodes, increasing awareness of weak nodes' current resource states, or to be coordinated by and more frequent for strong nodes, reducing weaker nodes' maintenance load,

(churn) node joins and fails can be initialized and overseen by strong nodes, and

(replication) data can be replicated to and managed by strong nodes, with replica availability and the degree to which eventual consistency is preformed on each responsible node depending on its resource level.

3.3.2 Location Awareness

Similar to resources, location can also be integrated into the various design issues from Section 3.2, for example:

(keys) and (map) Analogously to resource awareness, the keyspace and nodeIDs can be chosen to reflect physical location, for example a two-dimensional space that wraps around to form a torus (as in CAN [RFH$^+$01]), with a distance preserving mapping between a node's GPS coordinates and its nodeID in the keyspace.

(links) and (routing) To avoid neighbored nodes in the overlay network that are distant in the underlay network, which causes increases in latencies, lookup path hop lengths, and load (and thus decreasing robustness), overlay links can be established to nearby nodes and routing can prefer nearby nodes or overall short routing paths.

(maintenance) Nodes could reduce the overall network maintenance load by performing link maintenance based on the distance to their links, with nearby links maintained more frequently than distant links. This would specifically reduce costly cross-network traffic.

(churn) Node joins and failures could be broadcast within a restricted distance of the respective node, ensuring that "local" nodes are informed of the nodes in their vicinity.

(replication) To ensure that data is more available in case of node failure or network partitioning, which makes lookup routing more difficult especially for physically long lookups, data replications can be placed geographically near to where they are needed.

Note that the examples given for (links), (routing), and (replication), as previously mentioned in Section 2.2, will be of central interest for this work.

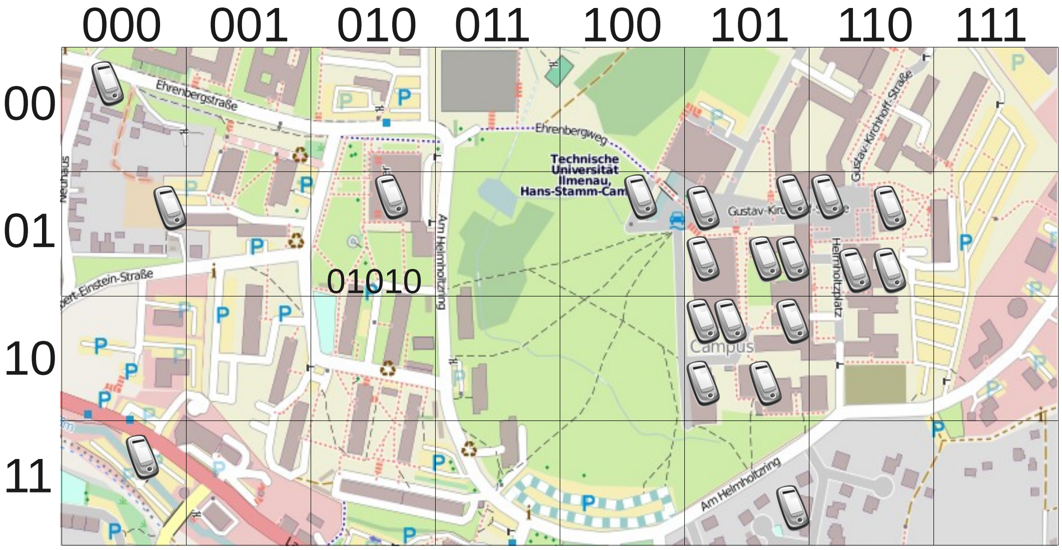


Figure 3.6: Nodes can be assigned nodeIDs depending on their geographic location, as demonstrated here as a combination of a vertical and horizontal coordinates and shown for nodeID 01010. However, highly populated areas result in nodes sharing nodeIDs or, with finer granularity, nodes assuming responsibility for much smaller keyranges. For uniformly distributed data in the keyspace, storage load is thus highly skewed, reducing scalability.

3.3.3 Requirement Tradeoffs

Not necessarily all of the requirements can be concurrently fulfilled for each of the design issues, since for some design issues certain requirements simply do not apply while for other design issues requirements may pose tradeoffs. However, as discussed in Section 1.3, all of the design issues must be self-organizing and scalable. For other requirements, those that are neglected for one design issue must be compensated for or balanced out by another. Each of the DHT design elements is now considered with respect to the system requirements and the respective possible tradeoffs.

(keys) The choice of the keyspace can facilitate a location aware partitioning of and routing within the keyspace, for example, by choosing a two or three dimensional space that reflects the real physical world. However, as also mentioned below, location aware partitioning of the keyspace can lead to a loss of the necessary scalability and load balancing. Multi-dimensional keyspaces can also facilitate high scalability by reducing the key distances between nodes.

(map) Keys can be assigned to nodes based on their locations and/or resources to ensure a certain intrinsic location and/or resource awareness (see Figure 3.6 for an example). However, the scalability and load balancing that are facilitated by the uniform distribution of nodeIDs in the keyspace when nodeIDs are randomly chosen is compromised by this location/resource awareness, and complicated and costly self-organizing workarounds for ensuring uniform distribution must be found in that case.

(links) Scalability for the overlay network means that each node maintains a low number of overlay links that increases only slowly as the network grows. Choosing physically nearby overlay links facilitates location awareness, as choosing overlay links based in some fashion on their resource availabilities facilitates resource awareness. However, self-organizing approaches for identifying strong nearby nodes may be costly and may potentially decrease scalability and load balancing if specific nodes (for example physically central nodes or a few nodes with high resource availability) have significantly more links than others.

(routing) To ensure scalability, each lookup should be routable to its destination with a low number of overlay hops. On the other hand, location awareness means that the total distance traveled by a lookup is low, which often leads to more overlay hops of shorter distances (but in mobile ad hoc networks, fewer underlay hops). Moreover, resource awareness means that the overall resources used for a single lookup are kept low or distributed to strong nodes, but this depends on both the number of hops and the resources of the nodes along the lookup. Hence, an increase in overlay hops from location awareness can lead to an increase in resource usage and thus a decrease in resource awareness as well as a decrease in scalability. However, resource (location) awareness may also overburden strong (central) nodes and under use weak (non-central) nodes, necessitating additional load balancing schemes that distribute manageable routing load to nodes depending on nodes' resource availabilities (locations) in a self-organized fashion.

(maintenance) The load incurred from maintenance must be kept scalable, which may be especially challenging for nodes with a high number of links (for example, from resource or location awareness on the overlay network). While resource awareness and location awareness could integrate node characteristics into maintenance decisions, for example maintaining strong or far-away links less often, this may compromise those connections and the correctness of node information, possibly leading to lost or inconsistent data after failed updates or lookups.

(churn) Self-organization means that there is no central node that registers and unregisters nodes that join and leave the system. With high churn rates, scalable approaches for disseminating information about new and failed nodes can only incorporate a limited number of affected nodes. Yet, this information is important to ensure data availability and consistency, for example, when the owner of a piece of replicated data fails. Resource or location awareness could be integrated by delegating the task of collecting and disseminating this information to strong nodes or nodes within a certain region (for example, physically close to the joined/failed node), but again, these approaches come with load balancing and possibly scalability issues.

(replication) Replication is, of course, the main design issue that addresses data consistency and one of the most influential for data availability. To remain scalable, the amount of additional traffic generated by replica maintenance should be kept manageable, while to remain consistent, the replicas must be maintained so as to provide sufficiently up-to-date data. Here again, resource awareness that replicates data predominantly to strong nodes must beware of possible load balancing issues. Similarly, location awareness may, for example, strive to place replicas physically close to the data owner, to where the data is most frequently queried and used, or to where the data originated. In any case, hotspot locations may emerge, generating a large amount of replication management and rerouting traffic on a restricted physical area and hindering both scalability and load balancing.

Robustness as		self-org.	scalability	load bal.	consistency	resource aw.	location aw.	self-org.	scalability	load bal.	consistency	resource aw.	location aw.	
(map)	self-organization													(links)
	scalability													
	load balancing													
	consistency													
	resource awareness	●	●	●				●	●	●				
	location awareness	●	●	●		●		●	●	●		●		
(routing)	self-organization													(maintenance)
	scalability													
	load balancing													
	consistency													
	resource awareness			●							●			
	location awareness		●	●		●					●	●		
(churn)	self-organization													(replication)
	scalability													
	load balancing													
	consistency		●											
	resource awareness		●	●					●	●				
	location awareness		●	●		●			●	●		●		

Table 3.3: Tradeoffs as listed above for the various design issues.

To illustrate the similarities and differences, the above mentioned tradeoffs are listed in Table 3.3. Note that this list is not necessarily exhaustive and that choice of keyspace has been left out due to the lack of tradeoffs that can be deduced from this design aspect alone. Although not included in the above list, location and resource awareness also pose tradeoffs to one another for each design issue since an increase in location awareness most likely comes at the cost of a decrease in resource awareness and vice versa.

3.4 Contributions

While this work strives to answer how the given requirements can be fulfilled simultaneously in a DHT, an exhaustive search of the set of all possible solutions is implausible and beyond the scope of this work. Instead, the focus is placed on establishing a framework with which to compare systems on the basis of these requirements, with a focus on resource and location awareness, and then developing and comparing several systems that decrease or eliminate the effects of the tradeoffs listed above in order to fulfill all of the requirements. Although replication is considered a design aspect of DHTs in Section 3.2, it is treated as a separate issue in the remaining structure of this work, as it is typically treated separately from the other design issues in literature. In particular, the contributions of this work include:

Awareness Framework. A framework for assessing and comparing the resource awareness of various systems is described in Chapter 2. Taxonomies are introduced for classifying the handling of resource awareness and location awareness within DHTs (or more generally, peer-to-peer systems). These taxonomies are used to classify many existing approaches as well as discern suitable approaches for the introduced use cases.

Novel resource and location aware DHTs. Several novel DHTs which fulfill all of the robustness requirements (i)-(vi) have been designed, as found in Chapter 5, and compared with each other and existing DHTs, as found in Chapters 6 and 7, to better understand the behavior of the discussed tradeoffs in various scenarios. The main DHT design elements considered for resource and location awareness are overlay construction, routing, and maintenance, and results provide insight into how well suited these design elements are for awareness.

Flat vs. hierarchical overlay. Several flat and hierarchical DHTs are also compared in Chapter 6 to determine whether one of the structures is better suited for the integration of resource and location awareness with the given requirements. The effects of varying numbers of hierarchy levels are also examined, further corroborating the results between one hierarchy level (flat structure) and multiple levels.

Novel resource and location aware DHT replication. A replication protocol that builds on the DHTs from Chapter 5 is presented in Chapter 9 and analyzed in Chapter 10. This replication decreases the total number of necessary replicas while increasing the overall resource and location awareness by placing replicas on strong nodes and nodes that are near to where data is needed.

This work thus contributes both an analysis of the effects of varying overlay structures on overall resource and location awareness as well as novel DHT and replication techniques which fulfill all of the derived requirements.

Chapter 4

Related Work: DHTs

Work on DHTs became popular around the turn of the millennium with a burst of high level ideas regarding DHT protocols. In these DHTs, a keyspace (such as the wrap-around space $0^m \dots 1^m$) is first established. Then mappings of each node x to a key in the keyspace, referred to as its nodeID, and each piece of data to its datakey are determined (for example, a SHA-1 hash function of a node's IP to determine its nodeID). A distance function is defined (or implied) over the keyspace and each key κ is allocated to the node who's nodeID has the smallest distance to κ. The last step is then to determine which nodes maintain links with each other (for example, direct neighbors in the keyspace) and how a lookup for key κ is routed along those links to the node responsible for κ. Many of these basic DHTs and their differences are highlighted below in Section 4.1.

With basic DHTs in place, further features such as data replication (see Chapter 8), load balancing, node heterogeneity, and location awareness became interesting. Extended functionalities that dealt with DHT weaknesses were introduced and tested, optimizing DHTs for both real usages as is Section 4.1.1 and further academic study. Specific optimizations often assume a specific underlying scenario, for example a high churn rate for a replication protocol with a high redundancy factor; a mobile ad-hoc network for location awareness; or a heterogeneous network with nodes of varying computing power for load balancing. How awareness has previously been addressed for DHTs is discussed in Section 4.2, and well known and a sampling of more specific resource and location awareness approaches are outlined in Sections 4.3 and 4.4, respectively. Furthermore, those approaches which are both resource and location aware are examined in Section 4.5 with respect to the requirements established in Chapter 1. Evaluation measures used in awareness approaches are discussed in Section 4.6 to frame how these approaches have been expected to perform in order to meet specific design goals. Several resulting open questions, in particular with respect to the design of a distributed data management system the fulfills our established requirements, are discussed in Section 4.7.

4.1 Original DHTs

The PRR [PRR97a] tree from Plaxton, Rajarman, and Richa, was the forerunner of the many later DHTs, and was designed to help locate copies of objects in a data/file sharing system. The first DHTs following suit in literature were mainly (analogously) based on an m-bit identifier ring as keyspace, with the exception of the Content Addressable Network (CAN) which uses a d-torus (a multi-dimensional version of the identifier ring), and focused primarily on defining routing protocols and neighbor links without significant attention to location awareness, replication, resource awareness, dynamic load balancing, or other extensions that came with later work. All of the approaches mentioned here and listed in Table 4.1 use random nodeIDs within their keyspace, most often derived from the SHA-1 hash of some value such as a node's IP. CAN [RFH$^+$01], Chord [SMK$^+$01], and Pastry [RD01] were among the first and most well known approaches on which significant future work was built. Further approaches discussed here are Tapestry [ZHS$^+$04], and P-Grid [ACMD$^+$03] which reflect the basics of Pastry with slightly altered design aspects; Koorde [KK03] which uses a mix of Chord with de Bruijn graphs [LKRG03]; Symphony [MBR03] which is drawn from Chord; Kademlia [MM02] which is built on an XOR metric; Viceroy [MNR02], which uses layers of rings referred to as a butterfly structure; and BATON [JOV05] which uses a tree structure but is not a true DHT. These approaches' characteristics are summarized in Table 4.1. The main topologies - ring, torus, binary tree, prefix tree - are demonstrated with respect to their link structures in Figure 4.1 and their routing patterns in Figure 4.2 using a simple example. For these examples, Chord was used" as the reference ring topology, CAN as the torus, Kademlia as the binary tree, and Pastry as the prefix tree.

CAN is built on a d-torus, or a d-dimensional Cartesian space that wraps around on the edges (note that an m-bit identifier ring can be seen as a 1-dimensional Cartesian space that wraps around on its edges), and nodes are assigned d-dimensional regions for which they are responsible. When a node x joins the network, the node responsible for x_{ID} splits its current region and hands over the respective half to x. Links are established to direct neighbors on the d-torus (i.e., nodes who's regions abut on exactly d-1 dimensions) and lookups are routed to the neighbor that is in the direction of the lookup key. For d=2, CAN provides an interesting starting point for DHTs based on geographic routing when nodeIDs are chosen dependent on nodes' positions [RGJZ04, RHKS02, WR03, JMW03], similar to Figure 3.6 from Chapter 3. Of course, this leads to scalability problems, as discussed later in Sections 4.4 and 2.3. While most DHTs generate networks in which the allocation of identifiers to nodes is independent of the order in which nodes join the network, this is not the case for CAN. Thus, if the same set of nodes with the same set of identifiers join the DHT in two different orders, the nodes' regions may very well differ, causing node's links to differ as well. Hence, it is difficult to analyze CAN's routing or link complexity, since the network's structure (i.e. nodes' regions and links) is not given alone by its nodes with their respective nodeIDs. However, if a CAN with N nodes is divided into N equal zones, then its routing complexity is $O(\log N^{1/d})$ with $2d$ links per node to maintain.

In contrast, and like all of the other protocols mentioned in this section (with the exception of BATON), the Chord protocol uses an m-bit identifier space which wraps around. This keyspace can be expressed as an identifier ring modulo 2^m with values from 0 to $2^m - 1$ or alternatively as values from the interval $[0, 1)$ which are then mapped to the respective binary keyspace. In Chord, distance is measured as the clockwise distance on the keyring, and an ID is mapped to the node to which it has the smallest distance, i.e. to its successor node on the keyring. Each node x with nodeID x_{ID} establishes links to its direct successor and predecessor nodes on the keyring as well as additional links, called fingers, to the nodes responsible for the keys $x_{ID}+2^i$ for $i = 1, \ldots, m-1$ (i.e. the direct successors of these keys). Routing is performed greedily, with lookups passed to the closest predecessor of the destination key and then to

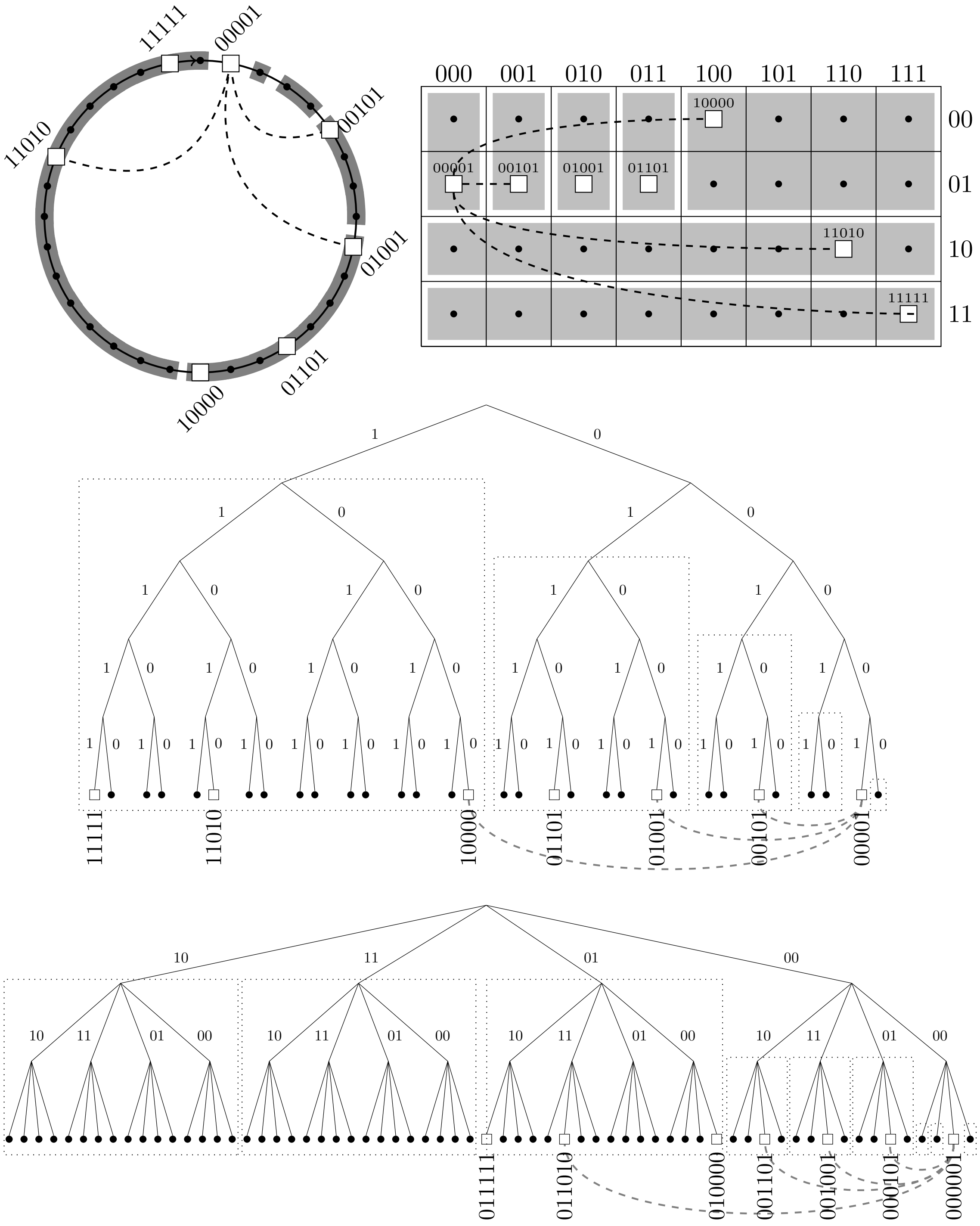


Figure 4.1: Four topologies are shown here with seven nodes and keyspace $0^5 \ldots 1^5$: a ring, a two-dimensional wrap-around space (torus), a binary tree, and a prefix tree (shown with six bits). The topologies are shown with all keys, contrary to the standard view of nodeIDs only (e.g., prefix node 010000 would otherwise be shown as node 0100). The torus is shown flat, with edges that wrap around; the gray areas express the keyranges that each node is responsible for and links are established to bordering areas. For each structure, possible links for node (0)00001 along with the keyranges they are chosen from are indicated.

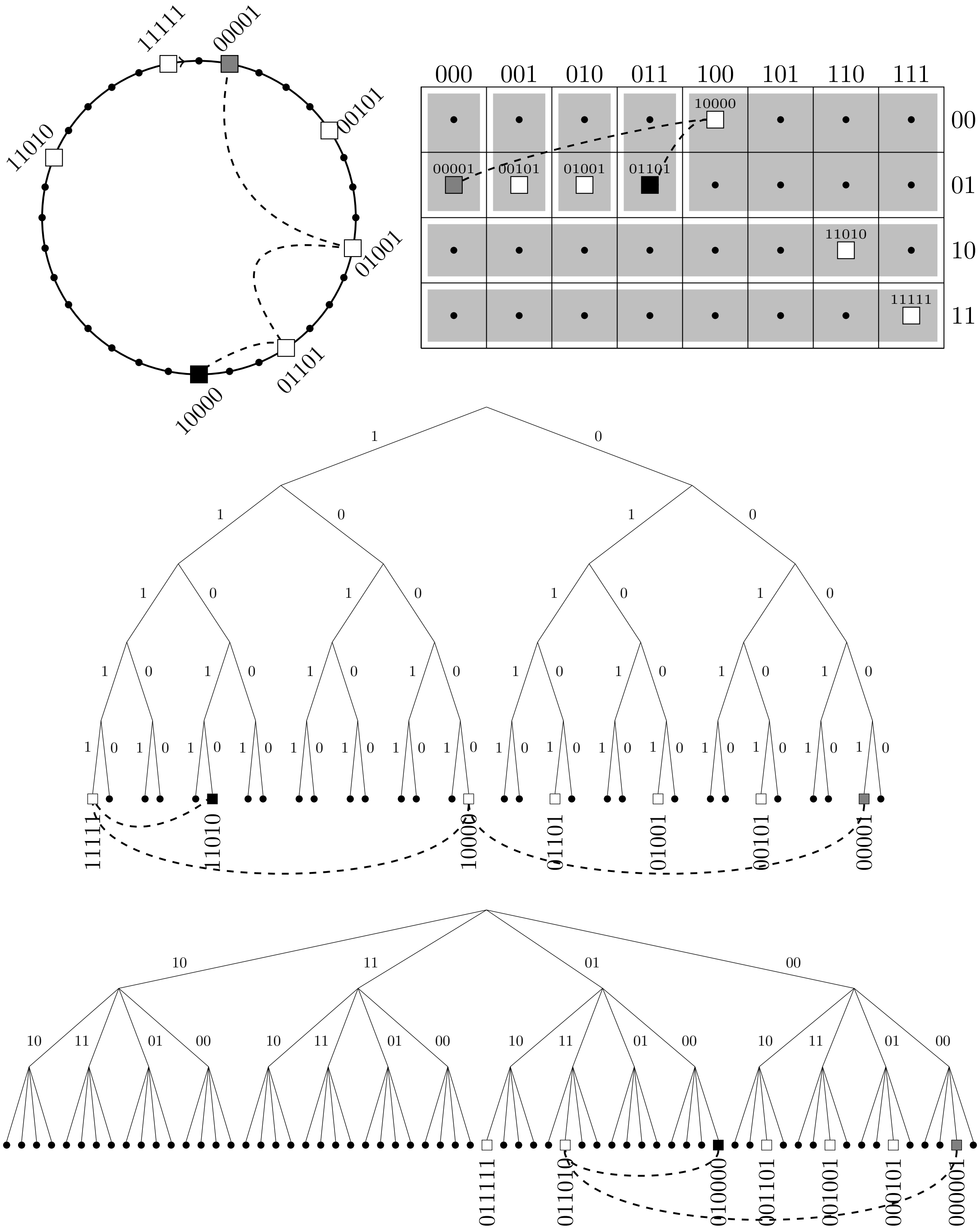


Figure 4.2: Examples of how routing may be performed on various topologies for a lookup from node 00001 (gray) to a nodes shown in black. Since the links shown in Figure 4.1 for 00001 are used, the destination node has been varied to ensure paths of length > 1. Here, the ring and torus use greedy key based routing, while the tree structures use digit fixing, which can also be greedy depending on its implementation.

Approach	Keyspace	Topology	Distance	Routing	Routing Complexity	# Links	LA
PRR [PRR97b]	m-bit ring	prefix tree	clockwise dist.	digit fixing	$O(\log_b N)$	$O(\log_b N)$	
CAN [RFH+01]	d-torus	d-torus	Euclidean	greedy using local neighborhood	$O(N^{1/d})$ N equal zones	$2d$ equal zones	
Chord [SMK+01]	m-bit ring	ring	clockwise dist.	greedy	$O(\log N)$	$\log N$	
Koorde [KK03]	m-bit ring	ring	clockwise dist.	digit fixing	$O(\log N/\log\log N)$ $O(\log N)$	$O(\log N)$ $O(1) = 2$	
Pastry [RD01]	m-bit ring	prefix tree	clockwise dist.	digit fixing	$O(\log_b N)$	$O(\log_b N)$	✓
Tapestry [ZHS+04]	m-bit ring	prefix tree	clockwise dist.	digit fixing	$O(\log_b N)$	$O(\log_b N)$	✓
P-Grid [ACMD+03]	m-bit ring	prefix tree	clockwise dist.	digit fixing	$O(\log N)$	$O(\log N)$	
Symphony [MBR03]	m-bit ring	ring	clockwise dist.	greedy	$O(\log^2 N)$	$O(1) = k$	
Kademlia [MM02]	m-bit ring	binary tree	XOR	greedy	$O(\log N)$*	**	✓
Viceroy [MNR02]	m-bit ring	mult. rings	clockwse dist.	hierarchical	$O(log^2 N)$	$O(1) = 7$	
BATON [JOV05]	-	binary tree	-	greedy	$O(\log N)$	$O(\log N)$	

Table 4.1: Basic DHT approaches. LA = location awareness. *on the basis that Kademlia is similar to Chord, **no upper bound given.

the owner in one final step. When a finger is found for each of these $m-1$ ranges, Chord has $\log N$ links and a routing complexity of $O(\log N)$. While Chord was designed without regard to replication and location awareness, its basic structure has been demonstrated to be very resilient and adaptable [CCN+06, GGG+03] and many extensions have been proposed to add specific functionalities (see Table 4.2). " Koorde uses the basics of Chord - the keyring, the successor lists, the join protocols - but adjusts the fingers to the structure of a de Bruijn graph [LKRG03]. When based on a simple degree-two de Bruijn graph, Koorde has one extra link (i.e. finger) with which it can achieve a routing complexity of $O(\log N)$ by digit fixing. However, there is a factor of three hidden in the $O(\log N)$ that could prove significant in implementation and with so few links, fault-tolerance cannot be ensured. To achieve fault-tolerance, $O(\log N)$ fingers are necessary, for which a $\log N$-degree de Bruijn graph is taken as model, and routing can be achieved in $O(\log N/\log\log N)$ (but again with an unpredictable hidden factor).

The widely used DHT structure introduced with the static PRR - and later adapted to Pastry, Tapestry, and P-Grid - also uses an m-bit identifier keyspace with Pastry and Tapastry using the same allocation of keys to nodes as in Chord, but here the network is viewed as a prefix tree instead of a ring. Instead of routing a lookup in one direction along the identifier ring towards its destination key κ, lookups are based on digit fixing, increasing the length of the prefix shared by κ and the current routing node x's nodeID x_{ID} in each routing step. In PRR, Pastry, and Tapestry, the m-bit identifiers can be expressed in base $b = 2^i$, resulting in wider tries with up to b child nodes for a single parent node. In order to perform digit-fixing routing, each node x establishes for each prefix x_{ID}^P of x_{ID} a link to some node with nodeID prefix $x_{ID}^P \cdot d$ for each digit $0 \leq d < b$ of the used base b (i.e. for $b = 4$, digits $0, 1, 2, 3$). Of course, one of these digits corresponds to x_{ID}'s next digit, and is therefore ignored. The result is a routing table with $b-1$ entries for each $\log_b N$ prefix length, plus neighbor sets with the closest nodes in the keyspace and/or the physical space. Thus, these protocols have $O(\log_b N)$ links and a routing complexity of $O(\log_b N)$. While the routing complexity is lower for increasing b, note that we can view the number of links $(b-1)\cdot\log_b N$ as

$$\begin{aligned}(b-1)\cdot\log_b N &= (2^i-1)\cdot\log_{2^i} N\\ &= (2^i-1)\cdot\frac{\log N}{log2^i}\\ &= \frac{2^i-1}{log2^i}\log N,\end{aligned}$$

which clearly grows for larger i and thus larger b, resulting in a tradeoff between routing complexity and number of links to maintain. The examples shown in Figures 4.1 and 4.2 use a base 4.

Both Pastry and Tapestry also integrate a form of location awareness: Pastry replicates data items to a neighborhood set, so that a nearby nodes can be used for lookups when possible; Tapestry goes further by pushing replicas and object pointers throughout the system, so that local replicas can be found; and both aim to find proximate links for their digit-fixing routing tables. While these protocols have many implementations (see Table 4.2), they are actually quite similar to Chord, with links in each keyrange $(x_{ID}+2^i, x_{ID}+2^{i+1})$ (see Figure 4.1).

P-Grid uses a slightly different protocol for choosing nodes' nodeIDs; each node x chooses a prefix p and assumes responsibility for all keys that start with prefix p. As more nodes join the network, these prefixes can be lengthened in order to distribute load to more nodes. Furthermore, they can be both lengthened or shortened to balance load depending on the data distribution in the keyspace by individual nodes either increasing or decreasing the range for which they are responsible. Note that nodes' ranges can overlap in P-Grid, and additional data redundancy is also injected into the system with adaptive, sampling-based replication and a respective update algorithm [DHA03]. However, due to a potentially m-deep virtual tree and a routing protocol similar to Pastry and Tapestry which performs digit-fixing bit by bit, the worst case routing complexity can be up to linear respective to the network size. Luckily, random links to other nodes have been shown to tackle this problem [Abe02].

While Kademlia also uses a tree structure to visualize the network, it uses the XOR metric for the distance between two keys. Links are maintained as "k-buckets", with the most recently seen k nodes with a given prefix saved as links. Starting with the simple "0" and "1" prefixes, once a node x's k-bucket for prefix p is full, it is split into two buckets of longer prefix length (with prefixes "$p \cdot 1$" and "$p \cdot 0$") in case p is a prefix of x_{ID}. Otherwise, the least recent entry is replaced. Thus, the linked regions resemble those of Pastry with base $b = 2$, but the choice for the next hop during lookups is based greedily on the XOR metric. Due to this similarity, Kademlia clearly has the same $O(\log N)$ bound for the number of links as Pastry (with a factor of k), but a clear upper bound on the routing has not been given in literature, although its similarity to Chord is mentioned. One of Kademlia's interesting features is its caching along lookup routes: once a lookup is successfully completed, the (key, value) pair is cached at the closest node to the key along the lookup path that did not return the value, potentially resulting in high redundancy rates and therefore lower lookup latency.

Using a binary tree structure, BATON is inspired by B$^+$-Trees to enable range queries in a DHT by preventing data from being reordered through a hash function that evenly distributes load. While it has many similarities to Chord, Pastry, and P-Grid, it cannot be classified as a DHT since it hashes neither nodes nor data into a keyspace. Joining nodes are inserted into a suitable position in the tree in order to maintain the tree's balanced structure. Links are established to child and parent nodes as well as specifically defined "adjacent" nodes and nodes that are on the same level within the tree at a distance 2^i within that level (nodes are numbered within a single level), for a total of at most $O(\log N)$ routing table entries. Nodes do not use their nodeIDs to determine which data they are responsible for like other DHTs. Rather, each node receives data from its parent node upon joining the network along with a lower and upper bound for its range of values, which its neighbors store in their routing tables. While BATON boasts an index-like overlay structure for easy range queries with a range query lookup complexity of $O(\log N)$, node failures and churn present a significant challenge, as restructuring is necessary when nodes leave the network. Furthermore, load balancing is very critical for skewed data (which is not randomly uniformly redistributed with a hash function) and a suggested heuristic approach costs $O(\log N)$ *per* insertion or deletion without providing guarantees.

The two remaining DHTs discussed in this section, Symphony and Viceroy, are again based on a ring-structure similar to Chord and achieve relatively good routing behavior for a constant number of links. Symphony distinguishes itself by using random shortcuts (comparable to Chord's fingers) by pulling key distances from a probability distribution function from the family of harmonic distributions. This gives Symphony a structure similar to Kleinberg's small-world grids [Kle00] and provides expected (not worst case) routing complexity of $O(\log^2 N)$ for a fixed number of links.

Approach	Applications
CAN [RFH$^+$01]	PIER, pSearch
Chord [SMK$^+$01]	CFS, SOSIMPLE, pStore, PRISM, M-Chord, Mercury, SOSIMPLE
Pastry [RD01]	PAST, Pastwatch, SCRIBE, SplitStream
Tapestry [ZHS$^+$04]	OceanStore, Bayeux, Mnemosyne, SpamWatch
P-Grid [ACMD$^+$03]	UniStore, GridVine
Kademlia [MM02]	Kad Network, Mojito DHT, I2P-Bote, cSpace OverNet, Mainline DHT, RetroShare

Table 4.2: Noteworthy applications of original DHTs and their characteristics.

Viceroy was designed for networks with extreme scale and traffic, and takes a very different approach by constructing one central ring and then many multiple levels of rings as well. Nodes randomly choose a ring-level based on their perceived size of the network such that there are approximately $\log N$ levels. After links are established to the predecessor and successors within the level-ring, one link to a higher level ring and two links to lower level rings are also established, resulting in a total of seven links. Routing is performed in three phases, first lookups are routed upward to level one, then routed back down as far as possible, and then routed using successor links to the correct node. Viceroy has been attributed with an expected routing complexity of $O(\log N)$, but $O(\log^2 N)$ with high probability.

4.1.1 Applications

There are a wide range of applications running on or using DHTs. Drawing from the peer-to-peer *type of service* classification from Chapter 3, applications are grouped here into the categories content distribution; data management; streaming, backup, and information retrieval; communication and collaboration, and are discussed with respect to DHTs. For an overview of which applications use which DHTs (where applicable), see Table 4.2.

Content distribution. File sharing produces a large portion of the internet's traffic, and many of the file sharing networks' clients now have DHTs implemented as well, which help by storing information about how to find peers and files. For example, the Gnutella [AH00] client gtk-gnutella [gtk13] and the former client Limewire [Lim13] both use the Kademlia-based Mojito DHT [Liu09] and the BitTorrent [Bit13] clients such as Vuze (formerly Azureus) [Vuz13] use the Kademlia-based Mainline DHT [JOK09]. The Kademlia-based Kad Network, on the other hand, is mostly used along with the file sharing eDonkey2000 network [eDo13] to store and search for eDonkey2000 file hashes, ratings, and locations. Clients such as eMule [eMu13] are specialized in connecting to both the Kad Network and eDonkey2000.

Distributed file system applications include the Pastry-based PAST [DR01], which determines where files are stored (as opposed to only storing pointers to the data), similar to OceanStore [KBC$^+$00], the Cooperative File System (CFS) [DKK$^+$01], or the DHT-like Freenet [Fre13,CSWH01b]. The Tapestry-based Mnemosyne [HR02] application provides data-hiding, or steganography, by encrypting and "hiding" sensitive data in large volumes of random data written on each peer. SCRIBE [CDKR02], on the other hand, is a Pastry-based publish/-subscribe application which hashes the concatenation of topic names and their publishers to determine which peer is responsible for published data. Other peers can then subscribe to the published data at the responsible node. Meanwhile, caching applications such as the Coral Content Distribution Network [Fre10] and CoDeeN system [Cod13] cache web content by storing popular websites on peers to provide quicker loading of slow or overloaded websites.

Squirrel [IRD02] is an example of a peer-to-peer web caching application, and which uses Pastry as an index to locate peers within the intranet that have cached copies of desired objects. High/multi-dimensional data sharing systems include Mercury [BAS04], which supports multi-attribute range queries by constructing multiple Chord rings which each store data according to varying attributes and can thus be simultaneously queried, and M-Chord, which supports high-dimensional similarity searches by mapping high-dimensional data onto one-dimensional identifiers [NZ06].

Database management. DHT applications range from the distributed query processing system PIER [HHL$^+$03]; to the distributed continuous (i.e. stream) query processor Medusa [CBB$^+$03]; to Amazon's Dynamo that uses key-value storage but focuses on extremely high data availability for its zero-hop DHT, where each node has sufficient links to directly route lookups [DHJ$^+$07]; to the P-Grid systems GridVine PDMS (Peer Data Management System) which uses schemes and schema mappings to utilize semantic information [ACMHP04] or UniStore [KSR$^+$07] which focuses on efficient query processing over distributed structured data.

Streaming, backup, and information retrieval. SplitStream is one example of a (SCRIBE) DHT-based content distribution system designed specifically for distributing load fairly for high-bandwidth streaming [CDK$^+$03]. Bayeux [ZZJ$^+$01], on the other hand, is a publish/subscribe system similar to SCRIBE but developed specifically for multimedia streaming. Backup systems that use DHTs include pStore [BBST01], which stores blocks of backup data on peers determined by the underlying Chord, Pastiche [CMN02], which uses DHTs to merely find backup "buddies" on which to store backups for one another, and PeerStore [LZT04], which is built on an unstructured network and uses the DHT only for finding and tracking backup locations. Information retrievale DHT-based search engines include BitTorrent's BTDigg [Btd13] and YaCy [Yac13], while pSearch [TXM03] and PRISM [SGE$^+$05] are complete file sharing systems with content-based searching through super-peers forming and additional DHT to store multi-dimensional semantic vectors. Peer-to-peer spam filters such as SpamWatch [ZZZ$^+$03] may also be considered in this category, since their primary purpose is to identify similar content within emails.

Communication and collaboration. Communication protocols that use DHTs include distributed email services such as the I2P (Invisible Internet Project) service I2P-Bote which stores email messages in a Kademlia DHT [I2P13]. Retroshare, on the other hand, uses a DHT to store peers' IPs for peer-to-peer email, instant messaging, and file sharing [Ret13]. The Kademlia-based cSpace also offers a variety of friend-to-friend services, including screen sharing and instant messaging [cSp13,RB07]. Yet other applications offer voice over IP services with DHT-based directory services, such as OpenVoIP [BS08] or SOSIMPLE [BLJ05], or distributed version control systems such as Pastwatch [YCM06].

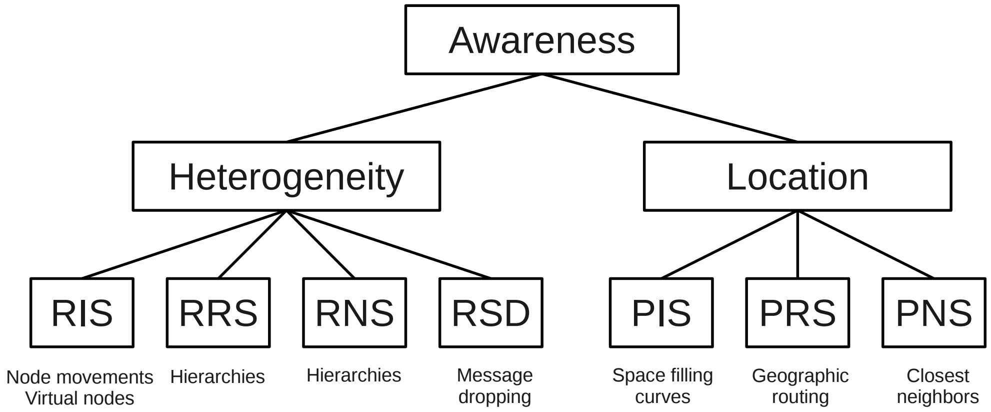


Figure 4.3: Characterization of DHT heterogeneity and location awareness with application examples.

4.2 Awareness

While the original DHT's focused on the basic construction of links and routing, many networks such as those discussed in Chapter 1 can profit from additional awareness about the network's and nodes' states. As we will see in Section 4.3, resource awareness approaches have been developed for heterogeneous networks and highly dynamic systems with high churn rates by treating nodes differently within the DHT depending on their characteristics. Similarly, much effort has been invested into developing routing and load allocation optimizations with the help of nodes' physical locations, or location awareness, for mobile (ad hoc) networks and networks with high load, as we will see in Section 4.4.

Although mobile networks are often heterogeneous and heterogeneous networks often involve mobile nodes, these two issues have been viewed almost exclusively separately in the past. Heterogeneous networks, where nodes have varying capabilities or attributes of some sort, are often addressed by balancing nodes' loads with respect to their capabilities: the stronger or more reliable a node is, the more load it is assigned. Here, the main approaches are the use of virtual nodes, node movements within the identifier space, and hierarchical overlay structures. However, a fair amount of work has also focused on (hierarchically) structuring overlays for heterogeneous nodes whose varying attributes are *not* associated with strengths, for example organizations or shared physical locations. Mobile (ad hoc) networks, on the other hand, are treated primarily from a location aware routing perspective, adapting the overlay structure and protocols to (ad hoc) routing over wireless links: overlay hops should be nearby hops that incur as little underlay load as possible. Some combination of proximity aware identifier selection, routing selection, and node selection (see Section 4.4) are integrated into the DHT to this end, often with the help of clusters and cluster heads (i.e., a rudimentary hierarchical design).

4.3 Resource Awareness: Heterogeneous Netwoks

As mentioned earlier, the three main approaches to addressing heterogeneous node capabilities have been the use of hierarchical DHT structures, virtual nodes, or node movements within the identifier space, with the primary aim of balancing either communication or storage overhead. However, several new approaches focus specifically on decreasing the energy consumption of mobile nodes, such as Gurun et al. [GNZ06] who suggest that the most important energy conservation approach could be idle-times when nodes do not receive messages or Kelenyi and Nurminen [KN08] who suggest that weak nodes should simply drop a percentage of incoming messages. With these approaches in mind, a classification for resource awareness approaches can be derived similar to the classification for location awareness from Gummadi et al. [GGG+03] as described in Section 4.4. This classification contains four categories (as opposed to the three used for location awareness) which can be applied concurrently within a single application:

RIS Resource Identifier Selection: the selection of a node's nodeID depends on, among other things, the node's available resources and the node's level of load. Virtual nodes and node movements both use RIS, as may certain hierarchies.

RNS Resource Neighbor Selection: nodes select links based on their own and/or their links' available resources, as is often the case in hierarchies.

RRS Resource Route Selection: each routing hop for a lookup is chosen based not only on the destination key, but also on the current node's and/or its links' available resources, as may be the case in resource aware hierarchies.

RSD Resource-based Service Denial: services such as message forwarding or participation in maintenance my be denied depending on a node's resource availability and level of load. The newer approaches mentioned above fall into this category [GNZ06, KN08].

While the fourth category RSD diverges from the classification for location awareness, it comprises an important strategy for dealing with low or non-existent resources: withholding resources. Note that RNS and RRS often occur hand in hand, for instance when strong nodes are chosen as superpeers which are both highly linked (RNS) and preferred as next-hops (RRS). Figure 4.3 shows both the resource and location awareness characterizations (see Section 4.4) with application examples for each category.

Many of the suggestions for DHTs on heterogeneous networks are described below and grouped together based on their traditional approach: virtual nodes, node movements, hierarchies based on varying resources, and hierarchies based on varying group associations. Hierarchical approaches are furthermore classified as vertical or horizontal, where a vertical hierarchy uses a form of gateway or superpeers to switch between otherwise self-contained hierarchy layers while in a horizontal hierarchy each node maintains connections to nodes in other hierarchy layers. Table 4.3 contains a characterization according to which awarenesses and approaches they use. Those that use some form of resource awareness are further characterized in Table 4.4 according to the resource dimensions from Section 2.1, and in Table 4.5 according to RIS, RNS, RRS, and RSD. Note that the main focus here is placed on approaches that actively consider heterogeneous networks, with either the aim of distributing load of varying sizes to nodes depending on nodes' resources or building a single DHT on nodes from heterogeneous networks. Since many of the virtual node and node movement approaches were not necessarily designed for node heterogeneity, but rather to balance load in homogeneous networks, most of the approaches in Table 4.3 involve some form of hierarchy.

Approach	Applied to	Location Aware	Resource Aware	Node Movement	Virtual Nodes	Hierarchy	Hierarchy Type	Multi-overlay
HIERAS [XMH03]	Chord,*	✓				✓	horizontal	
expressway [XMK03]	CAN,*	✓	✓			✓	horizontal	
[GEBF$^+$03]	*	✓	✓			✓	vertical	✓
Coral [FM03]	Chord, Kademlia,*	✓				✓	horizontal	
SkipNet [HJS$^+$03]	-	✓				✓	horizontal	
Canon in G Major [GGGM04]	Chord, Symphony, CAN, Kademlia	✓				✓	horizontal	
[MD04]	*	possible				✓	vertical	✓
SmartBoa [HLZ$^+$04]	-		✓			✓	horizontal	
Cyclone [ALAS05]	Chord(=Whirl), *	✓				✓	horizontal	
Y_0 [GS05]	Chord, *		✓	✓	✓			
SuperPastry [CCR05]	MS Pastry		✓			✓	vertical	
HeteroPastry [CCR05]	MS Pastry		✓					
HONet / HDBNet [TXZ$^+$05]	De Bruijn Graph, *	✓	✓			✓	hybrid	✓
LIP [RQMH06]	*	✓				✓	vertical	
idle phase [GNZ06]	Chimera [Chi13], *	✓	✓					
superpeer-child [ZDK06, ZDK07, ZHDK09]	Chord		✓			✓	vertical	
message dropping [KN08]	Kademlia, *		✓					
hierarchical Bamboo [TWS$^+$09]	Bamboo	✓	✓			✓	vertical	
SkewCCC+ [BBKK10]	SkewCCC	✓	✓	✓	✓	✓	horizontal	

Table 4.3: Characterization of heterogeneous DHT approaches, * refers to the applicability to any arbitrary DHT.

Approach	R.1 Characteristics	R.2 Resources	R.3 Determining availability	R.4 Scale	R.5 Notation
expressway [XMK03]	reliability central location	load forwarding capacities connections availability	self-decided	yes-no	scalar
[GEBF$^+$03]	stability availability	CPU bandwidth uptime	superpeer collects and evaluates characteristics	yes-no	scalar
SmartBoa [HLZ$^+$04]	stability	bandwidth	current ability to handle load	0-128	scalar
Y_0 [GS05]	load bound	storage space processing capacities last-mile bandwidth	-	cont. range	scalar
SuperPastry [CCR05]	load bound	bandwidth storage space processing capacities	-	yes-no	scalar
HeteroPastry [CCR05]	load bound	bandwidth storage processing capacities	-	cont. range	scalar
HDBNet [TXZ$^+$05]	load bound	processing capacities bandwidth	-	cont. range	scalar
superpeer-child [ZDK06, ZDK07, ZHDK09]	load bound failure prob.	bandwidth	actual # uploaded msg vs. # willing to accept	cont. range	scalar
hierarchical Bamboo [TWS$^+$09]	stability strength	processing capacities storage space bandwidth uptime	-	low-medium-high	scalar
SkewCCC+ [BBKK10]	strength	bandwidth	-	2^0-2^k	scalar

Table 4.4: Characterization of resource handling in various heterogeneous, resource aware DHTs.

Approach	RIS	RNS	RRS	RSD
expressway [XMK03]		✓	✓	
[GEBF+03]		✓	✓	
SmartBoa [HLZ+04]		✓		
Y_0 [GS05]	✓			
SuperPastry [CCR05]		✓	✓	
HeteroPastry [CCR05]		✓	✓	
HDBNet [TXZ+05]		✓	✓	
idle phase [GNZ06]				✓
superpeer-child [ZDK06, ZDK07, ZHDK09]		✓	✓	
message dropping [KN08]				✓
hierarchical Bamboo [TWS+09]		✓	✓	
SkewCCC+ [BBKK10]	✓	✓	✓	

Table 4.5: Characterization of resource integration in various heterogeneous, resource aware DHTs.

As we will see, most of these approaches are difficult to adapt to all of the requirements for our scenario. The resource-unaware hierarchical approaches discussed here focus on organizational node heterogeneity with the goal of using nodes from separate organizations or clusters to form a joint DHT. For example, various companies my each maintain an independent inter-company DHT but collectively form an additional global DHT on which nodes can search for and save data at other companies' sites. While these approaches have also been referred to as homogeneous approaches [ALS07], we refer to nodes' varying organizational associations as heterogeneity. Location awareness clearly makes sense in these cases, as is reflected in Table 4.3. On the other hand, almost all of the approaches that do use a form of resource awareness have focused on (among other resources) heterogeneity as nodes' varying bandwidths (see Table 4.4). In this case, location awareness is not the rule but has been used in some approaches which are examined more closely with respect to our requirements in Section 4.5.

Resource aware approaches (see Table 4.4), while they all share a general focus on bandwidth, have interpreted resources as expressing upper load boundaries or node availability, while node availability is alternatively referred to as reliability, stability, availability, failure probability, or strength. While each of these terms has a different nuance, their exact meanings are not adequately discussed to differentiate between them. And while all of them use simple scalar notation (i.e. a single value to express a node's resources), they do vary in their used scales from simple binary scales (i.e. nodes "have" or "have not") to continuous and set ranges to express how much resources single nodes possess. However, how the actual values are determined has been left open for all of the approaches, although a few do abstractly address the idea.

4.3.1 Virtual Nodes and Node Movement

The core concept behind virtual node and node movement approaches is that each node is assigned a portion of the keyspace proportional to the given node's ability to handle load. Thus, the physical nodes have varying quantities of data that they are responsible for storing and answering queries on. However, these concepts assume that higher resource availability infers larger storage capacities, which is not necessarily applicable to a network in which nodes have varying access to resources such as energy or computing power, but homogeneous (or very bounded) storage capacities. Furthermore, considering that the main communication load in a DHT is often accrued from maintenance, it is not nodes' storage capacities that need to be treated heterogeneously but rather the maintenance overhead required of the nodes.

When using virtual nodes, each physical network node balances its load independently by hosting a number of virtual overlay nodes that corresponds to its capacity, each with its own set of keys and links, as introduced by Karger et al. [KLL$^+$97]. Since capacity refers to the amount of load a node can handle, the number of virtual nodes hosted by a single physical node may be dynamic with adjustments depending on the physical node's current load, such as in later work from Karger and Ruhl [KR04] (however, this approach only activates one virtual node at a time) which extended work from Rao et al. on static systems without virtual node insertions and deletions [RLS$^+$03]. For example, a strong node with low load may add more virtual nodes in order to obtain a larger portion of keys as well as more links and more routing load. Thus, one physical node with multiple virtual nodes also maintains multiple sets of overlay links and performs all of the routing and maintenance required for each of those virtual nodes. This incurs not only higher costs at the node hosting multiple virtual servers, especially from the relatively high cost for maintaining direct successors and predecessors within ring based key structures, it also increases the number of DHT nodes and thus the expected lookup hop count (which typically depends on N, the total number of nodes, for example $O(\log(N))$). To counteract this cost increase, it has been suggested that virtual nodes act as a single entity by sharing links and routing responsibilities [BBKK10], thus reducing the redundant maintenance load and simultaneously improving the routing performance by lowering the lookup hop count. However, in order to determine a suitable number of virtual nodes for any physical node, some knowledge about the total number of nodes, the overall overlay load, and/or the average load per node is often required, thus introducing additional network maintenance. Furthermore, virtual node dynamics for load balancing generate additional churn from nodes adding and removing virtual nodes as their load fluctuates.

While the previously mentioned virtual node approaches aim to balance load without explicit attention to nodes' varying resource availabilities, both Y_0 from Godfrey et al. [GS05] and the hierarchical ScewCC++ from Bienkowski et al. (see below) [BBKK10] incorporate resource awareness for heterogeneous systems. Y_0 uses both virtual nodes and node movements to adjust the load for nodes with varying capacities such as storage space, processing speed, or last-mile bandwidth, assigning high capacity nodes multiple virtual nodes within a keyspace block to reduce the link maintenance.

Similar to virtual nodes, and often in combination with them [KR04,RLS$^+$03,BBKK10,GS05], node movements within the identifier space achieve load balance by adjusting the data that each node stores [Man04, NW03, AKU03, BAS04, GBGM04]. These adjustments are made by shifting the key ranges that nodes are responsible for, thus assigning new nodeIDs to nodes.

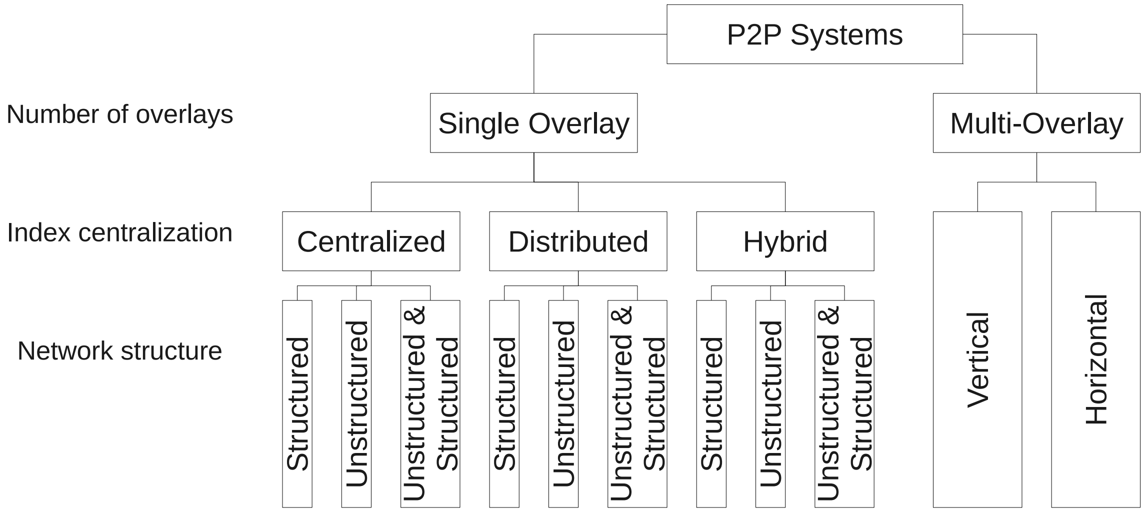


Figure 4.4: Peer-to-peer system classification from Koskela et. al [KKHY13].

In order to determine when and how a node movement is necessary, information about the distribution of load in the system must be monitored and nodes must be aware of their load in relation to other nodes' load. The keyspace may also be partitioned into ranges over which load information can be gathered [GBGM04]. In general, nodes with low load will move towards nodes or areas with high load to "steal" overburdened nodes work [KR04] (or vice versa). Once a node initializes its movement to a different nodeID, it must perform a data handoff for all or some of its data and establish a new set of links (unless the move has preserved the neighborhood relations of network nodes as in distributed balanced tables [GB03]). These movements are suitable for adapting to a statically skewed data distribution within the keyrange, but the additional costs for load balancing make it impracticable for a highly dynamic scenario with both high levels of churn and changes in nodes' target or maximum acceptable load. The extra dynamics that would be introduced by constant node movements would be costly while jeopardizing the system's robustness.

4.3.2 Hierarchies

Artigas et al. [ALS07] considered a hierarchical DHT to be "a collection of disjoint clusters of peers defining together a k-layer architecture", which expresses the most important attribute of a hierarchical structure: that a peer performs or routes differently, depending on which hierarchy layer it associates itself. Moreover, a node may be involved in many hierarchical layers, but it must function at a given time or for a given operation from only one of those layers. Artigas et al. go on to differentiate between homogeneous design and superpeer design, which we refer to as horizontal and vertical design, respectively: In horizontal hierarchies, nodes play symmetric roles while in vertical hierarchies, hierarchy layers can only be traversed with the help of gateway (i.e. superpeer) nodes. Thus, horizontal and vertical refer only to nodes' ability to switch between hierarchy layers, i.e. the links that they maintain between layers.

Recall that DHT hierarchies are typically used to express nodes' heterogeneity in either their available resources (often called capacities) or their group associations. In the first case, nodes

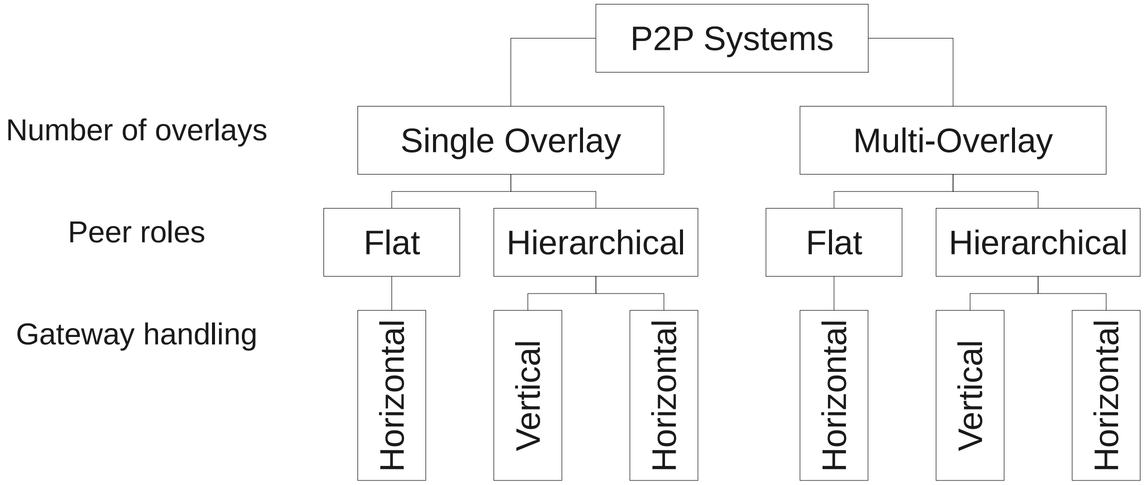


Figure 4.5: Structural characterization of structured peer-to-peer systems, depending on the number of autonomous overlays involved, whether the peers' roles vary, and whether nodes traverse various overlays/hierarchy layers via gateway nodes.

with more resources are given more load and responsibility within the DHT, often as superpeers or cluster-heads. Where superpeers are used, super nodes are fully responsible for performing lookup routing and for a large portion of maintenance, thus neglecting the varying nuances of nodes' resource availabilities. In the second case, nodes that belong to otherwise disparate groups such as companies or university departments collaboratively form a single DHT, although the groups may each maintain an independent overlay, resulting in a multi-overlay system. Superpeers often act within one group (or overlay) as gateways to communicate with the global network while the remaining group nodes build a local network (i.e. in a vertical structure). These structures use routing protocols that tend to route lookups as far as possible within one group before forwarding them on via a superpeer to a different group. While these grouped hierarchical DHTs are not as relevant for our specific use cases and goals as the resource-based approaches, they are described here to provide an overview of existing hierarchical approaches.

But first, recall the peer-to-peer taxonomy described in Table 3.1, which included a category *number of overlays* as introduced by Koskela et al. [KKHY13]. Koskela et al. performed a survey of peer-to-peer systems which use some variation of group management strategy, identifying the main challenges for these systems (as discussed in Chapter 3) and discussing various approaches according to the P2P characterization shown in Figure 4.4. This figure includes only a portion of the total taxonomy, showing the new *number of overlays* category with the older structural categories *index centralization* and *network structure* introduced by Androutsellis-Theotokis and Spinellis [ATS04].

While Koskela's *number of overlays* category is especially important for hierarchical overlays, it fails to provide consistency with the remainder of the taxonomy. While multi-overlay systems may be vertical or horizontal, they may also be flat (i.e. not hierarchical) or vary in their index centralization and/or network structure similarly to single overlay systems. Moreover, hierarchical structures are not restricted to single-overlay systems and can also be vertical or horizontal on single-overlay systems. Thus, one additional category is necessary for the structural characterization as flat or hierarchical (*peer roles*) and hierarchical systems can furthermore be classified as horizontal or vertical (*gateway handling*). These two additions to Brands and Karagiannis's taxonomy are shown in Figure 4.5, and together with the taxonomy from Table 3.1 give us nine categories. Note that a flat overlay cannot be vertical, since vertical gateway handling implies a variation in nodes' roles between gateway nodes and non-gateway nodes, thus implying a hierarchy. Note that the approaches discussed in this work are both hierarchical and flat as well as vertical and horizontal.

The hierarchical approaches discussed in this section are characterized in Table 4.3 (ordered by year of appearance), with regards to the relevant taxonomy categories concerning the number of overlays, the peer roles (hierarchy), and the gateway handling (vertical/horizontal) in addition to which (if any) existing DHT they were built on, use of location and resource awareness, and use of node movements and virtual nodes. Of the existing hierarchal DHTs for group associations, HIERAS from Xu et al. in 2003 [XMH03], an approach from Garces-Erice et al. from 2003 [GEBF$^+$03], Coral from Freedman and Mazieres in 2003 [FM03], SkipNet from Harvey et al. in 2003 [HJS$^+$03], Canon in G Major from Ganesan et al. in 2004 [GGGM04], an approach from Mislove and Druschel from 2004 [MD04], Cyclone from Artigas et al. in 2005 [ALAS05], and LIP from Risson et al. in 2005 [RQMH06] are several well known approaches which we start with. The remaining approaches - expressway by Xu et al. in 2003 [XMK03], SmartBoa from Hu et al. from 2004 [HLZ$^+$04], SuperPastry from Castro et al. in 2005 [CCR05], HONet from Tian et al. in 2005 [TXZ$^+$05]. multiple superpeer-child approaches from Zoels et al. in 2006 to 2009 [ZDK06, ZDK07, ZHDK09], a hierarchical Bamboo from Tian et al. in 2009 [TWS$^+$09], and SkewCCC++ from Bienkowski et al. in 2010 [BBKK10] - all use some sort of resource awareness.

Groups

HIERAS [XMH03] and Canon in G Major [GGGM04] are identical in their basic form, can be applied to any DHT, but have been built on Chord - Canon's implementation of Chord is called Crescendo - and use a location aware hierarchy with many layers. Each node participates in some Chord ring at every layer, with the sparsest and most topologically close Chord rings in the bottom layer which are merged in successive upper layers so that the top layer consists of all nodes. While HIERAS uses a distributed binning scheme via network latencies to landmark nodes, Canon assumes that nodes are already separated into logical (and location aware) groups such as university departments. Nodes share a single keyspace and establish a fully linked local DHT in addition to links in each layer, thus HIERAS and Canon's hierarchies are both horizontal. Lookups are routed first to the closest (i.e. preceding) node within the initiating node's lowest (most local) layer before being routed in the successively higher, more global layers.

Coral [FM03] is introduced based on Chord or Kademlia, but can also be applied to other DHTs, and uses distributed sloppy hash tables (DSHT) in which caching plays a central role. Cluster overlays are built based on proximity awareness and three layer "regional layer", "continental coverage", and "planet-wide cluster" are used for varying degrees of proximity. Round trip times within clusters are monitored to determine when a cluster has become too large and should be split. Data is input into the responsible nodes in each layer, such that lookups are performed in the initiating regional layer before moving up to a higher layer.

SkipNet [HJS$^+$03] generalizes Skip Lists [Pug90] for a DHT which uses both a name ID and a numeric ID naming scheme simultaneously. While the numeric ID resembles a hashed value, the name ID represents a node's domain, for example *databases.tuilmenau.de*, with varying levels of abstraction. With these joint naming schemes, SkipNet aims at achieving both content and path locality, meaning that data can be stored close to a given location or within a given group and lookup paths are local whenever possible, leaving groups only when necessary. Thus, domains can be defined in which data is stored and searches can be performed within domains, which the authors call constrained load balancing, since load is balanced in a random fashion within the (sub)domains. However, individual nodes' resources are ignored, and the use of virtual nodes is assumed for nodes with homogeneous resources and overlay failures are assumed to be primarily domain related (e.g. a broken connection behind one ISP).

Mislove and Druschel's [MD04] approach combines independent DHTs whose nodes may be separated from the global network while using varying overlays. Each node that is not behind a firewall or network address translation that prevents it from direct communication joins the global overlay. Each sub-network has its own ringID for routing, and these gateway nodes are responsible for passing queries between sub-networks using Scribe application multicast trees. Lookups are performed with a (key, ringID) pair, and the global overlay of gateway nodes stores (key, ringID) pairs that are looked up when the ringID of a given key is not known. The global overlay does not use location awareness, but it may be present in the independent sub-networks' DHTs.

Cyclone [ALAS05], whose Chord implementation is called Whirl, and also may be used for various DHTs, also uses node identifiers with prefixes and suffixes similar to SkipNet. However, the suffixes are part of the keyspace keys used for cluster identification (for example, 128 bit keys with the last three bits used for the cluster identification) and the nodes share a single keyspace, providing automatic load balancing between the clusters. Clusters are merged similar to HIERAS and Canon.

LIP [RQMH06] was actually designed for internet telephony for saving contact information, aiming to handle node and data location movements between various clusters while supporting both short and long lived nodes. It uses multiple hierarchy layers which are connected via gateway nodes whose addresses are stored and replicated within the respective DHT. While DHTs at various hierarchy levels are built using location awareness by grouping together nodes, for example, that are behind a single ISP (as in SkipNet), routing is aided by caching lookups along their paths.

Garces-Erice et al. [GEBF$^+$03] present a grouping hierarchy that, in contrast to the other presented approaches, takes node resources into account. Nodes of various organizations are organized into arbitrary, autonomous overlays which choose one stable node as a superpeer. Together, the superpeers form a global DHT and are expected to have low failure probability. Superpeers must recognize which keys belong to their clusters and use this information to route lookups between clusters.

Resources

Xu et al. [XMK03] present an approach to resource awareness that builds "expressways" of strongly connected nodes through the network that form an extra hierarchical layer, with nodes self-determining whether they are strong and central enough to join the expressway layer. Expressway nodes have many links to other expressway nodes while normal nodes have only local links and links to nearby expressway nodes.

In SmartBoa [HLZ$^+$04], nodes are divided into up to 128 resource based levels, with each node determining its level based on its current load and increasing or decreasing its level if it has too much load or can handle more load. Nodes are arranged in a horizontal hierarchy and nodes in different levels vary primarily in the number of links they establish, with a level k node maintaining approximately $N/2^k$ links. Thus, routing tables have no significant upper bound and nodes may maintain up to N links, i.e. be connected to every other network node. A multicast algorithm is used for joins, leaves, and routing table updates instead of periodic probing, so that SmartBoa has limited applicability in highly dynamic scenarios.

SuperPastry and HeteroPastry [CCR05] were introduced for comparison with each other and unstructured overlays for heterogeneous networks. SuperPastry uses a vertical hierarchy with strong nodes as superpeers and weak nodes as children, while HeteroPastry uses a flexible choice of link nodes (similar to DHash++ [DLS$^+$04]) to establish links to nodes with high resource levels. HeteroPastry thus builds a flat overlay in which nodes' links are relatively strong, also enabling nodes to specify a maximum in-degree depending on nodes' capacities (similar to the varying out-degrees in SmartBoa). While the authors use a simple scalar for expressing nodes' resource levels and choose links based only on a single resource, they note that "it is possible to design neighbor selection functions that combine several capacity metrics and even network proximity."

In HONet [TXZ$^+$05], based on De Bruijn networks [LKRG03], nodes organize into location aware clusters that each use their own overlay and keyspace (similar to HIERAS and other group hierarchies). Each cluster has a stable node as cluster head which together form a core (global) overlay. Each node is identified via a cluster ID (CID) and a member ID (MID, the node ID within its cluster) and lookups can only be performed with a (CID, MID) pair. Nodes restrict their degree based on their capacity and network conditions, although (similar to other approaches) this idea is not described formally. Each node with adequately high capacity establishes shortcut links to physically close nodes from other clusters. A lookup to (CID, MID) is first routed to a "reflector" of CID within the local cluster which is responsible for maintaining information about how to route to CID. Then, using this information, it is routed to either a node within its local cluster with a shortcut to cluster CID or to the core overlay via the root node.

Zoels et al. have suggested multiple approaches that take node capacity into consideration to optimize a standard superpeer-child architecture, where superpeers form an overlay ring and each child node is connected only to its respective superpeer [ZDK06,ZDK07,ZHDK09]. They begin by showing that while the centralization of a superpeer-child architecture increases the risk of overloading superpeers, it decreases the overall system costs [ZDK06]. The distribution of leaf nodes to supernodes in order to balance load between supernodes is then considered in [ZDK07]. An optimal superpeer:total-peers ratio is sought in [ZHDK09] with consideration to nodes' load factors, i.e. the amount of load nodes have compared to the amount of load nodes are willing to accept.

A hierarchical adaptation of the location aware DHT Bamboo [RGRK04] (Bamboo is an extension of Pastry described in Section 4.4.1) uses clusters that are connected via superpeer cluster-heads [TWS$^+$09]. Nodes are divided into ordinary (weak) nodes, superpeers (strong nodes), and index peers (resource strength between weak and strong). Ordinary peers maintain connections to their superpeers only, superpeers form a global overlay for inter-cluster lookups, and index peers store records of ordinary nodes' information.

Another more recent approach, SkewCCC++ [BBKK10], focuses primarily on protection from adversaries and assumes bandwidth as the only resource. A node that can communicate with $3 \cdot \ell$ neighbors is assigned a resource strength (level) of ℓ. Based on a hypercube, the higher a node's strength is, the more dimensions the node has links in. Each node also hosts as many virtual nodes as its strength, but these virtual nodes are treated as a single entity for maintenance and routing purposes. However, load balancing on node or data insertion and deletion is costly and requires whole subsets of the system nodes to choose new nodeIDs, rendering this approach unsuitable for highly dynamic scenarios.

4.4 Location Awareness

While the original DHTs route efficiently, most were not designed to utilize location information (see Table 4.1). Proximity-awareness has garnered interest in many areas related to DHTs, including caching and replication protocols, hybrid overlays which combine structured DHTs with proximity-based unstructured networks [EDPK09, MBK07], and DHTs' overall designs. The various approaches to integrating location awareness into a DHT's design can be characterized into three categories, as suggested be Gummadi et al. [GGG$^+$03] and similar to the slightly earlier terminology from Castro et al. [CDHR02]:

PIS Proximity Identifier Selection: the selection of a node's nodeID depends on, among other things, the node's physical position (Mithos [WR03], SAT-Match [RGJZ04]).

PNS Proximity Neighbor Selection: nodes select links based on their distance in the physical space (Pastry [RD01], DHash++ [DLS$^+$04]).

PRS Proximity Route Selection: each routing hop for a lookup is chosen based not only on the destination key, but also on the physical locations of the current node, the current node's links, and/or the destination key. (Tapestry [ZKJ01]).

Note that physical location and physical distance can be viewed from many perspectives, for example as geographic locations, virtual positions based on latency between nodes, or the latency between nodes directly. Each of these proximity aware approaches comes with its own inherent strengths and weaknesses, making their applicability dependent on the network they are planned for.

PIS impedes network scalability by non-uniformly distributing keysrange sizes to nodes if the nodes are not uniformly distributed in space. If, for example, many nodes join the network in a small physical area, they are likely mapped to a small portion of the total keyspace. This results in small keyranges for these nodes, but potentially higher query and storage load overall, increasing the load on nodes from other physical areas. Thus, load balancing must be achieved through other optimizations. Furthermore, in many networks, nodes which are physically close have correlated failure probabilities, for example in wireless networks when base stations fail. However, failures of direct successors can make system recovery more difficult and jeopardize the network's ability to successfully route messages. These drawbacks were especially important in the design choices made during this work and discussed in Section 2.3. On the other hand, PIS can effectively provide a network that reflects the actual physical network, making geographic routing more effective and eliminating the high latency that is otherwise incurred by a lookup's last routing hop.

PNS may cause higher search overhead for finding suitable links that are as close as possible or cause an imbalanced routing load due to central nodes with more in-links. However, these issues can be worked around by, for example, piggybacking network and node information on network messages or setting caps on the maximum number of in-links a single node will allow. On the other hand, PNS can provide many nearby links, so that a significant portion of routing and maintenance can be performed with close nodes and has no effect on the system's scalability.

PRS affects only routing paths, also potentially causing an imbalanced routing load. However, the most difficult topic to address is the tradeoff between routing paths' number of hops and physically traversed distances. The next-hop in routing is chosen to a node that may be closer in the physical space than the link otherwise chosen by greedy key-based routing, and is thus not necessarily closest in the keyspace. Depending on a DHT's structure, this can increase the number of hops and make upper bounds on routing complexity more difficult to determine. PRS has the advantages of PNS, without the extra overhead to find good nodes, but has been reasoned (and shown for Pastry) to perform less effectively than PNS [GGG+03].

As discussed in Section 2.2, the effectiveness of location integration - i.e. PIS, PNS, or PRS - depends on how location is defined and used, as characterized by the location dimensions L.1 to L.5. Noteworthy approaches from implementations and literature, with location expressed (L.1) as either (i) actual physical location or (ii) relative location in respect to other nodes, location notated (L.2) as either coordinates, bins, or graphs, and location determined (L.3) in a variety of fashions, include:

(i) the physical position of a node given by its GPS coordinates,

(i) virtual coordinates calculated through the triangulation of latency to landmark nodes whose GPS coordinates are known (e.g. Global Network Positioning, GNP [NZ01]),

(ii) virtual coordinates built using latency between node pairs (e.g. Practical Internet Coordinates, PIC [CCRK04]; Vivaldi [DCKM04]; geographic routing [RRP+03]),

(ii) graph based internet distance information that expresses actual node links, gathered locally or centrally (e.g. IDMaps [FJJ+01]), or

(ii) binning schemes that group nearby nodes into single bins (e.g. [RHKS02]).

So, in addition to the range of possibilities for integrating location awareness into a DHT, there is clearly a broad interpretation of what location is. Although location is approached differently, we will see in Section 4.6 that the optimization goals for DHTs, as expressed through the used evaluation measures, are often identical.

The following two sections look at two variations of location aware DHTs: general proximity aware approaches (often extensions of the original DHTs from Table 4.1) and approaches specifically designed for mobile ad hoc networks.

4.4.1 General Location Aware Approaches

Location awareness started to be applied to the original DHT ideas shortly after their introduction, and new DHTs with the intent of reducing latencies or cross-network traffic emerged. Shortly after its introduction, Castro et al. [CDCR02, CDHR02] evaluated Pastry's location awareness properties, concluding that location awareness can be achieved in protocols such as Pastry, Chord, and Tapestry without significant changes in their load balancing and routing quality. They offer a discussion of the evaluation measures which offer meaningful feedback about the effects of location awareness, as mentioned later in Section 4.6. Ratnasamy et al. suggest a binning scheme for integrating location awareness into DHTs, unstructured overlays, and server selection [RHKS02], using PIS with CAN for the DHT variation to achieve proximity awareness.

Gummadi et al. [GGG$^+$03] also offer a discussion of possible evaluation measures, while examining the effects of different DHT routing algorithms' "geometries" (i.e. the structure of their overlays and how it is used to route) on their robustness and location awareness. They explain how DHTs can provide algorithmic flexibility for their neighbor and/or route selection by allowing a choice for links and/or next hop decisions after the foundational overlay and routing have been chosen. They furthermore examine the degree of algorithmic flexibility present in tree (PRR), hypercubes (CAN), rings (Chord), butterfly (Viceroy), XOR (Kademlia) and hybrid (Pastry) geometries. Algorithmic flexibility for link choices can be obtained when links are not necessarily determinate and can be chosen from a keyrange with multiple nodes. For Chord, for example, node x's links could be chosen not as the first successor of the keys $x_{ID} + 2^i$ but rather from the keyrange $[x_{ID} + 2^i, x_{ID} + 2^{i+1}]$. An evaluative comparison of proximity aware tree, hypercube, ring, and XOR geometries leads to the conclusion that PNS is significantly more important than PRS for decreasing lookup path distances (latency), while the overlay geometry is important only with respect to how well it implements PRN and PRS (i.e. how much algorithmic flexibility it offers). Ultimately, they found the ring structure to offer the highest level of flexibility and speculated that the ring structure may be superior for proximity awareness.

Jain et al. [JMW03] also suggest proximity aware variations for CAN, Chord, and Pastry that they compared with two measurement-based overlays in regard to the relative delay penalty (RDP), link stress, and load balancing. While they offer a discussion of the significance of their evaluation measures, they use CAN and Chord with PIS and PRS, while Pastry uses PNS only, making the results difficult to reasonably compare. Bamboo also explores optimization of Pastry with a high churn scenario by applying several PNS techniques, reactive vs. periodic recovery for failures, and timeout calculations for lookups [RGRK04].

Motivated by the development of the Cooperative File System (CFS), Dabek et al. [DKK$^+$01] introduced a PRS variation of Chord used by their DHash system, in which routing hops are chosen with the help of link latency information. This approach was further extended to DHash++ [DLS$^+$04] which uses PRS and PNS along with Chord. Proximity is established using Vivaldi [DCKM04] to generate virtual network coordinates for each node that reflect the underlying network latencies via a spring-mass estimation. Each node uses these virtual coordinates to chose its physically (in terms of latency) closest neighbor within keyrange blocks. DHash++ saves each data block in 14 erasure-code fragments, from which any seven are required to reconstruct the block. The authors discuss the effects of, among other things, caching data, recursive vs. iterative routing, PNS, and replication and server selection on the system's latency.

Kaune et al. [KLKP08] introduce a proximity aware version of Kademlia that uses PNS and PRS much in line with DHash++ and other approaches. The authors provide a discussion of multiple underlay metrics which can be used to express and integrate location, in other words, several variations for the location dimensions L.1-L.3.

While the above approaches focused on PNS and PRS, the following have applied PIS. SAT-Match, which was evaluated on CAN, uses TTL-k flooding to find nodes with the closest round trip times, and nodes then "jump" to their closest neighbors by adjusting their nodeIDs [RGJZ04]. Mithos [WR03] uses a protocol similar to Vivaldi in order to establish node locations, from which logical quadrants are formed so that nodes on bordering quadrants establish links. eQuus [LSW06] uses a hypercube topology in which nodes within a given proximity to one another form cliques which are then connected. Items are replicated to all nodes within a given clique to provide fault-tolerance.

4.4.2 Mobile (Ad Hoc) Networks

In mobile ad hoc networks (MANETs), each overlay hop incurred from routing a DHT lookup may require multiple underlay hops on DHT nodes since DHT nodes are also responsible for underlay routing. Furthermore, the use of overlay hops for which no underlay route is known often results in broadcast messages to determine a route. This clearly causes unnecessary overhead since a single broadcast message destined for an intermediate overlay hop may very well reach the given lookup's ultimate destination. MANETs also comprise networks with high movement and churn rates, causing frequent changes in "good" routes and requiring highly dynamic protocols for overlay maintenance and data persistence. While MANETs are typically heterogeneous, DHT design for MANETs has primarily focused on minimizing the number of necessary underlay hops, thus indirectly reducing load on all nodes. Note that while the approaches described here were specifically designed for MANETs, they are often similar to the group hierarchy approaches from Section 4.3.2 which establish groups based on node location (i.e. clusters of nearby nodes) that are then connected via a hierarchy.

Generally speaking, DHTs in MANETs employ some combination of cross-layer PIS, PRS, and PNS, often using network layer (i.e. underlay route) information to augment overlay decisions and thus merging layers within the OSI model. Many of these overlays can be considered hierarchical, in part due to clustered structures. Many of the numerous MANET protocols are described below, and their approaches to location-awareness and heterogeneity, i.e. the use of PIS, PNS, PRS, hierarchical structure, node movements, and/or virtual nodes, are shown in Table 4.6.

GHT (Geographic Hash Table) [RKY$^+$02] is among the foundational work on sensornets with very similar goals to Ekta [PDH04], MADPastry [ZS05], and CHR [ARK$^+$05], which were among the first DHTs suggested for MANETs. In fact, GHT considers both nodes with restricted energy availability and mobile nodes. It uses geographic hashing to map and replicate data to a geographic area that corresponds to its own key and employs geographic routing to find short, efficient paths. Its resource awareness stems from the assumption that data, which is distributed according to where it is needed or where it was produced, is not necessarily uniformly distributed and will thus cause bottlenecks at certain nodes. A structured replication protocol is designed to distribute data from an overloaded bottleneck node to (geographically)

Approach	Applied to	PIS	PRS	PNS	NM	Hierarchy	RA
PIS CAN [RHKS02, RHKS02]	CAN	✓					
Gummadi et al. [GGG$^+$03]	PRR, Chord Kademlia, CAN		✓	✓			
Jain et al.	CAN, Chord	✓	✓				
[JMW03]	Pastry			✓			
Mithos [WR03]		✓					
SAT-Match [RGJZ04]	CAN	✓			✓		
DHash++ [DLS$^+$04]	Chord		✓	✓			
Bamboo [RGRK04]	Pastry			✓			
eQuus [LSW06]		✓					
location Kademlia [KLKP08]	Kademlia		✓	✓			
GHT [RKY$^+$02]		✓	✓	✓			✓
Ekta [PDH04]	Pastry		✓	✓			
MADPastry [ZS05]	Pastry	✓			✓	2 - CH	
CHR [ARK$^+$05]		✓	✓	✓			
hier. Pastry [YV11]	Pastry	✓			✓	2/3 - CH	
ROBUST [MPP10]	Bamboo (Pastry)	✓		✓	✓	2 - CH	
PNS-CHORD [CF05b]	Chord			✓			
[KKF06]				✓			
VRR [CCN$^+$06]				✓			

Table 4.6: Location aware DHTs. The approaches below the bold line are geared towards MANETS, NM stands for node movements for location awareness purposes (as opposed to for load balancing), and RA for resource awareness.

surrounding nodes, but increases query costs from $O(\sqrt{n})$ to $O(2^d\sqrt{n})$ (with $4^d - 1$ replicating nodes) and is thus better suited for frequent storage with infrequent querying [RKY$^+$02].

Ekta [PDH04], which is based on Pastry, uses underlay routing information to choose links (PNS - analogously to Pastry) and make overlay routing decisions (PRS). Furthermore, saved underlay routes are constantly updated by monitoring the forwarded overlay messages. MADPastry [ZS05], on the other hand, uses select landmark keys to delegate landmark nodes. Nodes form location-based clusters based on their physically closest landmark node, whose nodeID determines a joint nodeID prefix shared by all nodes in its cluster. Thus, each node's nodeID depends on its cluster's landmark node (PIS), and a new nodeID must be chosen each time a node changes clusters (i.e. when the node becomes physically closer to a different landmark node). Physical node movements thus lead to node movements within the key space, causing unnecessary overhead in highly dynamic mobile networks due to data-reallocation and overlay link maintenance. Furthermore, since each node stores the network's landmark nodes in its routing table, landmark nodes receive heavy routing load although they are chosen regardless of nodes' strengths.

CHR [ARK$^+$05] (Cell Hash Routing) uses geographic clusters by dividing the space into cells of a fixed size (PIS). The clusters act jointly as super-peers, with data stored, geographic lookups routed as in GHT [RKY$^+$02] (PNS, PRS), and lookups processed on a cluster scale, as opposed to a node scale. While CHR demonstrates strong proximity-aware attributes, its structure restricts its scalability for highly dynamic scenarios.

Cramer and Fuhrmann considered how to use a DHT ring as a routing structure in unstructured networks without routing capabilities, suggesting ISPRP [CF05a]. They also examined the practicality of PNS on MANETs, specifically a proximity-aware Chord PNS-CHORD [CF05b]. They defend Chord's adaptability to the physical network and suggest a hybrid approach to finding neighbors using PNS, flooding for neighbors in small index ranges, and caching nodes within a k-hop neighborhood for larger index ranges.

Kummer et al. introduced an approach that uses every underlay hop to reroute the current overlay hop to a more ideal node [KKF06]. They suggested a ring-based DHT in which nodes establish overlay links only to their direct logical successors and predecessors as well as their direct physical neighbors. Upon forwarding a lookup to the current destination hop along the underlay, each intermediate node checks whether one of its overlay links is closer to the final destination key. If so, the current destination hop is changed to this logically closer overlay node, creating a logical shortcut and thus decreasing the total network traffic and lookup hop lengths. Similarly, the network routing protocol VRR (Virtual Ring Routing) was also introduced as a blend of point-to-point and overlay routing, implementing key-based routing directly on the link layer [CCN$^+$06]. Nodes are organized in a ring and information about physical links and bi-directional routes between virtual links is stored at each node. Much like the approach from Kummer et al., optimizations to the overlay route are made in each underlay step.

Another alteration of Pastry/MADPastry was suggested by Yu and Vuong [YV11] for massively multi-player online games (MMOGs) over MANETs. Here, cluster heads are used for additional link information collection and dissemination, resulting in what can be seen as a 3-tiered hierarchical system. ROBUST [MPP10], based on Bamboo [TWS$^+$09], uses location-based clusters with super-peer cluster heads. A node's cluster is determined based on its physical proximity to the network's cluster heads and a node's nodeID's prefix is based on its cluster's cluster head. ROBUST also uses *proximity synchronization*, or a node movement (in the ID-space) to a closer cluster head after nodes' physical movements.

4.5 Resource and Location Aware DHTs

Of the approaches discussed here, six of them use some form of both resource and location awareness [XMK03, GEBF$^+$03, TXZ$^+$05, BBKK10, TWS$^+$09, RKY$^+$02], and are therefore of special interest for this work. However, each of these approaches also has significant boundaries which restrict their applicability, and only GHT [RKY$^+$02] has been evaluated with some respect to resources. GHT was designed primarily for stationary sensornets and uses PIS to distribute nodes and data keys in the keyspace, ultimately making node movements costly. Moreover, GHT does not explicitly consider varying levels of resources. This is especially evident in its evaluation, which takes only a maximum storage and message load into account, assuming that all nodes can handle a similar load.

The expressway approach [XMK03] and the approach from Garcés-Erice et al. [GEBF+03] both employ a simple binary approach to resources with superpeers responsible for clusters of weak nodes. They thus ignore nodes' varying resource availabilities, and were only evaluated for a small set of measures (see Section 4.6). While HONet [TXZ+05] employs a continuous scale for resources, it requires that the cluster ID of a given piece of data is known in order to perform a lookup, making global lookups either unfeasible or much more costly. For example, this information may be stored in the core overlay with extra clusterID lookups performed whenever the cluster ID is unknown. The hierarchical version of Bamboo [TWS+09] differentiates between three resource availability levels, but its description and evaluation are unfortunately very restricted so that neither its applicability nor its implementation are clear enough to justify consideration. Finally, in ScewCCC+ [BBKK10], tedious re-balancing is required for data inserts and deletions, rendering the system unfit for dynamic scenarios, and it lacks any evaluation with which to counter these assumptions.

4.6 Evaluation Measures

Although the various DHT approaches described thus far justify their development using widely varying challenges, deficits, and goals, they ultimately share the central goal of robust data storage, i.e. high data availability, along with the other peer-to-peer challenges described in Section 3.1. While these challenges can be addressed via the many DHT design issues and thus evaluated in many fashions, literature tends to focus on the following four questions for evaluation:

- Can lookups be successfully routed to their destinations?
- Are query delays within acceptable ranges?
- Are nodes overburdened with heavy load, causing bottlenecks or node failures?
- Is data adequately replicated to ensure that it is not lost due to node failures?

However, the literature discussed in this chapter has mostly ignored the last replication-related question, which has been addressed in replication specific work as discussed in Chapter 8. These questions have been evaluated using a diverse set of measures concerning the nodes in general, links, lookups, and maintenance which are summarized in Table 4.7. A very broad range of measures have been used due to the lack of standardized benchmarks for DHT evaluation. Ranges of implemented measures are grouped together to form the categories in Table 4.7. For example, the category *link latency* contains implemented measures that range from the costs associated with using a link [TXZ+05], to the latency of individual overlay hops [RFH+01], to the distribution of links within areas such as the local ISP, region, country, continent, and world [KLKP08]. More precisely defined, the lookup overlap refers to how much the paths of two lookups for the same key κ originating at two different nodes

Maintenance	latency		round-trip-times of maintenance messages
	load	bandwidth	bandwidth consumed by maintenance activity
		messages	number of messages sent and/or received for maintenance activity
Lookup	hop count		number of hops per lookup
	latency	time/distance	round-trip-time (or physical distance) of lookups
	stretch	RDP	relative delay penalty ratio of overlay latency to quickest physical path per lookup
		underlay	ratio of underlay distance to shortest possible underlay distance per lookup
		hop	ratio of overlay hops : fewest possible overlay hops per lookup
	failure	overlay	percentage of failed lookups
		underlay	percentage of dropped underlay hops or packets
	overlap		percentage of lookup routes that converge for similar lookup routes
Node	degree		number of links per node
	keyspace		percentage of nodes' maintained keyspace or total storage
	load	bandwidth	bandwidth used by node
		bytes	bytes sent/received by node
		messages	messages sent/received by node
		underlay msg	number of packets sent/received on underlay
Link	resources		strength/capacity of peers to which a node is linked
	latency		latency of a node's links
	load		load (messages, bytes, etc.) per linked node pair

Table 4.7: Measures used to evaluate DHTs.

overlap, while the lookup stretch refers to the ratio of the distance of the shortest possible end-to-end connection between an initiating node and a destination node to the distance of the total lookup path.

Each of the measures from Table 4.7 can be (i.e. has been for DHTs) interpreted to reflect how well one or more of the peer-to-peer challenges has been overcome. Figure 4.6 provides an overview of these relationships. The *security* and *grouping of information* challenges have been omitted, since they are not clearly addressed by any of the measures. Note that the measures are merely indirect indicators of specific attributes (i.e. challenges) and can often be interpreted to assess multiple related challenges. For example, a measure that is used to evaluate performance can also be interpreted to evaluate scalability by increasing the network size. Likewise, a measure used to evaluate resource management can also be interpreted to evaluate fairness. On the other hand, the measures can also be used with relation to one another to establish more complex measures. For example, in the comparative performance survey from Li et al. [LSM^{+}05], nodes' lookup latency (performance) was plotted against

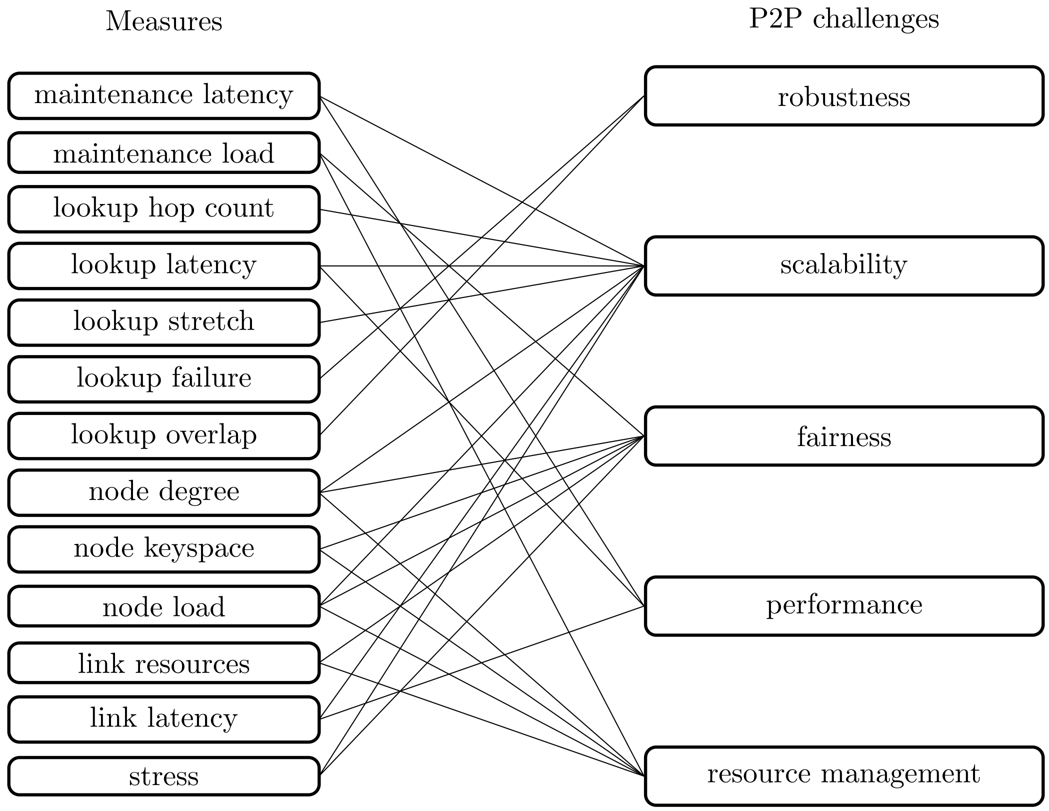


Figure 4.6: Mapping of measures to peer-to-peer challenges they help to evaluate.

nodes' bandwidth load (cost scalability).

In their survey on peer-to-peer search methods, Risson and Moors focused on robustness, especially its dependability and adaptability aspects [RM06]. They discuss how the measures for dependability are well established as the mean-time-to-failure (MTTF), mean-time-to-repair (MTTR), availability (combination of MTTR and MRRF), maintainability, and safety [RM06, Lap85]. It is thus surprising that none of these measures have established themselves as common DHT evaluation measures. The original DHTs focused on the basic construction of links, load balancing, and efficient routing while DHTs specializing in heterogeneous networks and MANETs shifted the focus to include the differences between nodes and the robustness of systems in the dynamic and/or heterogeneous systems. However, when examining the evaluation measures used by the original, the heterogeneous, and the location aware DHTs, there is no clear shift in measures as might be expected. Table 4.8 provides an overview of which approaches were tested with which measures. Measures used to validate the correctness of an implemented algorithm have been omitted from this table (for example, how many links nodes are able to find for their routing tables) as have mathematically analyzed measures (for example node degree). Approaches discussed above but not evaluated are also omitted. As one can easily see, the measures associated with lookups have been more popular in literature than the remaining measures.

How these evaluation measures, the specific scenario requirements from Section 1.3, the peer-to-peer challenges from Section 3.1, and the DHT design issues from Section 3.3 relate to

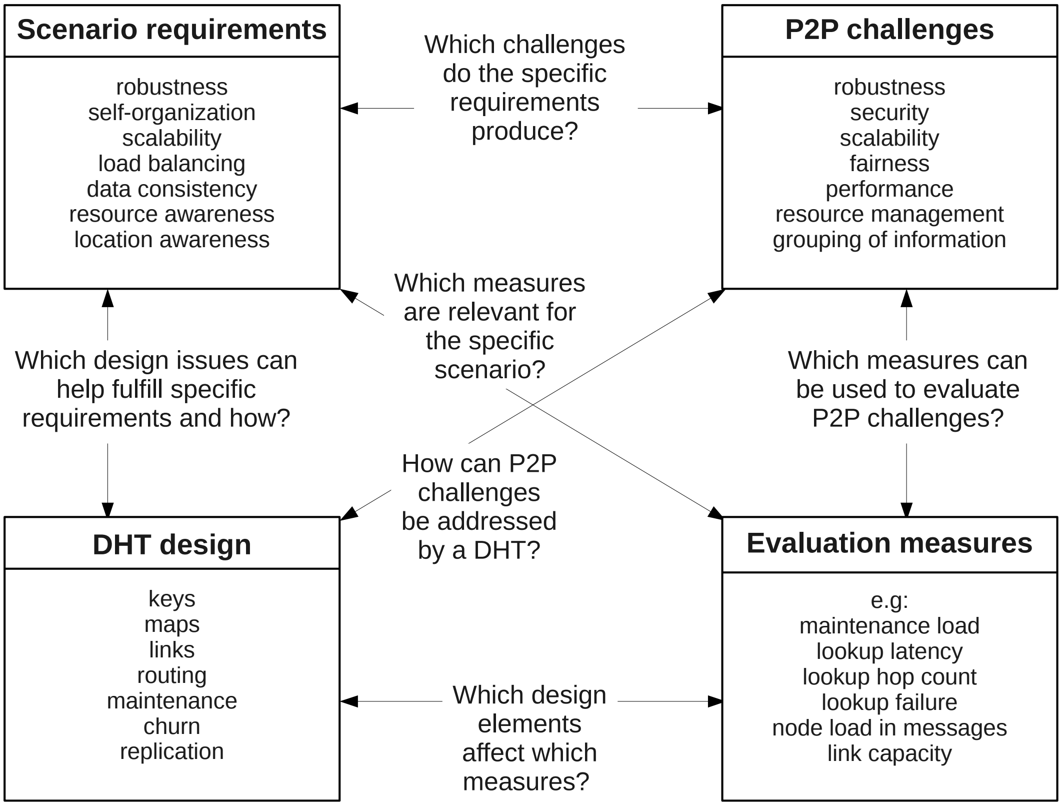


Figure 4.7: The relationship between each of four viewpoints from which the problem can be approached: general peer-to-peer challenges, specific scenario requirements, DHT design issues, and DHT evaluation measures.

each other is illustrated in Figure 4.7. For the further development of DHTs for our use case scenarios, we are specifically interested in the intersection of the scenario requirements with the other three issues. The relationship between the peer-to.peer challenges and the scenario requirements was discussed in Section 3.1 while the DHT design tradeoffs for the scenario requirements were explored in Section 3.3.3. We consider now the specific scenario requirements, in particular resource and location awareness, with respect to the evaluation measures. Location awareness has been a focus of every MANET application and is effectively measured with latency and stretch measures. Resource awareness, on the other hand, must be evaluated with respect to each node's individual resource level. Lookup failure or node load alone cannot express how well the resources of individual nodes are fairly used. Rather, the measures used to evaluate fairness (maintenance load, node degree, node keyspace, node load, link resources, stress), together with a differentiation between the nodes' various resource availability levels, can reflect how well resources have been taken into account. The resource levels established in Chapter 2 must thus, in contrast to previous literature, be integrated into the evaluation process.

		Maint.		Lookup					Node			Link		
	Approach	latency	load	hop count	latency	stretch (RDP)	failure	overlap fraction	degree	keyspace portion	load	resources	latency	load
original DHTs	CAN [RFH$^+$01]			✓	✓	✓							✓	
	Chord [SMK$^+$01]			✓	✓		✓			✓				
	Pastry [RD01]			✓		u								
	Tapestry [ZHS$^+$04]	✓	bw			✓	✓				bw			
	P-Grid [ACMD$^+$03, Abe01]		m				✓							
	Symphony [MBR03]		m		✓		✓				m			
	BATON [JOV05]		m											
hierarchical DHTs	HIERAS [XMH03]			✓	✓									
	expressway [XMK03]					✓								
	[GEBF$^+$03]			✓										
	CORAL [FM03]				✓									
	SkipNet [HJS$^+$03]			✓	✓	✓	✓							
	Canon [GGGM04]			✓	✓	✓		✓	✓					
	Cyclone [ALAS05]			✓										
	Y_0 [GS05]			✓					✓	✓	m			
	HeteroPastry [CCR05]				✓		✓		✓	✓	m	✓		
	HONet [TXZ$^+$05]			✓		✓							✓	
	Superpeer-child [ZDK06]										m			
	Superpeer-child [ZDK07]								✓		m			
	Superpeer-child [ZHDK09]				✓		✓				m			
	SkewCCC+ [BBKK10]			✓										

Continued on next page

Continued from previous page

		Maint.		Lookup					Node			Link		
	Approach	latency	load	hop count	latency	stretch (RDP)	failure	path convergence	degree	keyspace portion	load	resources	latency	load
location aware DHTs	[GGG+03]			✓	✓	h	✓	✓						
	[CDCR02]		m	✓	✓	✓		✓	✓			✓		
	PIS CAN [RHKS02]					✓								
	[JMW03]					✓					bw,p			✓
	SAT-Match [RGJZ04]			✓	✓	✓					b			
	DHash++ [DLS+04]				✓							✓		
	Bamboo [RGRK04]				✓		✓				bw			
	[KLKP08]				✓								✓	
	Mithos [WR03]			✓	✓									
	eQuus [LSW06]			✓		✓								
	GHT [RKY+02]						✓			✓	m			
	Ekta [PDH04]				✓		u				um			
	MADPastry [ZS05]						✓				m,b			
	CHR [ARK+05]			u										
	ROBUST [MPP10]		b	u	✓		✓							
	PNS-CHORD [CF05b]			u		u								
	[KKF06]			u			u							
	VRR [CCN+06]			u	✓	u	u				bw			
	comparison [LSM+05]				✓		✓				bw			

Table 4.8: Measures used to evaluate DHTs. Abbreviations used to classify more specifically how measures were used: m-message, h-hops, bw-bandwidth, b-bytes, u-underlay, um-underlay messages. p-pairs of source-destination nodes for forwarding.

4.7 Open Questions

This work's central question regarding the ideal system to meet the requirements developed in Chapter 1, particularly the resource and location awareness, gives rise to many other questions when considering previous literature. Location and resource awareness seem to pose a natural tradeoff, and have often been considered separately. However, if and how they impede each other can only be guessed. Can a distributed storage system gain resource awareness simply by integrating location awareness or vice versa?

Of the new resource and location awareness characterization dimensions introduced in Chapter 2, which, if any, actively affect how successful awareness can be implemented? For example, can a continuous scale for resource awareness better adapt to a dynamic system than a binary scale (R.4)? Or can the round trip latency between two nodes better decrease a system's total location-related costs than considering the number of hops between nodes (L.5)? Some dimensions - such as the resource awareness notation (R.5), with scalars, tuples, or functions - have typically been restricted in literature (here to scalars). How can these, or other, dimensions be implemented in new fashions?

Furthermore, the difference between the effectiveness of the newly introduced resource awareness approach characterizations RIS, RNS, RRS, and RSD remain unclear. There has been ample literature involving systems that use one or more of these approaches, but they have not been considered on a higher level similar to Gummadi et al.'s [GGG^{+}03] consideration of location awareness through PIS, PNS, and PRS. Which advantages and disadvantages does each approach contribute, which can be considered the "strongest" (similar to Gummadi et al.'s preference to PNS)?

However, these ambitious questions are not in the scope of this work. Rather, the focus here is placed on a more humble question with a primarily RNS (partially RRS) approach: Which structure is best for the developed requirements? Or, how do the variations in the taxonomy from Figure 4.5 influence the effectiveness of resource and location awareness? Moreover, how does the number of layers in a hierarchy (with one layer being effectively flat) generally effect the system robustness and specifically effect resource and location awareness?

In order to answer any of these questions, evaluation metrics must first be found that accurately reflect the desired robustness requirements. To date, there are no agreed upon evaluation metrics or processes for evaluating peer-to-peer systems, let alone benchmarks. A standardized framework, as in Chapter 2, is necessary in order to establish common conditions for any wide scope comparison. Furthermore, in the long term, only standardized evaluation metrics can provide relevant cross-approach or cross-study comparisons and both their use and their interpretation must be discussed to ensure their comparability.

Chapter 5

Resource and Location Aware DHTs

Consider now a Chord implementation run on servers and smart-phones alike: While servers have unlimited energy availability and large storage capacities, smart-phones have very limited energy and storage capacities. Since nodes' delays may vary greatly depending on their connectivity, fingers should clearly be chosen across low latency links (i.e. to "near" nodes). While virtual nodes or node movements could help to suitably distribute data, both fail to relieve weaker nodes with dwindling energy of costly maintenance and routing responsibilities. On the other hand, the integration of a super-peer structure with binary "have" or "have not" nodes invariably over-uses or under-uses the resources of the energy (un)restricted devices. Subsequent node failures lead to a drop in the overall network storage and computational capacity and thus compromise not only data availability and consistency but also the network's scalability. Without location awareness, lookups are routed blindly back and forth across the network, putting unnecessary strain on weaker nodes and increasing the delay and failure rates of lookups.

In this chapter, several structured location and resource aware peer-to-peer storage systems, i.e. DHTs, are developed in order to address several open questions. First, how do location and resource awareness effect each other and do they pose tradeoffs or can they actually support one another. Secondly, can a hierarchical or flat approach provide better joint awareness and what is the effect of multiple hierarchy layers. This question indirectly answers whether it is worth the effort for nodes to differentiate between their resources, i.e. strengths, on a finer scale than "have" and "have not"? While this chapter takes a structural approach, this alone is sufficient for neither data availability in a dynamic network nor location awareness when data is randomly distributed (i.e. scalably and balanced). Thus, approach specific resource and locations aware replication is developed in Chapter 8 to supplement the DHTs developed in this chapter.

In the following sections, one flat and one hierarchical novel overlay is presented, as well as a hybrid version of these two overlays and a cluster-based variation of the flat approach. While all of these approaches are more or less based on the flexible ring-topology Chord, they could conceivably be adapted to almost any other DHT. In the flat approach, each node chooses links and routes messages independent of its own resource availability, i.e. nodes have symmetric roles. Meanwhile, in the hierarchical approach, nodes establish links and route lookups depending directly on their resource levels, thus separating nodes into hierarchical layers.

In the flat approach, each node chooses for a given link that node within a given keyrange with the “best mixture” of high resource availability and near location, as defined variably by the system. This provides many opportunities to optimize and adjust the system by adjusting the relationship between resource and location awareness. The hierarchical approach, on the other hand, provides less room for adjustment with a more rigid structure which inherently removes load from the weakest nodes, which function as leaf nodes. Due to these obvious differences, we can expect the approaches to behave significantly differently in various scenarios. Hypotheses regarding their performance are formulated in Chapter 7, for example, weak nodes in the hierarchal approach could be expected to have longer lifetimes while the flat structure could be expected to adapt better to network changes.

5.1 DHT Foundations

Recall that for resource awareness in DHTs, RIS, RNS, RRS, and RDS approaches can be applied, similar to the original PIS, PNS, and PRS for proximity awareness. Gummadi et al. [GGG$^+$03] argue that using proximity identifier selection comes at the cost of scalability and load balancing, as might be expected from resource identifier selection. Since the random distribution of identifiers to nodes is a key source of DHTs' scalability and inherent load balancing, this work develops and compares approaches which use only random identifiers. Furthermore, virtual nodes and node movements, forms of PIS, are also excluded. Recall from Section 4.3.1 that virtual nodes and node relocation introduce additional maintenance overhead while making the basic assumption that weaker nodes have less storage space to contribute. We assume that lower resource nodes have an integral role in storing data and propose systems which distribute data evenly on all nodes while addressing heterogeneity of nodes through their maintenance and routing responsibilities. A replication mechanism for placing data is developed in Chapter 9 to augment these RIS and PIS-free approaches and distribute varying storage load when necessary.

The DHTs developed here are based on Chord [SMK$^+$01] in part because Chord is the basis of the location aware DHash++ which has a rather simplistic and adaptable structure, and in part because Chord has wide-spread popularity both in literature and in applications (see Chapter 4. While all of the following DHTs are based on Chord, they could also be adapted to many other DHTs. Analogous to Chord, consistent hashing [KLL$^+$97] is used to distribute keys to nodes. Each node x chooses a random (or hashed) nodeID x_{ID} from the binary key space $0 \dots 2^m - 1$, which is viewed as a ring with key values increasing in a clockwise direction. Each node positions itself at its nodeID on the key ring and establishes links to its immediate predecessor and successor as well as a *successor list* with its r nearest (clockwise) successors. The successor list makes repairs possible after unexpected node failures. Each key $\kappa \in \{0, \dots, 2^m - 1\}$ is assigned to the first node x whose nodeID x_{ID} is equal to or succeeds κ on the key ring. The asymmetric *key distance* from a node x with nodeID x_{ID} (or key) to a node y with nodeID y_{ID} (or key) is called the *key distance*:

Definition 5.1.1 (Key Distance)**.** *The key distance from x to y is the clockwise distance on* *the key ring from x_{ID} to y_{ID}:*

$$d_{key}(x,y) := y_{ID} - x_{ID} \mod 2^m.$$

Note that this distance is not symmetric ($d_{key}(x,y) \neq d_{key}(y,x)$).

5.2 Flat DHT - RBFM

The flat approach is called *RBFM* for Resource Based Finger Management [RB11], and can be easily used for almost any underlying DHT. Based on DHash++ [DLS+04], each node in RBFM chooses its links based on other nodes' key distances, physical distances, and resource levels. Each additional shortcut link (i.e. not successor or predecessor links) is called a finger and is chosen from a designated finger key-interval. From this interval, a node with the best balance of low physical distance and high resource level is chosen. Links are maintained at varying frequencies: links to strong nodes are maintained less frequently than links to weak nodes. Thus, weak nodes have fewer incoming links and maintenance to strong nodes is reduced, reducing the overall load for weaker nodes.

5.2.1 Finding Links

Similar to DHash++ and borrowing the terminology from Chord, each node views the keyspace as broken into finger intervals:

Definition 5.2.1 (Finger Interval)**.** *The i^{th} finger interval for node x with nodeID x_{ID} is*

$$B_{x,i} := [x_{ID} + 2^{i-1}, x_{ID} + 2^i)\; i \in \{1, 2, \ldots, m\}.$$

Each node x with nodeID x_{ID} chooses one finger $x.F[i]$ per finger interval $B_{x,i}$ for $i \in \{1, 2, \ldots, m\}$. The corresponding node that $x.F[i]$ points to is notated $x.F[i].node$. But while DHash++ chooses for $x.F[i].node$ the node with the smallest physical distance to x in each finger interval, RBFM chooses a node based on both its physical distance *and* resource level, or its *resource distance*. This resource distance is inspired by Vivaldi's [DCKM04] distance metric for network coordinates which uses an additional height dimension to distance a node from the entire network. Similarly, RBFM use a node's resource level to distance it from the entire network, ensuring that the lower a node's resource level is, the further it will be distanced from every other node. First, we define each node x's resource height x_h via a resource height function $h : \{0, 1, \ldots, l_{max}\} \to \mathbb{R}^+$ for some stretch constant $c > 0$:

$$x_h = h(x_R) := c \cdot (l_{max} - x_R),\; \ell \in \{0, 1, \ldots, l_{max}\}. \tag{5.1}$$

The higher the resource stretch constant is, the farther the node is thus distanced from the rest of the network using the following resource distance.

Definition 5.2.2 (Resource Distance)**.** *The resource distance between nodes x and y with coordinates (x_1, x_2), (y_1, y_2), resource levels $x_R, y_R \in \{0, 1, \ldots, l_{max}\}$, and resource heights $x_h = h(x_R)$ and $y_h = h(y_R)$ is:*

$$d_{res}(x,y) = d_{phy}(x,y) + x_h + y_h.$$

ProspectiveLinks[m-1]

NodeID	P.Dist.	R.Level	R.Dist.
$p_2 ID$	1.6	2	2.6
$p_3 ID$	1.4	1	3.4
$p_4 ID$	0.9	0	3.9
$p_1 ID$	4.1	3	4.1

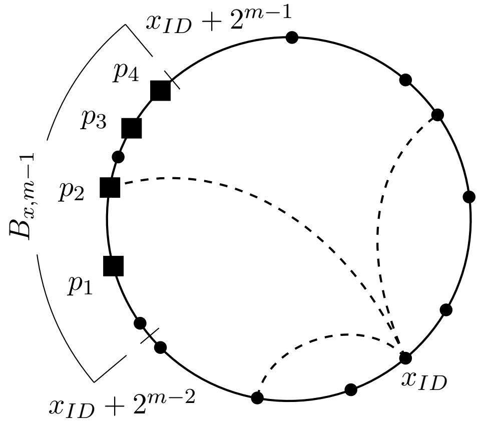


Figure 5.1: Key ring with six nodes in x's $m-1^{\text{st}}$ finger interval $B_{x,m-1}$, four of which x knows in its prospective links list (squares). A finger is established to p_2, the known node with the best resource distance (dependent on distance and resource level) to x.

In order to gain information about other nodes' resource distances, coordinates and resource levels are piggybacked on network messages. Each node x maintains a *prospective links* list with a list of nodes it has gained information about either directly or via piggybacked node hints. This prospective links list contains a list of the k best known nodes in terms of resource distance for each finger interval $B_{x,i}$, $i \in \{1, 2, \ldots, m\}$. Thus, at most k nodes in $B_{x,i}$ with the shortest resource distances to x are saved as (nodeID, physical distance, resource level, resource distance) tuples, for an upper bound of $k \cdot m$ entries. When receiving a message from sender y, node x uses y's coordinates and resource level to determine $d_{res}(x, y)$ and update its prospective links list accordingly (see Algorithm 3).

Algorithm 3 Updating prospective links list with $\leq k$ entries

```
procedure suggestProspectiveLink(nodeInfo)
    finger = getFingerInterval(nodeInfo.key)
    dist = getResourceDistance(nodeInfo.coordinates, nodeInfo.resourceLevel)
    if prospectiveLinkList.contains(finger, nodeInfo.key) then
        prospectiveLinkList.updateNode(finger, dist, nodeInfo)
    else if dist < prospectiveLinkList.getFarthestLinkDistance(finger) or
          prospectiveLinkList.size(finger) < k then
        prospectiveLinkList.addNode(finger, dist, nodeInfo)
        while prospectiveLinkList.size(finger) > k do
            prospectiveLinkList.removeFarthestLink(finger)
        end while
    end if
end procedure
```

Each node x maintains a *finger table* with one finger $x.F[i]$ in each $B_{x,i}$ for $i \in \{1, 2, \ldots, m\}$: if *prospective links* contains at least one entry for $B_{x,i}$, then the entry with the smallest resource distance is contacted with a finger request (see Figure 5.1); otherwise, the owner (i.e. successor) of key $x_{ID} + 2^{i-1}$ is contacted as in Chord (see Algorithm 4). To ensure that prospective links are up-to-date and alive, an entry is deleted as soon as it is used for a finger request. Thus, it cannot be used twice without confirmation that it is still alive, and prospective links are periodically cleared of old entries. The prospective links list entries are continually updated with fresh node information, so the network automatically adapts to changes in node resource levels or coordinates. Note that if there is a finger interval that contains no node, then multiple fingers will point to the same node, as in Chord. On the other hand, if there is at least one node in a finger interval $B_{x,i}$, then $x.F[i]$ will point to a node in $B_{x,i}$.

Algorithm 4 Establishing and maintaining fingers 1 to $m-1$

```
procedure MAINTAINFINGER(finger)
    lookupKey = myKey + getOffset(finger)
    if prospectiveLinkList.size(finger) > 0 then
        listEntry = prospectiveLinkList.getClosestEntry(finger)
        lookupKey = listEntry.key
        prospectiveLinkList.removeUsedEntry(listEntry)
    end if
    sendLookup(lookupKey)
end procedure
```

As we will see later, the larger i is, i.e. the farther away in the keyspace the finger is, the larger the finger interval is and the higher we expect $x.F[i]$'s resource level to be. This means that high resource level nodes tend to have more incoming fingers than low resource level nodes.

5.2.2 Routing

Lookup routing is performed greedily, identical to unidirectional routing in Chord: A node x which needs to lookup a key $\kappa \in \{0, \ldots, 2^m - 1\}$ first checks if it is the owner of κ, i.e. if κ is between x's predecessor and itself. If this is not the case, x forwards the lookup to the closest predecessor of κ in its routing table (including its successor list and its own nodeID x_{ID}). If x is the closest predecessor itself, then the key is maintained by x's successor, and the routing is completed after one hop. Figure 5.2 demonstrates how a message might be routed, with the hops becoming increasingly small.

Since fingers are not deterministically defined in this approach, allowing fingers to be spaced more irregularly, the expected (and worst case) number of hops which are necessary to locate a key is higher than in Chord. However, this increase can be expressed as a constant factor, leaving us with the same $O(\log N)$ routing complexity as in Chord. In fact, simulation results in Chapter 7 show that the difference in routing lengths is in fact negligible. Note that the term *with high probability* is used here to express a probability $\geq 1 - 1/N$.

Theorem 5.2.1. *Given a network with N nodes, with high probability, a message is routed from any node to the successor node of any key in $O(\log(N))$ hops.*

Proof. Assume that a node x is to forward a message to the node y responsible for key κ, and let p be κ's immediate predecessor node. We consider how many hops are necessary to reach p. Let p be in x's i$^{\text{th}}$ finger interval $B_{x,i} = [x_{ID} + 2^{i-1}, x_{ID} + 2^i)$. Then either:

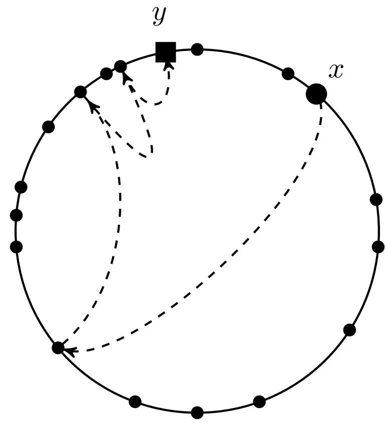


Figure 5.2: Greedy routing from x to y in four hops. The first hops cover a long key distance and can use stronger links. The latter hops become increasingly short and weak, since the smaller finger intervals offer fewer nodes to chose from.

- $x.F[i]$ is a predecessor of p, and forwarding to this finger reduces $d_{key}(x,p)$ by at least 2^{i-1}, or
- $x.F[i]$ is a successor of p. Since p is a successor of the key $x_{ID}+2^{i-2}$, $x.F[i-1].node$ is in the interval $B_{x,i-1}=[x_{ID}+2^{i-2},x_{ID}+2^{i-1})\subset[x_{ID}+2^{i-2},p_{ID}]$. Forwarding to this finger reduces $d_{key}(x,p)$ by at least 2^{i-2}.

We see that the key distance $d_{key}(x,p)\leq 2^i-1$ is reduced by a factor of at least $2^i/2^{i-2}=1/4$ for each forwarding. Thus, we are within one key of p after at most $m/\log(4/3)$ steps, which we see by assuming that we are the maximum of 2^m keys away from κ and that each step decreases this distance by 3/4:

$$\begin{aligned}2^m\cdot\left(\frac{3}{4}\right)^{m/\log(4/3)} =&2^m\cdot\left[\left(\frac{3}{4}\right)^{log(3/4)}\right]^{-m}\\ =&2^m\cdot 2^{-m}\\ =&1.\end{aligned}$$

Analogously, after $\log N/\log(4/3)$ steps we are within $2^m/N$ keys of p:

$$2^m\cdot\left(\frac{3}{4}\right)^{\log N/\log(4/3)} =2^m\cdot\left[\left(\frac{3}{4}\right)^{log(3/4)}\right]^{-\log N} =\frac{2^m}{N}.$$

Considering the consistent hashing used to generate nodeIDs and assuming that nodes are uniformly distributed in the keyspace, with high probability there are no more than $O(\log N)$ nodes in a $\log N/\log(4/3)$ sized keyrange. Thus, p can be reached within another $O(\log N)$ hops using direct successors. So p, and with it y, are reached in at most $O(\log N)$ hops. □

5.2.3 Link Maintenance

Given a dynamic network with frequent node joins, failures, movements, and changing resource levels, links must be updated on a regular basis to uphold the network's routing characteristics.

Links to nodes which have failed or no longer have the minimum resource distance in a finger interval must be reassigned; links to newly joined nodes must be established. Note that a node's outgoing fingers do not change when its resource level changes, but its incoming links most likely will since its new resource level will eventually be updated in other nodes' prospective links lists.

Since we correlate a node's resource availability with its reliability, we choose the frequency with which a finger f is updated depending on $f.node$'s resource level: Fingers to high resource level nodes require updates less often. This reduces the maintenance load for both low resource nodes, which initiate link maintenance, and high resource nodes, which respond to the link maintenance. Link maintenance is performed on a finger f after a time interval which depends on a reference interval t_{ref} and the resource level $f.node_R$:

Definition 5.2.3 (Finger maintenance interval)**.** *The interval between finger maintenance messages sent by node x to finger f is determined by the* finger maintenance interval*:*

$$g : \{0, 1, \dots, l_{max}\} \to \mathbb{R}^+$$

One possible finger maintenance interval which is dependent on a resource level ℓ is, for example, $g(f.node_R) = t_{ref} \cdot (f.node_R + 1)^2$. With this function, a finger with resource level 0 would be updated after the interval $time_{ref}$ while a finger with resource level 3 would be updated after the interval $16time_{ref}$. In later chapters, this example is referred to as a quadratic finger maintenance interval, but linear or constant examples are used as well. Thus, the function $g(\ell)$ provides an opportunity to tune the network's degree of robustness and maintenance overhead. While routing robustness and maintenance overhead usually imply a direct tradeoff, the dependence of the finger maintenance interval on nodes' resource availabilities softens this tradeoff by decreasing the maintenance overhead of specific, but not all, links. However, this only improves robustness when a node's failure or reliability is reflected by its resource level.

5.2.4 Node Joins and Failures

In order to join the DHT, a node x must have valid network coordinates, choose a nodeID and resource level, and contact one participating node. Once x has established links to its immediate predecessor p and successor s on the key ring, p and s send their prospective links lists to x, which x uses to initialize its own list, and corresponding keys are transfered from s to x. The node x continually updates its prospective links and periodically performs finger maintenance (see Algorithm 4) to establish and maintain its fingers. Node failure is handled as in Chord, but extended to remove a prospective link once its node is recognized as failed.

5.2.5 Adaptability

Since nodes consult their prospective links lists every time finger maintenance is due, changes in fingers' resource distances are continually reflected in the overlay's structure. When stronger nodes appear and are observed in a finger interval, they are added to the prospective links list and requested as fingers upon maintenance. When existing fingers become weak, their resource distances are increased, and since their entries in the prospective links list are updated, they become less attractive as fingers. Furthermore, old entries in the prospective links list are removed at regular intervals to avoid using outdated node information and reduce the risk of sending finger requests to failed nodes.

5.3 Hierarchical DHT- HRM

The hierarchical approach is called HRM for Hierarchical Resource Management [RBS12, RBS13], where nodes are separated into different hierarchical layers and maintain links within and between those layers. This approach is inspired by small world networks and in particular by Milgram's well known small-world experiment in which letters were sent to unknown destinations and arrived with an average of 6 intermediate forwards, giving rise to the popular term "six degrees of separation" [Mil67]. In order to achieved such short routing distances, letters tended to first be sent to very well linked, popular individuals near the destination. Well linked individuals then forwarded letters to increasingly more specialized but poorly linked individuals until the destination was reached. Following this model, this hierarchical approach has highly linked powerful nodes and poorly linked weak nodes so that messages are quickly routed upwards through the hierarchy before they are routed back down to weaker destination nodes.

The lowest layer, consisting of the weakest nodes, functions as a leaf layer where each leaf node maintains a parent node from some upper layer. Each layer is linked to form a location aware DHash++ [DLS+04], with the number of shortcut links (i.e. fingers) determined by the given hierarchy layer. Thus, the higher a node's resource availability (i.e. the higher the layer it is placed in), the more links it is expected to maintain. This significantly reduces weaker nodes' maintenance loads despite their additional load as parent nodes. Figure 5.3 provides a first impression of how the hierarchy is built and demonstrates how nodes with varying resource availability levels can be divided into different hierarchical layers (more complex allocations are discussed below). Furthermore, as explained below in detail, Algorithm 3 shows how nodes save information that is piggybacked on network messages about the physical distances to nodes from various hierarchy layers; Algorithm 6 clarifies how links are found using this saved information; and Algorithm 7 delineates the hierarchical routing procedure.

5.3.1 Varying Levels with Varying Responsibilities

In the following, the notation "layer" is reserved strictly for hierarchy layers and "level" for resource availability levels. The number of hierarchy layers used is $h_{max} - 1$, with layers $\{0, \ldots, h_{max}\}$, and a node x's hierarchy layer is denoted x_H. In the simplest case, $h_{max} = l_{max}$ such that for every node $x_R = x_H$, as in Figure 5.3. We generally assume that the higher a node's resources are, the higher its hierarchy layer will be. In the following, the following terms are used:

bottom layer for nodes in hierarchy layer $= 0$,

upper layer for nodes with hierarchy layer > 0,

top layer for nodes with hierarchy layer $= h_{max}$, and

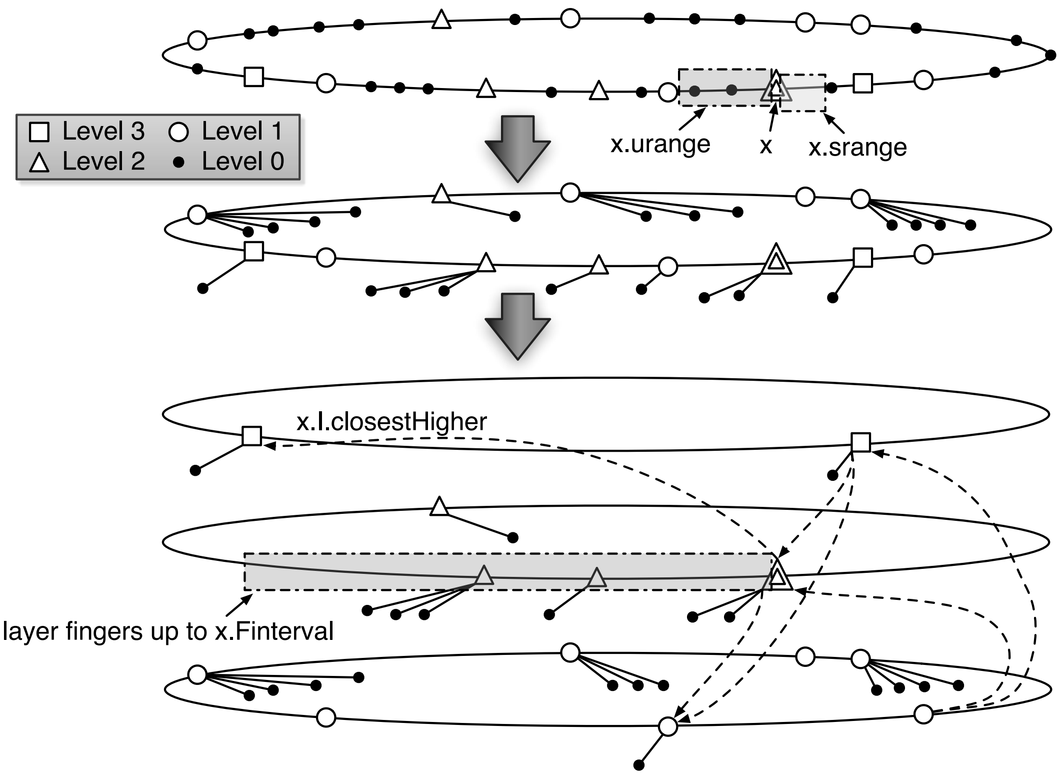


Figure 5.3: All hierarchy layers are shown together on the top key ring for $x_H = x_R$. Following the first arrow down, nodes are shown with their respective leaf nodes on one central ring. Following yet another arrow down, nodes are shown in their three upper hierarchy layer keyrings with their bottom layer leaf nodes. Each upper layer forms a location aware DHash++ overlay and links are established to the closest successors in other upper hierarchy layers.

lower layer for nodes with hierarchy layer $< h_{max}$.

In addition to each node's links to its predecessor and successors, there are three additional types of links:

leaf-parent links between bottom layer nodes and the closest preceding upper layer node,

inter-layer links connecting each upper layer node with its immediate successor in *each* of the $h_{max} - 1$ other upper layers, and

layer fingers providing each upper layer node with shortcuts within its own hierarchy layer.

Since nodes establish links and perform routing differently based on their *own* layers (and thus, indirectly their resource levels), this approach is clearly hierarchical. Furthermore, each node maintains links to all other layers, with the exception of leaf nodes which only have access to one particular other (parent) layer, so this approach can be classified as horizontal.

5.3.2 Finding Links

In order to find nodes within the hierarchical structure, two new key ranges are defined. The *simple key range* is used to assign nodes to keys as we are familiar with from RBFM and the

ProspectiveLinks[x_H][i]

NodeID	P.Dist.
p_2ID	0.4
p_3ID	0.9
p_4ID	0.9
p_1ID	1.7

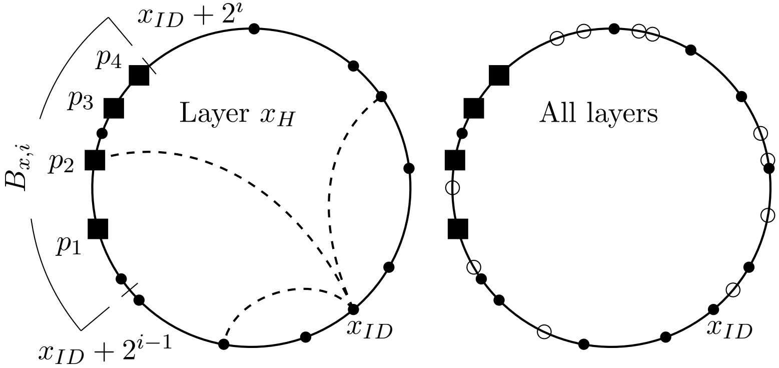


Figure 5.4: Key ring at right shown for node x with all nodes not in layer x_H as hollow circles and at left with layer x_H nodes only: six nodes in $B_{x,i}$, four of which x knows in its x_H prospective links list (squares). A layer finger is established to p_2, the known node with the best physical distance to x in layer x_H.

underlying Chord. The *upper key range*, on the other hand, is used by upper layer nodes for identifying the key range in which they must act as parents to leaf nodes.

Definition 5.3.1 (Simple Key Range)**.** *A node x's simple key range $x.srange$ spans the keys between its predecessor y's key (exclusive) and its own key:*

$$x.srange = (y_{ID}, x_{ID}].$$

Thus, since each key κ is assigned to the first node whose nodeID is equal to or succeeds κ on the key ring, it is stored on that node whose simple key range contains κ. For the upper key range, which is integral to routing success, each node maintains an upper layer successor and predecessor node, i.e. the first successor and predecessor nodes from *any* upper layer.

Definition 5.3.2 (Upper Key Range)**.** *An upper node x's upper key range $x.urange$ consists of all keys between x_{ID} and its upper layer successor y's nodeID y_{ID}:*

$$x.urange = [x_{ID}, y_{ID}).$$

Note that *x.srange* and *x.urange* overlap only in x_{ID}. Node x also saves information about its upper layer successor y's simple key range. This enables x to route messages destined for y directly instead of forwarding through leaf nodes.

Leaf Links

Each bottom layer node x ($x_H = 0$) maintains a link to its *parent node*, which is the first upper layer node preceding x in the keyspace. Thus, leaf nodes have parents from varying hierarchy layers and each upper layer node has the same expected number of leaf nodes to maintain. Leaf nodes have neither the inter-layer links nor layer fingers described below, so this parent link is necessary for efficient, successful routing.

Inter-layer Links

Each upper layer node x establishes a link $x.I[\ell]$ to its direct successor $x.I[\ell].node$ in each of the $h_{max} - 1$ upper layers $\ell \neq x_H$. This is made possible by the upper layer successor and predecessor links that each upper layer node maintains. These links form an additional ring on which requests can be routed with the help of the layer finger shortcuts (see Section 5.3.3).

Inter-layer links enable routing between any two hierarchy layers and are used to restrict the number of layer fingers (see below). Generally, the range of layer fingers ($0 < x_H < h_{max}$) can be restricted to the key range "gap" between two higher layer nodes. This ensures that the network is well connected throughout the more densely populated lower layers but longer key-distances are routed over higher layers. Thus, for layer $0 < x_H < h_{max}$ nodes, we are interested in the finger interval in which node x's nearest key-distance inter-layer link from any higher layer is located. We define $x.I.closestHigher$ as the closest of x's higher layer inter-layer links, $x.I.closestLevel$ as its hierarchy layer, and $x.I.closestInt$ as the finger interval in which it is found:

$$\begin{aligned} x.I.closestLevel &:= \operatorname*{argmax}_{\ell: x_H < \ell \leq h_{max}} d_{key}(x, x.I[\ell].node) \\ x.I.closestHigher &:= x.I[x.I.closestLevel].node \\ x.I.closestInt &:= j : x.I.closestHigher \in B_{x,j}. \end{aligned}$$

Note that the finger interval is as defined for the flat approach. The interval $x.I.closestInt$ is then used by layer 1 nodes (and possibly other lower layers, depending on the configuration) as a bound for the farthest layer finger established, causing short key space hops to be performed within a single layer and longer hops to be forwarded to higher layers. This inevitably causes lower maintenance and routing load within the lower layers.

Layer Fingers

In HRM, a node x only chooses *layer fingers* within its own hierarchy layer x_H. Furthermore, the number of fingers that nodes establish varies from layer to layer. A weak node x with $x_H = 1$ has as few fingers as necessary, establishing layer fingers only to nodes which are closer successors in the keyspace than $x.I.closestHigher$, i.e.:

$$d_{key}(x, x.F[i].node) < d_{key}(x, x.I.closestHigher). \tag{5.2}$$

Meanwhile, a strong node x with $x_H = h_{max}$ maintains fingers for each finger interval $B_{x,i}$ for $i \in \{1, 2, \ldots, m\}$. Nodes in additional layers ℓ with $1 < \ell < h_{max}$ maintain sets of fingers of varying sizes, depending on ℓ, which is determined as part of the system design.

Definition 5.3.3 (Finger Range)**.** *The furthest finger interval in which a node x in hierarchy layer x_H maintains a finger is called its* finger range*:*

$$x.Frange = Frange(x_R) \in \{1, 2, \ldots, m\}$$

and $x.Fkey = x_{ID} + 2^{x.Frange-1}$ its corresponding key value. For $x_R \in \{1, h_{max}$ the finger range is defined as:

$$x.Frange = \begin{cases} x.I.closestInt, & x_H = 1 \\ m, & x_H = h_{max}. \end{cases}$$

Thus, each upper layer node x maintains a *finger table* with one finger in layer x_H for each finger interval $B_{x,i}$ with $i \in \{1, 2, \ldots, x.Frange\}$. Note that the fewer links a layer maintains, the less maintenance load is incurred and the faster messages are passed on to other (higher) layers. Lookups are thus routed quickly out of the (weak) bottom layers and dispersed between the (strong) upper layers.

For example, a system with five hierarchy layers ($h_{max} = 4$) might use finger ranges:

$$x.Frange = \begin{cases} x.I.closestInt, & x_H = 1 \\ m - 1, & x_H = 2 \\ m, & x_H \in \{3, 4\}, \end{cases}$$

where the top two layers establish layer fingers for the entire keyspace, the middle layer nodes establish one less layer finger (thus ignoring half of the key space), and the layer one nodes only provide layer fingers only up to their first higher layer successor.

Algorithm 5 Updating prospective links lists with $\leq k$ entries.

```
procedure suggestProspectiveLink(nodeInfo)
    finger = getFingerInterval(nodeInfo.key)
    dist = getPhysicalDist(nodeInfo.coordinates)
    layer = nodeInfo.hierarchyLayer
    if prospectiveLinkList(layer).contains(finger, nodeInfo.key) then
        prospectiveLinkList(layer).update(finger, dist, nodeInfo)
    else if prospectiveLinkList(layer).size(finger) < k or
            dist < prospectiveLinkList(layer).farthestLinkDist(finger) then
        prospectiveLinkList(layer).addNode(finger, dist, nodeInfo)
        while prospectiveLinkList(layer).size(finger) > k do
            prospectiveLinkList(layer).removeFarthestLink(finger)
        end while
    end if
end procedure
```

Algorithm 6 Maintaining layer fingers 1 to $m - 1$

```
procedure maintainFinger(finger)
    lookupKey = myKey + getOffset(finger)
    myLayerList = prospectiveLinkList(myLayer)
    if myLayerList.size(finger) > 0 then
        listEntry = myLayerList.getClosestEntry(finger)
        lookupKey = listEntry.key
        myLayerList.removeUsedEntry(listEntry)
    end if
    sendLookup(lookupKey)
end procedure
```

Layer fingers are chosen in a location aware fashion as in DHash++. Nodes' coordinates, resource levels, and hierarchy layers are piggybacked on network messages, providing node information to other nodes at minimal overhead. Thus, an upper layer node x chooses for

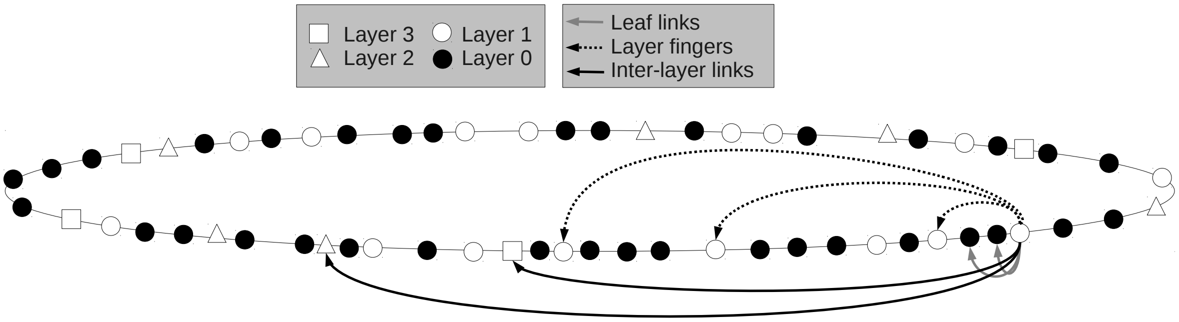


Figure 5.5: All hierarchy layers are shown together on one key ring with all links shown for a single node. Note that this layer one node's finger range is determined by its layer three closest higher inter-layer link - no fingers are maintained after this link.

$x.F[i]$ the known node in hierarchy layer x_H and finger interval $B_{x,i}$ which has the smallest physical distance to x. For this, x maintains a set of h_{max} *prospective links* lists, one for each upper hierarchy layer. The ℓ^{th} prospective links list contains the k physically closest nodes in $B_{x,i}$ for each $i \in \{1, 2, \ldots, m\}$ from hierarchy layer ℓ which are known to x. Thus, at most $k \cdot m \cdot h_{max}$ nodes are saved via their hierarchy layers, nodeIDs, and physical distances.

When x receives a message that originated at sender y, x uses y's coordinates to determine $d_{phy}(x, y)$ and update its prospective links list accordingly (see Algorithm 5). An i^{th}-finger request is sent to the physically closest entry in x's layer x_H prospective links list for $B_{x,i}$, if it contains an entry. Otherwise, the first successor of key $x_{ID} + 2^{i-1}$ in layer x_H is contacted (see Algorithm 6), which requires layer-specific lookup forwarding (see Routing). Upon node x's receipt of a finger request response from node y, if $y_H = x_H$ and $y \in B_{x,i}$ with $i \leq x.Frange$, then y is assigned to $x.F[i]$. This may not be the case if y's layer has changed or y was found without a prospective links list entry and is beyond the finger range. If a given finger interval contains no layer x_H node, then this finger entry remains empty.

Note that while node x only links layer fingers to nodes in layer x_H, maintaining prospective links lists for multiple layers causes little overhead while easing a node's transition between hierarchy layers (when, for example, a node's resource level changes). However, should this cause too much added computational load for weak nodes, the number of layers and/or finger intervals for which these lists are maintained can be adjusted. The prospective links list entries are continually updated with fresh node information to automatically adapt to changing coordinates and are deleted once used for a finger request to ensure their freshness. It is especially important for the lists from other layers to be periodically cleared of old entries, since they are not cleared with finger maintenance. Simulations have shown that $k = 1$ is beneficial in networks with high churn, reducing the use of failed prospective links and minimizing the lists' overhead.

Figure 5.3 shows the basic overlay structure, with hierarchy layers assigned as $x_H = x_R$. The connected key ring on which each node establishes its predecessor and successor is shown on top, and individual layers are shown below with the bottom layer nodes assigned to their upper layer predecessors (i.e. parents). Inter-layer links are shown for three nodes only and layer fingers were omitted. The closest higher layer inter-layer link and finger range has also been indicated for the layer 2 node x marked with a double triangle. Figure 5.5 shows all of the links for a single node.

5.3.3 Routing

Routing of lookups is not performed in a strictly greedy fashion like Chord, but rather in a series of greedy steps. Recall that a message is destined for the node whose simple key range contains the message's destination key κ. Let κ be the message's destination key in the following considerations. Note that we add one negligible piece of information to messages (in high churn scenarios): the key of the last upper layer node that handled the message. Since leaf nodes actually route nodes *backwards* to their parent nodes, this prevents unwanted routing cycles when nodes have not yet adapted to network changes and have out of date parent links or leaf node lists. Once a node x has determined that $\kappa \notin x.srange$, its routing behavior depends on its hierarchy layer as follows.

Leaf Nodes

If a message originates at a leaf node, the first step is to route it to the parent node. If a leaf node x receives a message with $\kappa \notin x.srange$ and its parent node was not the last upper layer node to have handled the message, it forwards the message to its parent node. This may be the case if the parent node's connections to its leaf nodes or upper layer successor are damaged, which is not uncommon in systems with high churn rates. Otherwise, it forwards the message to its successor node y if $\kappa \in y.srange$ or else to the closest preceding node from its successor list, which is only the case when x's parent node's leaf links are not complete.

Upper Layer Nodes

An upper layer node x basically routes greedily to the closest preceding node in a layer $\geq x_H$ using both its finger table and inter-layer list. For lower layer nodes with restricted finger tables, this means that messages to 'distant' destinations are routed upward. For top layer nodes, this means that messages are routed to the closest top layer predecessor of κ. At this stage, routing is greedy with the one condition that the next hop has layer $\geq x_H$. However, once the message has reached a node x for which κ is within x's layer finger range (i.e. $d_{key}(x, \kappa) < x.Fkey$), routing becomes less greedy, as higher layer links are preferred twofold: In the top layer, inter-layer links are entirely ignored and messages are only routed to other top layer nodes until there is no closer top layer predecessor to κ. Once routing is exhausted within one of the upper layers, i.e. there is no closer predecessor node with layer $\geq x_H$, then the message is routed back down the hierarchy by choosing the highest possible inter-layer link preceding κ. If there are no such links, then $\kappa \in x.urange$ and the message is delivered directly to the node y with $\kappa \in y.srange$: y is either x's upper layer successor (whose simple key range overlaps $x.urange$) or one of x's leaf nodes.

Thus, lookup routing traverses the layers only one time and lower layer nodes are used a little as possible. Once a message has been passed upwards, it does not come back down the hierarchy until it has no closer predecessor in that layer. Algorithm 7 provides an overview of the decisions that nodes make while routing.

Algorithm 7 Lookup routing procedure at local node thisNode.

```
procedure ROUTETO(msg, key)
    if key.inSRange(thisNode) then deliverToApplication(msg)
    else if key.inSRange(successorNode) then forward(msg, successorNode)
    else if thisNode.layer = 0 then
        if message.lastHop = parentNode then
            forward(msg, findClosestPredecessor(key, successorList))
            → finds the closest predecessor node of key in successor list of r nodes
        else  forward(msg, parentNode)
        end if
    else if key.inSRange(upperLayerSuccessor) then
        forward(msg, upperLayerSuccessor)
    else
        if isBeyondLayerRouting(key) then
            → key beyond range in which thisNode maintains layer fingers
            forward(msg, findClosestPredecessor(key, interLayerList), thisNode.layer)
            → find closest inter-layer link in higher layer than thisNode
        else if hasPredecessor(key, layerFingerList) then
            forward(msg, findClosestPredecessor(key, layerFingerList))
            → find closest layer finger
        else forward(msg, findHighestPredecessor(key, interLayerList)
            → find predecessor of key in highest possible layer
        end if
    end if
end procedure
```

Layer Routing

Finger requests and inter-layer link requests use layer sensitive overlay routing. These requests are not necessarily delivered to the node x for which $\kappa \in x.srange$, but rather to the first found successor of κ in a given layer ℓ. A node y considers itself the request's destination if $y_H = \ell$ and κ is between the sending node's nodeID and y_{ID}. If not, y forwards to its layer ℓ inter-layer link if it succeeds κ, and otherwise to its closest known preceding link in layer ℓ. This routing is not necessarily correct, especially in dynamic networks, but it is sufficient for join and maintenance messages and was sufficiently reliable in simulations.

Routing Characteristics

The expected lookup routing complexity in the hierarchical overlay keeps with Chord at $O(\log(N))$. Note, though, that the simulation results did show an increase in routing hops compared with Chord (1-2 hops), as expected due to the hops up and down the hierarchy (see below). The following finger ranges are used for the following routing theorem:

$$x.Frange = \begin{cases} x.I.closestInt, & 0 < x_H < h_{max} \\ m, & x_H = h_{max}. \end{cases}$$

Since the number of layer fingers is higher for alternative finger ranges, the routing can only be better.

Theorem 5.3.1 (Lookup Hop Length Theorem)**.** *Given a network with N nodes, the expected upper bound for the number of overlay hops required to route a lookup from any node to the successor node of any key κ is $O(\log(N))$ hops.*

Proof. We assume that a message is to be routed from any node to the node y responsible for key κ, i.e. $\kappa \in y.srange$. To show that routing terminates and find an upper bound on the routing hops, we consider the farthest possible lookup route. Since it takes at most one hop to route from a leaf to a parent node, we assume that each message originates at an upper layer node **(1 hop)**. Furthermore, it takes at most one hop to reach y from κ's upper layer predecessor, since y's upper layer predecessor (which is its parent node if $y_H = 0$) always knows y's simple key range. Thus, we need only determine the number of hops necessary to reach κ's (and y's) *upper* layer predecessor **(1 hop)**.

If the originating node, its successor, or its upper layer successor are the destination, then we are done. Otherwise, the message is passed upwards until it reaches the first layer in which κ is a predecessor of the current node x's farthest finger interval's corresponding key $x.Fkey$. In other words, κ is in x's finger range or $d_{key}(x, \kappa) < d_{key}(x, x.Fkey)$. To reach this layer, the message is passed up at most $h_{max} - 1$ layers, from layer 1 to layer h_{max} **($h_{max} - 1$ hops)**.

So assume that κ is a predecessor of $x.Fkey$ and will thus be routed within layer $\lambda := x_H$ on layer fingers until it has reached either the destination node or the closest predecessor node within layer x_H. Then, as shown for RBFM in Section 5.2.2 and as for DHash++, the routing complexity within layer x_H is at most $O(\log(x.N))$, where $x.N$ is the number of nodes in layer x_H between x and $x.Fkey$. So assume that the message is at the closest predecessor node z of κ in this layer **($O(\log(x.N))$ hops)**.

Assuming the message has not reached its destination, the message is passed at most once to each of the other upper layers and routed analogously (starting at higher layers first), because it is only passed to a new layer ℓ' once there is no closer predecessor in the current layer ℓ and the message can therefore never be passed back to ℓ. So we have a total of $h_{max} - 1$ remaining hops between other upper layers **($h_{max} - 1$ hops)**.

However, since we know that z is the closest predecessor node to κ in layer λ, we know that the remaining key distance between κ and y is less than the key distance between z and z's layer λ successor. Thus, we can use the expected number of network nodes *between* any two layer λ nodes (some constant c) as an upper bound on the key range and thus the number of nodes over which remain to route in each of the remaining layers. Thus, we expect at most $O(\log(c))$ routing hops in each remaining layer $\ell > 0, \ell \neq \lambda$ **($O(\log(c)) \cdot (h_{max} - 2)$ hops)**. Note that this is a loose bound, as the remaining key range and thus the number of nodes over which to route decreases as the message is forwarded through the layers. This gives us an expected total of at most:

$$\begin{aligned} 2 + 2(l_{max} - 1) + O(\log(x.N)) + (l_{max} - 2)O(\log(c)) &\leq 2 \cdot l_{max} + \hat{c} + O(\log(N)) \\ &= O(\log(N)) \end{aligned}$$

for some constant $\hat{c}$. □

In fact, simulation results did show an increase in routing hops for an increase in the number of hierarchy layers and compared with Chord, as expected due to the maximum $2 \cdot h_{max}$ hops up and down the hierarchy.

5.3.4 Link Maintenance

Given the dynamics of mobile networks and the various roles of links, maintenance is integral for detecting and addressing network changes and ensuring correct links and data availability. Since maintenance composes a large portion of the network load, it plays a major role in node lifetimes when resources are restricted. Nodes perform maintenance stabilization for their links at given intervals s_i as shown in Table 5.1 along with the number of links that are maintained by bottom (i.e. leaf) and upper nodes per link type. These intervals can vary, depending on a network's churn, but several are performed with the same frequency.

Link	Delay	Bottom nodes	Upper nodes
Predecessor	s_0	1	1
Successor	s_1	r	r
Parent node	s_1	1	-
Upper layer predecessor	s_1	-	1
Upper layer successor	s_1	-	1
Leaf node	s_1	-	many
Inter-layer link	$g(x_H)$	-	$h_{max} - 1$
Layer finger	$g(x_H)$	-	$\leq x.Frange \leq m$

Table 5.1: Varying forms of link maintenance with the delay between messages and the number of links that bottom and upper nodes must maintain. For inter-layer links, x refers to the individual link nodes (and x_H to their hierarchy layers) and thus varies, while for layer fingers x_H is constant.

Inter-layer links and layer fingers are maintained as in RBFM with varying maintenance intervals and are automatically adapted when nodes change hierarchy layers. Thus, each link is maintained using its finger maintenance interval that depends on the link node x's hierarchy layer as opposed to its resource level, giving us $g(x_H)$. Using the examples of finger maintenance interval functions from Section 5.2.3, bottom layer links are maintained according to a reference interval t_{ref} and higher layer links at varying multiples of t_{ref} for each hierarchy layer. Generally speaking, links to lower layers are maintained more frequently than links to higher layers. However, leaf and parent links as well as upper layer successor and predecessor links are maintained analogously to direct successor links, using direct maintenance messages at a fixed interval s_1. This is due to their importance in routing success; outdated leaf or parent links can be very costly, since messages must then be passed on direct successor links, while the upper layer successor and predecessor links are necessary for the upper layer ring and all upper layer activities.

5.3.5 Node Joins and Failures

To join the DHT, a node x must have valid network coordinates, choose a nodeID and resource level, map its resource level to a hierarchy layer, and contact one participating node. Once x has established links to its immediate predecessor p and successor s on the key ring, s sends its successor lists to x, which x uses to initialize its own lists, and corresponding keys are transfered from s to x. Once x has completed the basic join in the overlay, it must also perform either a leaf join or upper layer join (see below). The node x continually updates its prospective links lists and periodically performs finger maintenance (see Algorithms 3 and 6) to establish and maintain its fingers.

The basic reaction to node failures is as in Chord, with failed nodes also removed from the inter-layer list, prospective link lists, and potentially the parent link or leaf list once their failure is noticed. An upper layer node leaves gracefully by sending messages to each of its leaf nodes and its upper layer predecessor informing them of their new parents/leaf nodes. Otherwise, if a leaf node's parent has failed unexpectedly, the leaf node must perform a leaf join to reestablish a parent (see below). If a node is removed from the inter-layer list or parent link, then its information must be stored for a short period to deal with other nodes' old hints to the failed node.

Upper Layer Joins and Failures

The upper layer join serves two purposes: establishing the upper layer successor and predecessor nodes (from any upper layer) and transferring responsibility for leaf nodes. Node x uses an upper layer bootstrap node to send an upper layer join message, which is routed along the upper layer nodes to x's upper layer predecessor y, for which $x_{ID} \in y.urange$. In other words, x belongs to the key range in which y maintains leaf nodes. Node y responds to x with its own upper layer successor z and the list of y's leaf nodes which are now in $x.urange$. Then y informs each of these leaf nodes of their new parent node x and removes them from its leaf list.

If an upper layer node's resource level is reduced such that it becomes a leaf node, it forfeits its role as parent node by transferring its leaf nodes to its upper layer predecessor. As a leaf node, it then ignores finger and inter-layer link requests, upper and leaf join requests, and upper stabilize requests. Its leaf nodes are transfered to its upper layer predecessor y by sending a message to y with a list of its leaf nodes as well as messages to each of its leaf nodes with a hint to y. Leaf nodes do not respond to many of the requests that upper layer nodes respond to (finger and inter-layer link requests, upper and leaf join requests, upper stabilize requests), yet they still perform data lookups. If it is observed that a node has left the upper layers, either gracefully or unannounced, that node must be removed from inter-layer lists, prospective link lists, upper layer successor and predecessor links, and parent links (but not from successor and predecessor links).

Leaf Joins and Failures

Node x with $x_H = 0$ performs a leaf join to establish a live parent node. Recall that a node's parent is the first preceding upper layer node on the key ring. The leaf join message is forwarded to an upper layer bootstrap node and then routed to the upper layer node whose

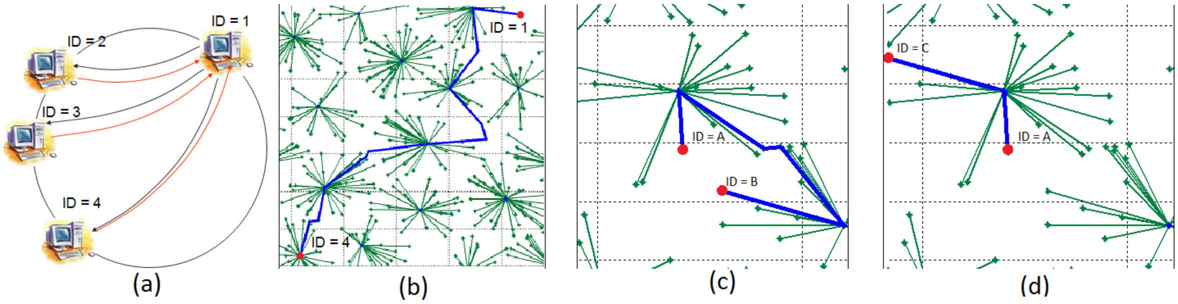


Figure 5.6: A single overlay hop as shown in (a) is performed on the underlay shown in (b) which routes messages along cluster heads. Physically close nodes may have longer connection routes than more distant pairs depending on the clusters they belong to, as demonstrated in (c) and (d).

upper layer key range contains x_{ID}. This parent node responds and enters x into its leaf list. Leaf node maintenance is performed frequently by the parent node so that failures are quickly recognized. Leaf node failure affects only the parent node and the direct successor and predecessor on the key ring.

5.3.6 Adaptability

The hierarchical approach poses more challenges regarding adaptability than a flat approach, since changes in resource levels that result in changes in hierarchy layers require nodes to change both their incoming and outgoing links. The prospective links lists aid this transition on both sides and in both directions, helping a node with a new hierarchy layer by providing lists of potentially nearby nodes within that layer, and helping nodes whose links have changed layers to find new links. Nodes can thus move relatively freely between the hierarchical layers as their resources change. Location movements, on the other hand, are treated much as in RBFM with nodes adapting their layer fingers automatically via their prospective links list.

5.4 Adaptations

The following two approaches present adaptations to the flat and hierarchical DHTs, respectively. They were primarily developed for evaluation purposes (see Chapter 7): the cluster-based RBFM for evaluating the effects of varying approaches on underlay behavior, and the hybrid DHT for comparing both RBFM and HRM to a resource aware traditional two-tiered hierarchy.

5.4.1 Cluster-based Flat DHT

Cluster-based versions of DHash++ and RBFM were developed for scenarios in which nodes use a form of cluster-based underlay routing, such as that illustrated in Figure 5.6. The resulting approaches were developed in joint work with Zafar et al. [ZARBH13] and are referred to as C-DHash++ (cluster aware DHash++) and C-RBFM (cluster aware RBFM). In the case that a node routes messages on the underlay first to a physically close cluster-head (or other node within its own cluster), which then performs the longer-distance forwarding, overlay links to physically close nodes in other clusters are much more costly with regard to energy, bandwidth, and time than links within a node's own cluster.

Thus, a node's finger selection is altered to depend not only on the physical/resource distance to prospective peers but also on the clusters to which they belong. The node information piggybacked on messages is extended to include the sending node's clusterID and peers prospective links lists are extended by an extra binary value "SameCluster." Nodes with identical clusterIDs are preferred to nodes with low physical/resource distance when entering nodes into the list. When choosing fingers, links within the same cluster are also given priority. Should there be multiple same-cluster nodes within one finger interval, the node with the shortest physical/resource distance is chosen as a finger. But should there be no same-cluster node for a finger interval, then the finger is chosen just as in the original DHash++ or RBFM. Examples of the prospective links lists and finger choices are shown in Figure 5.7 for C-DHash++ and in Figure 5.8 for C-RBFM.

5.4.2 Hybrid Hierarchical DHT

A hybrid variation of HRM with RBFM provides a two-tier overlay with multi-level resource awareness as well as location awareness. The hybrid approach uses only two hierarchical layers, with leaf nodes acting as HRM leaf nodes and the layer 1 superpeers building their upper layer shortcuts just as RBFM. Without the additional layers and inter-layer links, the result is a common superpeer-child structure, but the superpeers use additional resource awareness by selecting fingers to strong, nearby nodes. This approach has the benefit of additional location awareness in the upper layers compared with HRM. Since HRM has potentially sparse upper layers, upper layer nodes have fewer peers from which to choose physically close fingers, restricting location awareness. The hybrid approach, however, has fewer layers in which nodes are distributed and thus potentially more layer 1 nodes and in each finger interval, increasing the number of nearby peers from which fingers are chosen.

Figure 5.9 demonstrates the structure of the hybrid approach using four resource levels. The bottom resource level nodes are assigned to layer 0 and thus act as leaf nodes, while all other resource levels are placed in layer 1. Note that, analogous to HRM, nodes can be assigned to hierarchy layers in an arbitrary fashion. Thus, levels 0 and 2 could, for example, both be assigned to the bottom layer with the remaining levels in the upper layer. The figure shows leaf nodes' links to their upper layer predecessors and the fingers of one upper layer node. Note that fingers are established to strong, physically nearby nodes within each finger interval (finger intervals are indicated by gray backdrops).

5.5 Summary

Two primary resource and location aware DHTs, RBFM and HRM, which also fulfill the remaining derived scenario requirements, were introduced in this chapter. These overlays adapt a Chord DHT with inspiration from DHash++ for the flat RBFM and inspiration from small-world networks for the hierarchical HRM. In order to ensure the necessary scalability, proximity identifier selection was avoided and location awareness was built into nodes' choices of links. Of course, this restricts the degree of location awareness which can be achieved, but

Prospective Links [m-1] DHash++

NodeID	Phy.Dist.
p_2ID	1.6
p_3ID	1.4
p_4ID	0.9
p_1ID	4.1

Prospective Links [m-1] C-DHash++

NodeID	Phy.Dist.	SameClust.
p_2ID	1.6	No
p_3ID	1.4	Yes
p_4ID	0.9	No
p_1ID	4.1	No

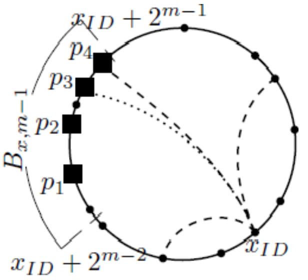


Figure 5.7: The difference between DHash++ and C-DHash++ links is illustrated on the finger interval $B_{x,m-1}$. The preferred DHash++ link is shown dashed and its cluster aware alternative for C-DHash++ dotted.

Prospective Links [m-1]: C-RBFM

NodeID	Phy.Dist.	Res.Level	Res.Dist.	SameClust.
p_2	1.6	2	2.6	No
p_3	1.4	1	3.4	Yes
p_4	0.9	0	3.9	No
p_1	4.1	3	4.1	No

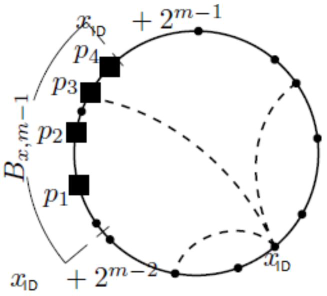


Figure 5.8: The preferred finger for finger interval $B_{x,m-1}$ in C-RBFM is not p_2 as in RBFM but rather the node p_3 within its own cluster.

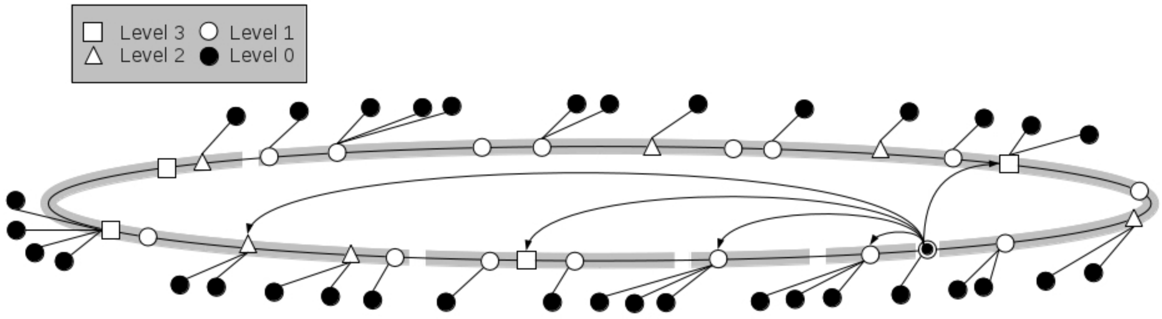


Figure 5.9: Hybrid overlay structure. Resource level 0 nodes have been placed in hierarchy layer 0 as leaf nodes, all other resource levels are placed in hierarchy layer 1. The fingers of a single layer 1 (and level 1) node are shown. The respective finger intervals from which fingers are chosen according to RBFM are shown with the gray backdrops.

location aware data placement strategies can be integrated with the help of replication, as suggested in Chapter 9.

For RBFM, a resource distance was defined which calculates the physical distance between two nodes plus an additional “resource height” with which nodes distance themselves from the entire network. This way, nodes establish links much like in Chord or DHash++ to both nearby and strong nodes, with the degree of location vs. resource awareness determined by a variable stretch constant. In order to use multiple hierarchy layers, HRM must take a different approach. A node’s resource level in HRM thus determines its hierarchy layer while location awareness is achieved only within the hierarchy layers analog to DHash++. With bottom layer nodes acting as mere leaf nodes but still responsible for storing data, HRM first routes lookups upwards in the hierarchy to strong, highly linked nodes for as much of the lookup path as possible. Both links and routing are based on nodes’ hierarchy layers and thus their resource levels. In addition to these varying overlay structures and routing, variable maintenance techniques were also introduced to reduce the largely superfluous maintenance to strong nodes and thus the overall network maintenance load.

Two adaptations of these primary DHTs, the cluster-based flat DHT and the hybrid hierarchical DHT, were introduced for the evaluation of the approaches’ performance on ad hoc multi-hop underlays and the effects of the number of hierarchy layers, respectively. The cluster-based DHTs C-DHash++ and C-RBFM add another dimension of location awareness by using information about nodes’ cluster-heads to prefer links within a node’s local cluster. This should reduce both the lookup underlay hop count and the physical distance traveled on underlays that use cluster-heads for routing. The hybrid hierarchical DHT, on the other hand, uses a two-tiered RBFM with single leaf and parent layers. Leaf nodes function as in HRM while the parent level nodes, which may come from multiple resource levels, form a resource and location aware RBFM overlay. Thus, although the number of hierarchy layers is reduced, a finer differentiation between nodes’ resource levels is still possible.

The degree of resource and location awareness of these approaches is assessed in the following chapters via mathematical analysis and simulative evaluation. The success of their awareness is gaged against their fulfillment of the remaining derived requirements, in particular the general data availability requirement. Resource and location awareness can only be considered if they uphold the systems’ data availability, but should in fact improve this availability.

Chapter 6

DHT Analysis

In order to compare these two approaches, various measures are examined through mathematical analysis. The most popular measures for DHT evaluation were discussed in Chapter 4, and are reflected in the following analysis and the evaluation that follows in Chapter 7. In contrast with previous work, however, and in order to facilitate an understanding of how nodes are treated with regards to their resources, many of these measures utilize the resource levels. For example, the maintenance or lookup load of nodes can be analyzed and evaluated *per resource level* as opposed to for an average node. Thus, instead of general system wide measures, resource-level-centric measures are used to evaluate and compare the effect of various design issues on individual resource levels.

The specific evaluation measures used in this and the following chapter are summarized in Table 6.1, with the primary measure used in Chapter 10 for the replication analysis marked "r." This table includes the categories for evaluation measures from Table 4.7 and adds two new measures that were important for the given scenario: node lifetime and the number of *forwarded* maintenance messages. The number of forwarded maintenance messages indicates how much network load is incurred by maintenance being routed via key based routing as opposed to using directly addressed messages. With the exception of maintenance latency, lookup overlap, and link load, each of the popular measures is used for at least one medium. The three unused measures are of marginal interest for our scenario since the delays incurred by maintenance are of little importance to actual robustness; the system can perform similarly without nodes taking overlapping routes to their destinations (and thus dispersing load); and the link load is in fact expected to vary greatly, with links to and from high level nodes experiencing heavier load. However, the maintenance latency *is* reflected in the expected link latencies used in this analysis, since it is these links that are being maintained.

Of the remaining measures, those that are simple enough and do not rely on paths with node dependencies are addressed using mathematical analysis, while their more complex counterparts are evaluated in simulation only. Lookup latency, for example, is very complex for analysis, since each hop depends on the hop node that preceded it and the fingers it happens to have (non-deterministically) established, but can be easily measured in simulation. The actual expected number of hops for both RBFM and HRM is $O(\log N)$, as already proven in Chapter 5 Many of the analytical measure categories have also been used in both analysis and simulation for comparison. However, the explicit mediums used may vary, for example with link resources and latency: while the analysis focuses on the resource level and distance of

		medium	resource-centric	analysis	evaluation
Maintenance	latency				
	load	bandwidth			
		messages	✓	✓	✓
		forwarded msg	✓		✓
Lookup	hop count			✓	✓
	latency	time/distance			✓
	stretch	RDP			
		underlay			✓
		hop			
	failure	overlay	✓		✓
		underlay			
	overlap				
Node	degree		✓	✓	
	keyspace		✓	r	
	load	bandwidth			
		bytes			
		messages	✓		✓
		underlay msg			✓
	lifetime		✓		✓
Link	resources		✓	✓	✓
	latency		✓	✓	✓
	load				

Table 6.1: Measures used in this work to perform mathematical analysis and simulative evaluation. Measures which are used on a resource level basis to compare individual resource levels are marked resource-centric. The keyspace portion measure marked "r" is used in the replication analysis.

links, the evaluation measures the average resource level and distance of used hop links (i.e. more popular links are weighted heavier than links with light traffic). The preferred medium for measuring load is the number of messages sent or received, since the size of lookup requests and maintenance messages is comparable, but the bandwidth or bytes from transferring of data may vary heavily depending on the application.

Note that nodeIDs are assumed to be uniformly distributed for the analytical observations. In reality, nodeIDs are often a deterministic SHA-1 hash of nodes' IP addresses, but unless the SHA-1 function is tampered with (which is considered hard to do) it distributes nodes nearly uniformly in the key space.

6.1 Flat RBFM

In RBFM, nodes choose links in a non-deterministic fashion based on both other peers' resource levels and physical distances (i.e. their resource distances), so the key to analyzing the system's behavior is first understanding how these links are distributed. Nodes' fingers are compared with the resource naive, location naive Chord and the resource naive, location aware DHash++. Using the expected resource level and physical distance of links, a failure probability for these links can also be derived under the assumption that weak nodes have higher failure rates. Maintenance load is also compared for varying finger maintenance intervals.

6.1.1 Expected Resource Level and Distance of Fingers

The probability distributions of the physical distance and resource level of a random node x's i$^{\text{th}}$ finger indirectly reflects the physical distance and used resource levels of lookups and maintenance. Since $x.f[i]$ is taken from a finger interval of size 2^{i-1}, the larger i is, the more nodes x has to choose $x.F[i].node$ from. Since x chooses the node to which it has the smallest resource distance, this node tends to be physically closer with higher resource availability as i increases. In fact, the analysis confirms that long key distance fingers tend to be physically close nodes with high resource levels (see Figures 6.1 and 6.2).

We derive these probability distributions for a node x's i$^{\text{th}}$ finger using the Zipf probability distribution $P(x_R = \ell) = p_\ell$ for the resource levels from Equation (2.1):

$$p_\ell := P(x_R = \ell) = \frac{1}{(\ell+1)^\rho} \cdot \frac{1}{\sum_{j=0}^{l_{max}} 1/(j+1)^\rho}$$

and some distribution of nodes with respect to x in the coordinate space. This distribution is expressed as the probability that a random node will have a given physical distance to x and given by the probability density function (pdf) over all nodes' physical distances, $f_D(t)$, and its cumulative distribution function, $F_D(t)$. The following theorem gives us the probability distributions for the physical distance to and resource level of a finger chosen by x from a given number k of random nodes.

Theorem 6.1.1. *Given is a node x and a set S of k random nodes with resource levels in $\{0, 1, \ldots, l_{max}\}$. Let $y \in S$ be a node with the minimum resource distance to x in S, $f_D(t)$ be the probability density function (pdf) over all nodes' physical distances, and $F_D(t)$ its cumulative distribution function. Then the probability that y has resource level $\ell \in \{0, 1, \ldots, l_{max}\}$ is*

$$P(R_{k,min} = \ell) = kp_\ell \int_0^\infty \Big(1 - \sum_{j=0}^{l_{max}} F_D(t + h(\ell) - h(j))p_j\Big)^{k-1} f_D(t)dt, \tag{6.1}$$

and the pdf for y's physical distance $t \geq 0$ is

$$f_{D_{min}}(t) = kf_D(t) \sum_{\ell=0}^{l_{max}} p_\ell \Big(1 - \sum_{j=0}^{l_{max}} p_j F_D(t - h(j) + h(\ell))\Big)^{k-1}. \tag{6.2}$$

Proof. These equations are perhaps best understood by considering the meaning of the terms in each of these equations:

$$P(R_{k,min} = \ell) = \underbrace{k}_{a.}\,\underbrace{p_\ell}_{b.}\,\underbrace{\int_0^\infty \overbrace{\Big(1 - \sum_{j=0}^{l_{max}} F_D(t + h(\ell) - h(j))p_j\Big)^{k-1}}^{d.} f_D(t)dt}_{c.},$$

a. Number of possible nodes that may have minimum resource distance.

b. Probability that the closest node has resource level $y_R = \ell$.

c. Probability that for all nodes $v \in S/\{y\} : d_{res}(x,v) \geq d_{res}(x,y)$.

d. Probability that, with fixed physical distance and resource level to the node in question y ($d_{phy}(x,y) = t$ and $y_R = \ell$), all other nodes have a large resource distance: $v \in S/\{y\} : d_{res}(x,v) \geq d_{res}(x,y)$.

In other words, there are k nodes in the set $S = \{y_1, \ldots, y_k\}$ which may each have the smallest resource distance to x, and their probabilities to be both the best node and have resource level ℓ must be summed:

$$\begin{aligned}
P(R_{k,min} = \ell) &= P(\exists y' \in S, \forall v \in S/\{y'\} : y' = \ell \,\text{and}\, d_{res}(x,y') \leq d_{res}(x,v)) \\
&= \sum_{i=1}^{k} P(y_{kR} = \ell \,\text{and}\, d_{res}(x,y_k) \leq d_{res}(x,v),\, \forall v \in S/\{y_k\}) \\
&= \sum_{i=1}^{k} P(y_{kR} = \ell) \cdot P(d_{res}(x,y_k) \leq d_{res}(x,v),\, \forall v \in S/\{y_k\} | y_{kR} = \ell) \\
&= kp_\ell \cdot P(\exists y' \in S, \forall v \in S/\{y'\} : d_{res}(x,y') \leq d_{res}(x,v),\, y' \in S | y'_R = \ell) \quad (6.3)
\end{aligned}$$

Now the probability that a single node y' with resource level ℓ has the smallest resource distance means that for each physical distance t from x ($d_{phy}(x,y) = t$), each of the other nodes in S has a distance that, together with its resource level, results in a larger resource distance. So for y' with level ℓ and physical distance t, this means that for all other $v \in S/\{y'\}$:

$$\begin{aligned}
& d_{phy}(x,v) + h(v_R) = d_{res}(x,v) \geq d_{res}(x,y') = t + h(\ell) \\
\Rightarrow \quad & \qquad\qquad d_{phy}(x,v) \geq t + h(\ell) - h(v_R).
\end{aligned}$$

Thus, the probability that a single random node v with unknown resource level has a larger resource distance than y' for a fixed $d_{phy}(x,y) = t$ and $y_R = \ell$ is:

$$\begin{aligned}
\sum_{j=0}^{l_{max}} (1 - F_D(t + h(\ell) - h(j)))p_j &= \sum_{j=0}^{l_{max}} (p_j - F_D(t + h(\ell) - h(j))p_j) \\
&= 1 - \sum_{j=0}^{l_{max}} F_D(t + h(\ell) - h(j))p_j
\end{aligned}$$

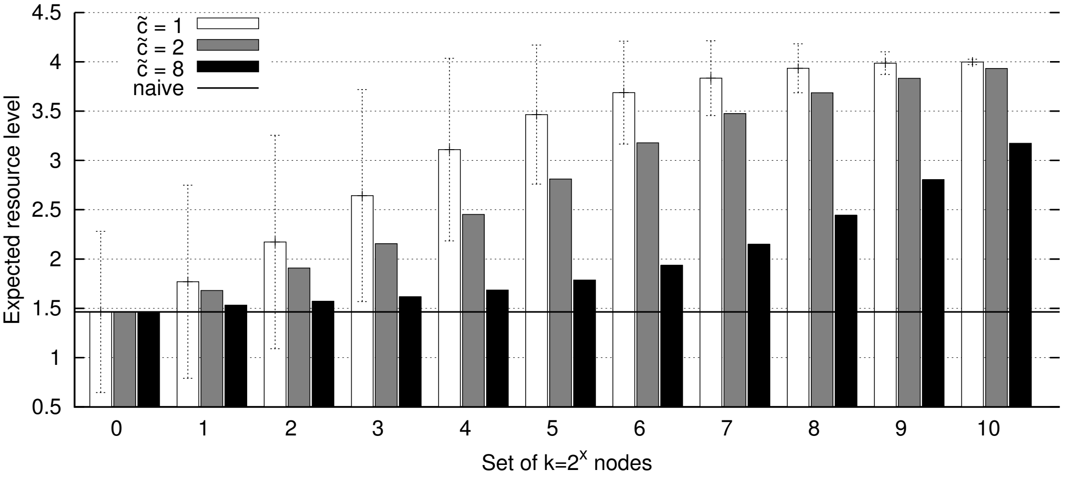


Figure 6.1: The expected resource level of the node with minimum resource distance from sets of $2^0, 2^1, \ldots, 2^{10}$ nodes are shown for $\rho = 2$ and $r = 10$, the standard deviation is shown for stretch $\tilde{c} = 1$ only.

Of course, in our case there are $k-1$ random nodes which must fulfill this condition, giving us the above term d. Integrating over all possible physical distances $d_{phy}(x,y) = t$, we obtain the above term c for 6.3:

$$\begin{aligned} &P(d_{res}(x,y') \leq d_{res}(x,v),\, y' \in S, v \in S/\{y'\} | y'_R = \ell) \\ &\qquad\qquad = \int_0^\infty \Big(1 - \sum_{j=0}^{l_{max}} F_D(t + h(\ell) - h(j))p_j\Big)^{k-1} f_D(t)dt, \end{aligned}$$

The pdf for the physical distance is derived analogously, so only a similar description of the various terms is given here:

$$f_{D_{min}}(t) = \underbrace{k}_{a.}\underbrace{f_D(t)}_{b.}\underbrace{\sum_{\ell=0}^{l_{max}} p_\ell \overbrace{\Big(1 - \sum_{j=0}^{l_{max}} p_j F_d(t - h(j) + h(\ell))\Big)^{k-1}}^{d.}}_{c.}$$

a. Number of possible nodes that may have minimum resource distance.

b. Probability density function for physical distance t of y.

c. Probability that for all nodes $v \in S/\{y\} : d_{res}(x,v) \geq d_{res}(x,y)$.

d. With fixed $d_{phy}(x,y) = t$ and $y_R = \ell$, probability that $\forall v \in S/\{y\} : d_{res}(x,v) \geq d_{res}(x,y)$

□

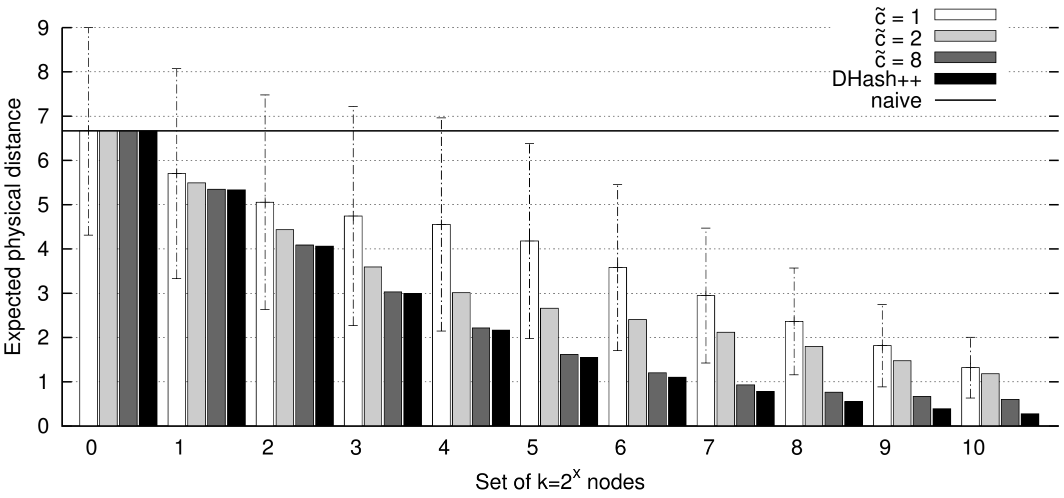


Figure 6.2: The expected physical distance of the node with minimum resource distance from sets of $2^0, 2^1, \ldots, 2^{10}$ nodes are shown for $\rho = 2$ and $r = 10$ along with the respective values for DHash++ and Chord. The corresponding standard deviation is shown for one stretch value $\tilde{c} = 1$.

While node x's i+1$^{\text{st}}$ finger interval contains 2^i keys, it only contains an expected $k = \frac{N}{2^m} \cdot 2^i$ nodes. However, the number of actual nodes which are known to x and thus used in the decision for a finger for each finger interval may be substantially smaller. Choosing the node with the minimum resource distance to x from these k nodes, (6.1) will give us the probability that x's i$^{\text{th}}$ finger has resource level $\ell \in \{0, 1, \ldots, l_{max}\}$ and (6.2) will give us the pdf of its physical distance to x.

Note that it is not possible to make any statements about the fingers' resource levels or physical distances without some assumption about the nodes' distribution within the coordinate space. For this reason, we consider one specific and simple case, where nodes are uniformly distributed around a central point x on a disk of radius r. We observe only x, which means that we are only examining the disk's center node, but similar observations were made for nodes on the edges of the disk.

$$f_D^u(t) = \begin{cases} 2t/r^2 & 0 \leq t \leq r \\ 0 & \text{else} \end{cases}, \tag{6.4}$$

$$F_D^u(t) = \begin{cases} t^2/r^2 & 0 \leq t \leq r \\ 0 & \text{else.} \end{cases} \tag{6.5}$$

To simplify (6.1) and (6.2) for this distance distribution, we use a concrete instance of the height function $h(x_R)$ from (5.1) with

$$c := \frac{r}{l_{max} \cdot \tilde{c}}$$

$$h(x_R) = \frac{l_{max} - x_R}{l_{max}} \cdot \frac{r}{\tilde{c}}.$$

This hight function determines the resource height as a fixed fraction $r/\tilde{c}$ of the network's physical radius multiplied with $(l_{max} - x_R)/l_{max}$. This means that the highest used resource height for distancing a node with resource level 0 from the rest of the network is a fraction $r/\tilde{c}$ of its longest possible physical distance to other nodes. Then using (6.1), we obtain a probability distribution which is *independent* of r

$$
\begin{aligned}
P(R_{k,min} = \ell) &= kp_\ell \int_0^\infty \Big(1 - \sum_{j=0}^{l_{max}} F_D^u(t + h(\ell) - h(j))p_j\Big)^{k-1} \frac{2t}{r^2}\mathrm{d}t \\
&= \frac{2kp_\ell}{r^2} \int_0^\infty \Big(1 - \sum_{j=0}^{l_{max}} F_D^u(t + \frac{l_{max} - \ell}{l_{max}} \cdot \frac{r}{\tilde{c}} - \frac{l_{max} - j}{l_{max}} \cdot \frac{r}{\tilde{c}})p_j\Big)^{k-1} t\mathrm{d}t \\
&= \frac{2kp_\ell}{r^2} \int_0^\infty \Big(1 - \sum_{j=0}^{l_{max}} F_D^u(t + \frac{j - \ell}{l_{max}} \cdot \frac{r}{\tilde{c}})p_j\Big)^{k-1} t\mathrm{d}t \\
&= 2kp_\ell \int_0^\infty \Big(1 - \sum_{j=0}^{l_{max}} F_D^{u'}(t + \frac{j - \ell}{c \cdot l_{max}})p_j\Big)^{k-1} t\mathrm{d}t
\end{aligned}
$$

with

$$
F_D^{u'}(t) = \begin{cases} t^2 & \text{if } 0 \leq t \leq 1 \\ 0 & \text{else.} \end{cases}
$$

The expected resource levels as given by these probabilities are depicted in Figure 6.1 for $l_{max} = 3$, specific values of k $(2^0, 2^1, \ldots, 2^{10})$, and stretch constant $\tilde{c} = 1, 2$, and 8, and the corresponding expected values for the physical distance to the node with minimum resource distance as shown in Figure 6.2. Although we do not expect that x knows all of the nodes in each $B_{x,i}$, the expected number of nodes per finger interval doubles per interval and we presume that each node knows a fair number of nodes per finger interval. Since finger interval $B_{x,i}$ contains 2^{i-1} key values, a node's $\lceil m + 1 - \log(N) \rceil^{\text{th}}$ finger interval is the first in which a node is expected to be found. Thus, each set of 2^j nodes in Figures 6.1 and 6.2 corresponds to a node's $j + \lceil m + 1 - \log(N) \rceil^{\text{th}}$ finger interval in a network of N nodes. The expected bound for nodes' first fingers is found by solving for j after setting the key size (2^{j-1}) of a node's j$^{\text{th}}$ finger interval to the expected size of an interval containing one node $(2^m/N)$:

$$
\begin{aligned}
2^{j-1} &= \lceil 2^m/N \rceil \\
\Rightarrow \qquad j &= \lceil \log 2^{m-1}/N \rceil = \lceil m + 1 - \log N \rceil
\end{aligned}
$$

Figures 6.1 and 6.2 show us how stretch affects fingers' resource levels and physical distances, with low stretch ($\tilde{c} = 8$) favoring lower physical distances and high stretch ($\tilde{c} = 1$) favoring higher resource levels in an apparent tradeoff. For a middle stretch of $\tilde{c} = 2$, a finger's expected resource level is doubled when there are 2^6 random nodes to choose from, while a mere 2^4 nodes are needed to reduce its expected physical distance by more than half. Note that the expected physical distance of a finger in a location unaware Chord is constant and given for $k = 1 = 2^0$, as is the expected resource level of both Chord and DHash++. Interpreting the results for the specific scenario in Figures 6.1 and 6.2, we expect that resource and location aware fingers cause a higher number of routing hops to be sent across high level, physically close nodes, resulting in less traffic on low level nodes, less cross-network traffic, and ultimately robuster lookups.

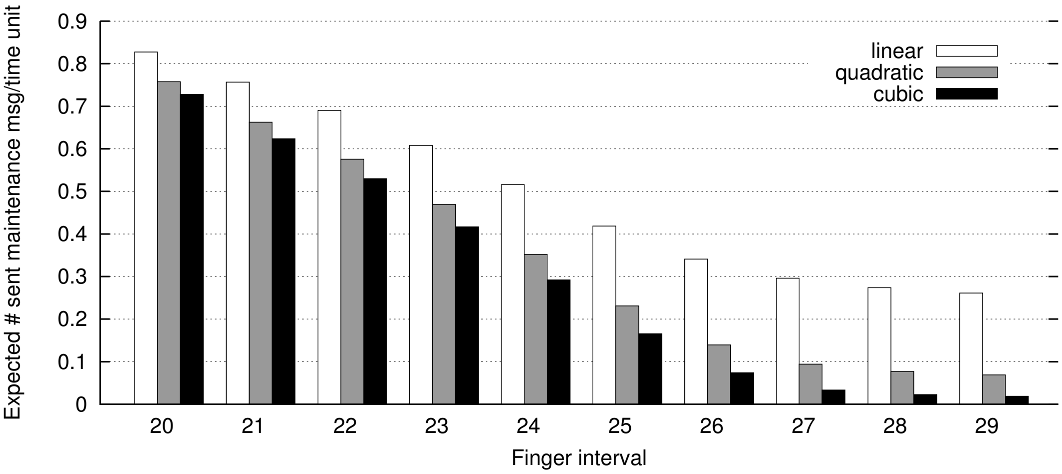


Figure 6.3: The expected number of sent messages per finger per time unit for linear, quadratic, and cubic finger maintenance intervals, with $10,000$ nodes, 32-bit keyspace, $l_{max} = 3$, $r = 10$, $\rho = 2$, and stretch $\tilde{c} = 2$. Note that the 20th finger interval in the first in which a node is expected.

6.1.2 Maintenance

The finger maintenance interval function $g(\ell)$ and (6.1) can be used to find the expected number of maintenance messages for a finger $F[i]$ per unit of time (see Figure 6.3). Let $f_i = x.F[i].node$ for simplicity and assume that there are the expected $k_i = \lfloor N \cdot 2^{i-m-1} \rfloor$ nodes in $B_{x,i}$ for $i \geq \lceil m + 1 - \log(N) \rceil$ and that they are known to x.

$$\begin{aligned} E(\#\text{ messages for } f_i) &= E\left(\frac{1}{g(R_{k_i,min})}\right) \\ &= \sum_{\ell=0}^{l_{max}} \frac{P(R_{k_i,min} = \ell)}{g(\ell)}. \end{aligned}$$

Figure 6.3 shows the expected number of maintenance messages sent per finger and time unit for a concrete scenario with 10,000 nodes and the following finger maintenance intervals:

$$g(f.node_R) = t_{ref} \cdot (f.node_R + 1)^{\beta}, \tag{6.6}$$

called *constant* for $\beta = 0$, *linear* for $\beta = 1$, *quadratic* for $\beta = 2$, and cubic for $\beta = 3$. Note that in Figure 6.3, a constant finger maintenance interval $g(\ell) = t_{ref}$ (i.e. as in Chord) would send one message per unit of time. This estimation concentrates only on sent messages, and determining the incoming links is much more complex. Figure 6.3 demonstrates how nodes using quadratic finger maintenance intervals require in expectancy significantly fewer messages as the finger intervals grow, with less than 50% of the maintenance messages necessary with a constant finger maintenance interval for the 23th finger interval (which corresponds to $2^4 = 16$ *known* nodes) and less than 20% after the 25th finger interval (which corresponds to $2^6 = 64$ *known* nodes). Since maintenance overhead accounts for a large portion of the total network load, we expect that this reduced maintenance overhead would lead to a significant increase in the nodes' lifetimes.

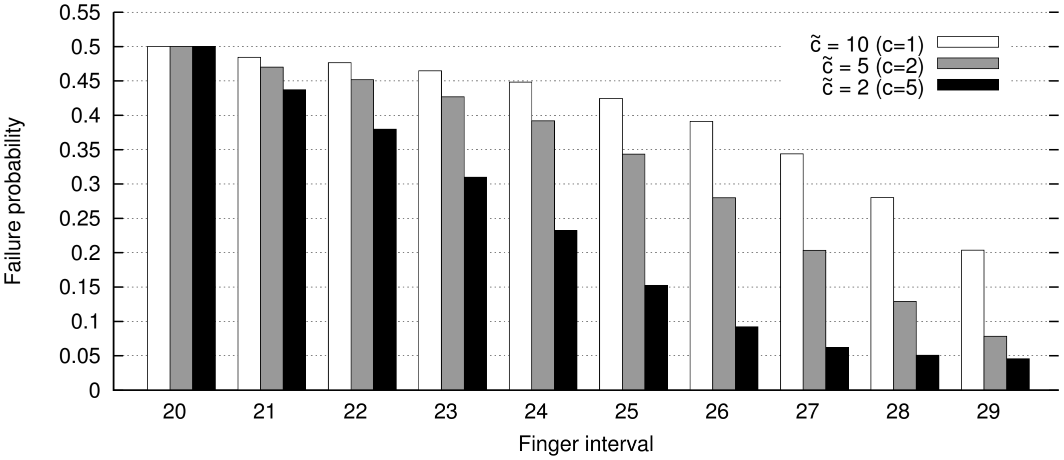


Figure 6.4: Failure probability for a finger in variable finger intervals with $10,000$ nodes, $5,000$ of which fail, 32-bit keyspace, $l_{max} = 3$, $r = 10$, $\rho = 2$, and variable stretch $\tilde{c} = 10, 5$, and 2.

The total expected number of maintenance messages $\tau(x)$ sent by a node x during one unit of time is found by summing over all of a node's fingers together with the maintenance delays s_0 and s_1 for predecessor and successor nodes, respectively:

$$\tau(x) = \frac{1}{s_0} + \frac{r}{s_1} + \sum_{i=\lceil m+1-\log N \rceil}^{m} \sum_{\ell=0}^{l_{max}} \frac{P(R_{k_i,min} = \ell)}{g(\ell)}$$

6.1.3 Failures

Recall that we use three failure scenarios, one of which uses nodes which fail simultaneously according to a given probability distribution. For this failure scenario, we consider the failure probability of a random node x's fingers given a fixed number of failures where the failure probabilities depend on nodes' resource levels. Using the conditional failure probabilities for the nodes from (2.3) and the resource probabilities for the fingers from (6.1), we have:

$$\begin{aligned} P(f_i \text{ fails}) &= \sum_{j=0}^{l_{max}} P(f_i \text{ fails and } (f_i)_R = j) \quad (6.7) \\ &= \sum_{j=0}^{l_{max}} P(f_i \text{ fails } |(f_i)_R = j) \cdot P((f_i)_R = j) \\ &= \sum_{j=0}^{l_{max}} P(F_x | x_R = j) \cdot P(R_{k_i,min} = j). \end{aligned}$$

Again using the example distance distribution from (6.4), Figure 6.4 shows an example scenario which demonstrates how the stretch affects fingers' failure probabilities. Note that for Chord and DHash++, this probability is constant for all fingers, resulting in fingers which are more likely to fail. Thus, we expect fewer finger failures for RBFM and thus a lower number of lookup failures.

6.2 Hierarchical HRM

While a node's links in RBFM have varying resource levels and distances - posing the question how these resources are distributed to the links and how these resources affect, for example, links' failure probabilities - HRM's links have deterministic hierarchy layers which reflect resource levels. In fact, resource levels are assigned to hierarchy layers in an arbitrary fashion, but for simplicity's sake, assume here that $x_R = x_H$ and $l_{max} = h_{max}$. Thus, we need not ask how resources are distributed to links or with which probability a given link might fail (with the exception of parent links), as we know that each node has fingers within its own hierarchy layer and links to the other hierarchy layers independent of the nodes' resource levels (i.e. deterministically determined). However, each hierarchy layer has location aware layer fingers, so the distribution of links' distances can be addressed similarly to RBFM. On the other hand, the expected maintenance messages sent per node, in contrast to RBFM, vary from layer to layer. The maintenance of layer fingers depends on the number of links that varying levels must maintain. And lastly, the failure probabilities of various levels' links, especially the non-deterministic leaf nodes' parent links, can be compared with those of RBFM links.

6.2.1 Expected Distance of Layer Fingers

Within each layer, nodes choose layer fingers based on location only, while the links' resource levels are dictated by the resource levels within that layer. However, an upper node x has fewer peers from which to choose fingers per finger interval than RBFM or DHash++, since only a fraction of the total network nodes are within layer x_H. Thus, HRM's location awareness is most certainly poorer than in DHash++ but it is unclear how it compares to RBFM, which also chooses farther fingers for the benefit of link strength. Assuming that $x_R = x_H$ and $l_{max} = h_{max}$ and using the resource probability function from Equation (2.1), there are an expected $p_\ell \cdot N$ nodes in layer ℓ. Thus, the first expected finger for a layer ℓ node is in the $\lceil m+1-\log(p_\ell \cdot N)\rceil$ finger interval. Furthermore, the expected number of nodes in the (layer) finger interval $B_{x,i}$ is

$$\left\lfloor \frac{p_\ell \cdot N \cdot 2^{i-1}}{2^m} \right\rfloor = \left\lfloor \frac{p_\ell \cdot N}{2^{m+1-i}} \right\rfloor .$$

Borrowing from Theorem 6.1.1, the pdf of a link's distance can be found. Assuming that the pdf $f_D(t)$ and cumulative distribution function $F_D(t)$ of physical nodes' distances from node x are given, then the pfd for the physical distance of the physically closest node in a random set of k nodes is (here derived from (6.2) with $h(\ell) = 0$):

$$\begin{aligned} f_{D_{min}}(t) &= k f_D(t) \sum_{\ell=0}^{l_{max}} p_\ell \Big(1 - \sum_{j=0}^{l_{max}} p_j F_D(t)\Big)^{k-1} \\ &= k f_D(t)(1 - F_D(t))^{k-1}. \end{aligned}$$

This pdf can also be applied to the example scenario from Section 6.1.3 with nodes distributed uniformly on a disk of radius r to compare the location awareness of HRM and RBFM:

$$f_{D_{min}}(t) = k\frac{2t}{r^2}\left(1 - \frac{t^2}{r^2}\right)^{k-1}$$

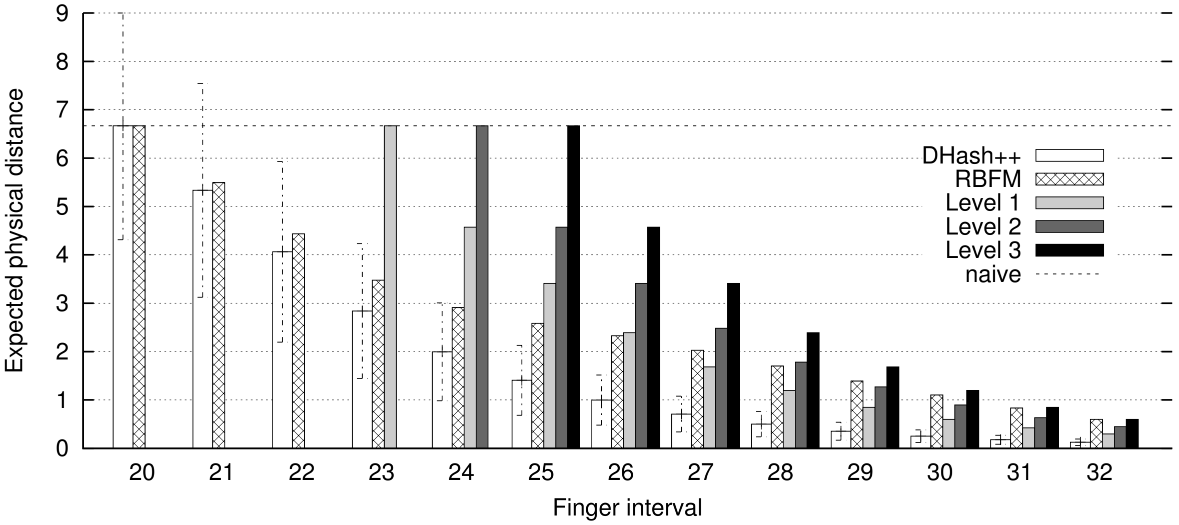


Figure 6.5: The expected physical distance of the i^{th} finger is shown for DHash++, a four level RBFM, and a the individual layers of a four layer HRM with $10,000$ nodes, $m = 32$, $\rho = 2$, and $r = 10$. For RBFM, the stretch value is set at $\tilde{c} = 1$. For HRM with $x_R = x_H$, the bottom level nodes have no layer fingers, so only levels $1, 2$, and 3 are shown.

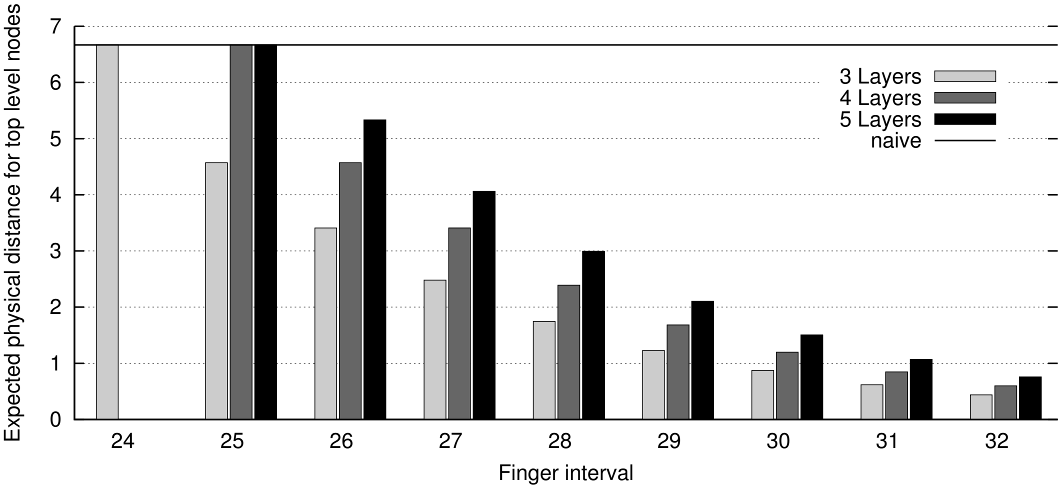


Figure 6.6: The expected physical distances of fingers in HRM are shown for the sparsest, i.e. top, layers of three different HRM with three, four, and five hierarchy layers with $x_R = x_H$. The setup used $10,000$ nodes, $m = 32$, $\rho = 2$, and $r = 10$.

Of course, when only observing sets of nodes of a given size, this HRM distribution has identical values as DHash++ as plotted in Figure 6.2. But since HRM has fewer nodes than RBFM for a given finger interval $B_{x,i}$ from which to select a (layer) finger, the finger interval sizes must also be taken into account. The expected distance of upper layer nodes' layer fingers has been plotted with regards to their finger intervals in Figure 6.5 using the example scenario with 10,000 nodes. A four layer HRM ($h_{max} = 3$) is used for this figure, and the respective distances for DHash++ and a four level RBFM are also shown for comparison. Note that the difference between DHash++, level 1, level 2, and level 3 resembles a shift within the finger intervals. For example, the expected physical distance achieved by DHash++ in the 21$^{\text{st}}$ finger interval is its second finger interval in which nodes can be expected to be found, corresponding with the 24$^{\text{th}}$ finger interval for level 1 nodes, the 25$^{\text{th}}$ finger interval for level 2 nodes, and the 26$^{\text{th}}$ interval for level 3 nodes. The standard deviation has thus been shown for DHash++ only, since it is quite similar for the respective fingers in HRM. RBFM, on the other hand, behaves differently due to its additional resource awareness, with a slower decrease in expected distance.

6.2.2 Maintenance

In contrast to RBFM, a node x's links, and thus its total expected maintenance load, depend on its own hierarchy layer x_H. If $p(\ell)$ is the probability that a random node belongs to hierarchy layer ℓ (giving us an expected ratio between the sizes of the layers), then the total expected maintenance load $\tau(x)$ can be expressed with the help of the maintenance delays from Table 5.1 and the number of layer fingers each node is expected to maintain. The first finger is expected for finger interval $\lceil m+1-\log(p_{x_R} \cdot N)\rceil$, but the last finger depends on the defined finger range given by $x.Frange$. For a layer 1 node, $x.Frange = x.I.closestInt$, in which case the expected number of fingers depends on the key of the first higher layer node. With an expected $N\sum_{\ell+1}^{l_{max}} p(i)$ nodes in higher layers randomly distributed in the key space, an interval must contain $2^m/(N\sum_{\ell+1}^{l_{max}} p(i))$ keys to be expected to contain at least one higher layer node. The first finger interval with at least this number of nodes is the interval $B_{x,j}$ with

$$j = \left\lceil m+1-\log\left(N\sum_{\ell+1}^{l_{max}} p(i)\right)\right\rceil.$$

Thus, the expected number of layer fingers for layer 1 nodes, which is referred to as θ_1, is given by the expected first interval that contains a layer 1 node and the expected last interval that does not contain a higher layer node:

$$\begin{aligned}\theta_1 &= \left\lceil m+1-\log\left(N\sum_{i=2}^{l_{max}} p(i)\right)\right\rceil - \lceil m+1-\log(Np_1)\rceil + 1\\ &= \lceil \log(Np_1)\rceil - \left\lceil \log\left(N\sum_{i=2}^{l_{max}} p(i)\right)\right\rceil + 1.\end{aligned}$$

On the other hand, a layer h_{max} node has fingers throughout the entire key space, so it has an expected

$$\theta_{l_{max}} = m - \lceil m+1-\log(Np_{l_{max}})\rceil + 1$$

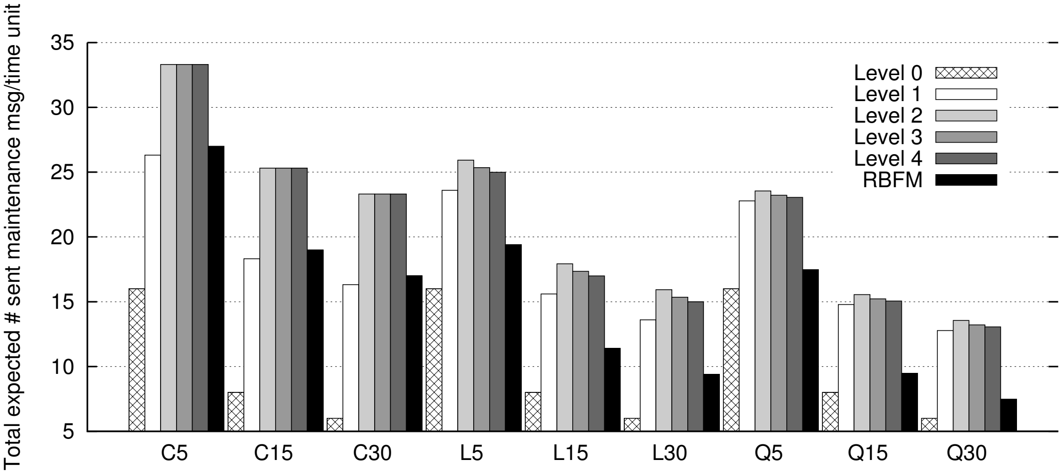


Figure 6.7: The total expected number of sent messages per node in a 5-level RBFM or one of 5 layers in HRM in a 60 second time frame. "C" notates a constant finger maintenance interval, "L" linear, and "Q" quadratic while the corresponding numbers refer to the values for $s_0 = 5, 15$, or 30. Calculations are based on $s_1 = 30$, $t_{ref} = 60$, 10,000 nodes, 32.bit keyspace, $l_{max} = 5$, $r = 10$, $\rho = 2$, stretch $\tilde{c} = 2$.

fingers. All of the other upper layers $1 < \ell < h_{max}$ are bound by $x.Frange$, which can be variably defined for $\ell = x_R$ as:

$$\theta_\ell = x.Frange - \min\{x.Frange, \lceil m + 1 - \log(Np_{l_{max}})\rceil\} + 1.$$

Furthermore, since there are an expected p_0N layer 0 nodes and $(1-p_0)N$ upper layer nodes, each upper layer node has responsibility for an expected

$$\theta_p = \frac{p_0N}{(1-p_0)N} = \frac{p_0}{1-p_0}$$

leaf nodes to which it maintains links. Thus, the total expected maintenance load per unit of time depends on a node's resource level (see Table 5.1):

$$\tau(x) = \begin{cases} \frac{1}{s_0} + (r+1)\frac{1}{s_1} & x_R = 0 \\ \frac{1}{s_0} + (r+2+\theta_p)\,\frac{1}{s_1} + \left(\sum\limits_{i=1, i\neq x_R}^{l_{max}} \frac{1}{g(i)}\right) + \theta_{x_R}\frac{1}{g(x_R)} & x_R > 0. \end{cases}$$

The example scenario used above with 10,000 nodes is used in Figure 6.7 to compare the total maintenance of the hierarchy layers in a 5-layer HRM with the total maintenance incurred by a 5-level RBFM. The values of s_0 and s_1 often dominate nodes' maintenance load, since these delays must be short enough to ensure that the ring structure remains intact in dynamic scenarios. In Figure 6.7, varying values of $s_0 = 5, 15$, and 30 are used, while $s_1 = 30$ and $t_{ref} = 60$ are held constant. However, the finger maintenance interval is varied between constant, linear, and quadratic as in Equation (6.6). The used used finger ranges were:

$$x.Frange = \begin{cases} x.I.closestInt & x_H = 1 \\ m - 1 & x_H = 2 \\ m & x_H = h_{max}. \end{cases}$$

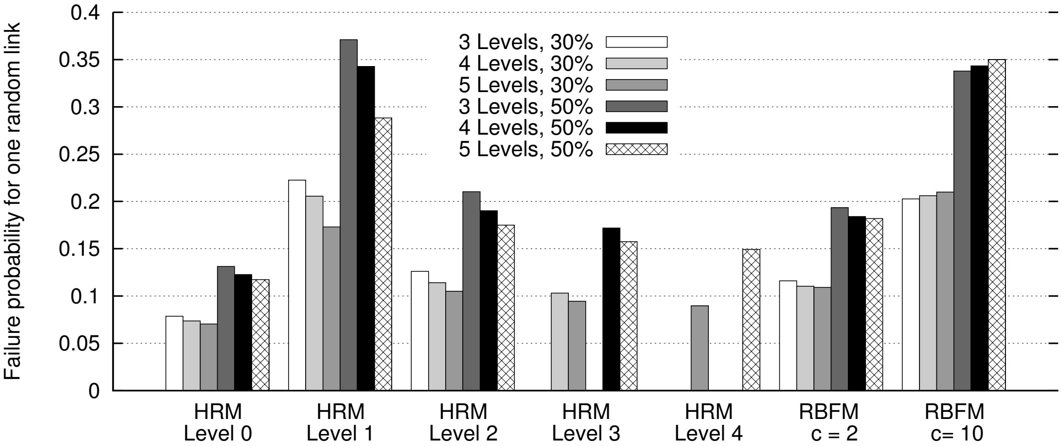


Figure 6.8: Approximate failure probability that one random node from a node's set of leaf link, inter-layer links, and layer fingers (or fingers for RBFM) fails. Calculated for 10,000 nodes, 50% or 30% of which fail, a 32-bit keyspace, $l_{max} \in \{3,4,5\}$, $r = 10$, $\rho = 2$, and variable stretch $\tilde{c} \in \{2,10\}$.

Note how the highest populated level 0 nodes consistently have significantly less maintenance load than the other levels or RBFM nodes. The upper nodes in HRM, however, have a higher load than RBFM as the result of their upper layer successor and predecessor and inter-layer links which require frequent maintenance. Indeed, these upper layer successor and predecessor links have a constant maintenance interval s_0 which keeps their load relatively high even for the lightest scenario "Q30." However, these upper level nodes account for only approximately 30% of nodes and these numbers account for outgoing links only - incoming links incur more load on stronger nodes for both HRM and RBFM. Note that the RBFM values correspond to the resource-unaware Chord or DHash++ for the constant finger maintenance intervals.

6.2.3 Failures

The probability that a given inter-layer link or layer finger will fail is already known for the above failure scenario (see Section 6.1.3), since these links' resource levels are known. However, leaf nodes' links to their parent nodes are non-deterministic and from any of the upper resource levels. The probability that a leaf node x's parent y is from a defined resource level ℓ is

$$P(L_y = \ell) = \frac{p_\ell}{\sum_{i=1}^{l_{max}} p_i}.$$

Analogously to Equation (6.7), the probability that a leaf node's parent fails is thus:

$$P(LF \text{ fails}) = \sum_{j=1}^{l_{max}} P(F_y | y_R = j) \cdot P(L_y = j)$$

with the failure probabilities from Equation (2.3). Using this leaf failure probability, Figure 6.8 shows an approximation for the probability that a randomly chosen link from the set of leaf, inter-layer, and layer finger links in HRM or a randomly chosen finger from RBFM fails. For this approximation, fixed numbers of leaf links, layer fingers (HRM), and fingers (RBFM) were assumed according to their respective expected values. The number of layers/resource levels was varied between 3, 4, and 5 and the number of total node failures ranged from 30% to 50% of the 10,000 network nodes. Level 0 HRM nodes have only one such link, so these values reflect only the probability that a parent node fails, but are significantly lower than the other HRM levels and the two RBFM configurations. Note the higher link-failure probability of level 1 HRM nodes, who maintain level 1 layer fingers. The link-failure in RBFM is strongly effected by the used stretch $\tilde{c}$, which determines the extent of resource awareness.

6.3 Summary

Comparing RBFM and HRM on a mathematical basis is difficult due to their varying construction and maintenance of links. While RBFM almost exclusively uses non-deterministic fingers, HRM's leaf and inter-layer links are deterministic. Moreover, these links also require a higher maintenance frequency to ensure the success of lookups (not just their efficiency), but this added load only affects upper layer nodes. Thus, bottom layer nodes experience a drop in the maintenance load for outgoing links as seen in Figure 6.7. While the expected physical distance of an RBFM node x's finger depends only on its finger interval and not on x_R, with higher resources and lower physical distances for increasingly farther intervals, in HRM fingers' physical distances depend on x's own hierarchy layer. More specifically, nodes in higher layers have fewer peers from which to choose physically close layer fingers and thus have less location awareness. With regards to link failure, RBFM link failure rates are reduced when resource awareness is increased through the stretch variable. Meanwhile, level 0 HRM nodes benefit from the lowest link failure rate tested, since they are linked to only one upper layer parent. Thus, HRM provides the failure and maintenance reduction for the highest populated bottom level, while RBFM provides more uniform node treatment when observing outgoing links.

These observation lead to several expectations for the evaluation in the following chapter. Since HRM bottom layer nodes have such dramatically lower load, we can expect these bottom nodes and thus HRM's total network to have longer lifetimes then RBFM. RBFM lifetimes should likewise increase as resource awareness is increased by the stretch variable. However, we also expect slightly longer paths in HRM, which increases the total network load and could also negatively affect node lifetimes. Furthermore, due to the lower location awareness is HRM's top layer (whose routing hops coincide with the long key-ranged hops over more aware nodes in RBFM) and the higher hop count per lookup, the physical distance traversed by lookup routing paths of RBFM can be expected to be significantly shorter than in HRM, but still longer than in DHash++. Unfortunately, it is difficult to predict which system might have more robustness with better lookup deliverability rates - while HRM's bottom layer nodes have only one link that can fail, and therefore lower average failure rates, leaf nodes become practically unavailable when their parent links are not intact.

Chapter 7

DHT Evaluation

As discussed in Chapter 6, many of the important evaluative measures can only be assessed with the help of simulation. In this chapter, four different simulation sets are used to compare RBFM and HRM with C-RBFM and the hybrid DHT as well as the standard Chord and DHash++. Their results help highlight the advantages and tradeoffs of the different systems, their applicability to highly dynamic scenarios with restricted resource availability, and general structural guidelines with regard to network behavior. When possible, these results are also compared with the analytical results from the previous chapter.

Each of the simulation sets uses various (multiple) simulation scenarios to evaluate specific aspects of the suggested DHTs. These scenarios are referred to as:

Static scenario. Nodes neither fail nor are added to the network after an initialization period. The network thus remains static.

Churn scenario Nodes are drained of their resources as they send and receive messages. Once a node's resources have been completely depleted, the node dies and a new node eventually joins the network in its place.

Drain scenario. Nodes are drained of their resources as they send and receive messages until they fail, but no new nodes are added. Thus, the time until a fixed percentage of network nodes fail can be measures as the system's lifetime.

Massive failure scenario. Nodes are static until a fixed large portion of the nodes simultaneously fail. These failures are dependent on node resource levels, such that weaker nodes have a higher probability of failure.

The four simulation sets each focus on a different aspect or DHT approach. In the following chapter, the results are broken apart on an evaluation measure basis, so that each measure or set of similar measures is addressed in a section that includes all applicable simulation sets. For clarity, each simulation set is given a name and a focus:

RBFM Comparison. These simulations focus on RBFM and the effects of its various parameters. It primarily uses a churn scenario, but results for a massive failure scenario in which 30% of nodes fail is briefly discussed.

			RBFM	Underlay	Appr. Comparison	Level Comparison	In Section
Maintenance	load	messages	✓		✓	✓	maintenance load
		forwarded msg	✓				
Lookup	hop count		✓		✓	✓	lookup distance
	latency	distance	✓		✓		
	stretch	underlay		✓			
	failure	overlay	✓		✓		lookup failure
Node	load	messages				✓	node load
		underlay msg		✓			
	lifetime		✓		✓	✓	load lifetime
Link	resources		✓	✓	✓		links
	latency				✓	✓	

Table 7.1: Measures used for varying evaluation scenarios.

Underlay Comparison. These simulations use a static scenario but concentrate on the effects on a possible cluster-based underlay network. The underlay hops necessary to complete (random) overlay lookups are observed along with the resources consumed by this underlay routing. C-DHash++ and C-RBFM are compared with RBFM, DHash++, and Chord.

Approach Comparison. These simulations compare RBFM, HRM, and the suggested hybrid DHT with Chord, DHash++, and two-tier DHTs, using a static and a drain scenario.

Level Comparison. These simulations focus on HRM and the effects of varying numbers of hierarchy layers on load, lifetimes, and latencies. They use a static and a drain scenario, varying the application and maintenance load so as to assess varying load profiles and the deliverability of messages under high load.

The evaluation measures used in each of these comparisons are listed in Table 7.1. In the following chapter, the simulation framework and setups of the evaluations are first discussed before the results are presented per evaluation measure (set). The measures are grouped together in the subsections maintenance load 7.2.1, lookup distances 7.2.2, lookup failure 7.2.3, node load 7.2.4, node lifetime 7.2.5, and links 7.2.6 as denoted in Table 7.1. By grouping the results from various simulation sets and scenarios together according to their measures, we have a broader scope with which to draw conclusions for each measure. Note that the comparative values for the lower resource levels are often the primary interest, since the highest number of nodes belong to these volatile levels.

7.1 Evaluation Platform and Setup

The simulations were performed in OmNET++ using the OverSim overlay framework [IB09]. The existing Chord implementation was extended for location and resource awareness for RBFM and significantly altered for HRM. For RBFM, this extension included adding resource levels to nodes which are automatically decremented when a message is sent or received, piggybacking resource levels and coordinates on messages, adding and maintaining prospective links lists, altering finger choices, adding resource-dependent finger maintenance intervals, and managing nodes' removals and additions once resources are depleted. For HRM, however, the entire routing structure was also changed, since nodes no longer use strictly greed routing. While the basic ring structure was maintained, nodes were separated into layers with specific layer routing for establishing and maintaining leaf links, inter-layer links, and layer fingers, and the prospective links lists were extended to save multiple layers.

When a node is created in the simulation, it is assigned an initial resource availability level in $\{0, \ldots, l_{max}\}$ and corresponding *resource values.* While the resource levels divide nodes into sets which we look at individually for resource-level-based results, the resource values provide a continuous range of values that can be incremented and decremented and mapped to resource levels. For example, nodes may have resource levels in $\{0, 1, 2, 3\}$ and resource values in $(0, 1600]$, and, with x_V referring to node x's resource value, the mapping of values to levels as:

$$x_R = \begin{cases} 3, & x_V = 1600 \\ 2, & 800 \leq x_V < 1600 \\ 1, & 200 \leq x_V < 800 \\ 0, & 0 < x_V < 200 \end{cases}$$

as in Table 7.2 for the approach comparison. For HRM, a node can be assigned to various layers in the hierarchy based on either its resource level or resource value. For example, as in HRM2:0-123 explained below, a node with resource level 3 may be assigned to layer 1 in the hierarchy. The number of hierarchy layers used is referred to as $h_{num} := h_{max} + 1$.

For the churn and drain scenarios, a node's resource value is drained by a fixed resource unit for each sent and received message if that node's resource level is less than l_{max}. Since top level nodes are assumed to have inexhaustible resources, they are excluded from resource drainage. The drain scenario, in contrast to the other three scenarios, has no fixed simulation time, and runs until a set portion of the nodes have failed and the network is thus considered failed. For the static and massive failure scenarios, on the other hand, resource drain is turned off and the simulations run for a set period of time. For the massive failure scenario, a fixed percentage of nodes are selected to fail simultaneously depending on their failure probabilities from Section 2.4.2.

		RBFM	Underlay	Appr. Comparison	Layer Comparison
Number of nodes	N	10000	1000	10000	1000
Node resource levels	$l_{max} + 1$	4	4	4	5
Creation resource value	Level 4	-	-	-	1600
	Level 3	800	800	1600	800-1599
	Level 2	400	400	800	400-799
	Level 1	200	200	200	200-399
	Level 0	100	100	100	100-199
Hierarchy layers	$h_{max} + 1$	-	-	4	3,4,5
Drain	Send	0.05	0.05	0.04	0.04
	Receive	0.05	0.05	0.02	0.02
Application frequency		60	60	30 sec.	10,35,45,60 sec.
Maintenance frequency	s_0	5	5	5 sec.	30 sec.
	s_1	-	-	20 sec.	60 sec.
	$s_2 = t_{ref}$	60	60	120 sec.	120 sec.
Simulation time		4000	5000	10000	20000
Rsc. distribution power	ρ	2	2	2	2
Maintenance interval	β	1,2	1,2	RBFM:2, HRM:1	1
Coordinates		OverSim	OverSim	random	random
Additional parameters		$k = 1, 3$	5000 clusters		

Table 7.2: Simulation variables used.

7.1.1 Configurations

Most of the configurations for the varying simulation sets can be found in Table 7.2, but specifics and expectations for each simulation are discussed in the following four subsections. All configurations use nodes with resource availability levels/values assigned to nodes based on the power of two Zipf distribution from Section 2.4.1. In addition to the maintenance messages listed in Table 5.1 using frequencies listed in Table 7.2, a dummy application on each node sent lookups at set intervals to randomly chosen keys. The dummy application interval is held constant for all but the layer comparison, which varies application load in order to achieve high, medium, and low ratios of the total number of maintenance messages to the total number of application messages sent. The application lookups are observed for the number of hops, hop distances, and successful delivery. The lower the rate of delivery, the less reliable the DHT stores and retrieves data.

7.1.2 RBFM Comparison

This comparison uses Chord, DHash++, and RBFM. The OverSim framework provides a coordinate set with over 200,000 coordinates, which was used for both the random and underlay comparisons with a resulting network coordinate diameter of approximately 2,000. Both linear and quadratic finger maintenance functions, $g(x_R) = t_{ref} \cdot (x_R + 1)^\beta$ for $\beta \in \{1, 2\}$, were used as well as varying sizes of the prospective links lists $k \in \{1, 3\}$. After nodes within the churn scenario fail, new nodes are added to ensure that there are a total of 10,000 nodes with a quadratic Zipf-distribution of their resource levels. Thus, a total of between 34,000 and 59,000 nodes (depending on the approach) were introduced to the network over the measurement period of 4,000 seconds. All results are based on the means of three runs which yielded negligible variances. This comparison evaluates the benefits of RBFM's resource usage by focusing on the direct measures of node lifetimes and lookup failures as well as the indirect measures of the mean resource levels and physical distances used for routing and the total number of maintenance messages sent. Expectations for this simulation included:

- Smaller prospective links lists prevent nodes from using stale node information of peers that have moved, changed resource levels, or perhaps since failed, while longer lists provide more alternatives when nodes *do* fail. Thus, failure rates should be lower for smaller lists while resource and location awareness are better for longer lists.

- Less use of lower level nodes for routing in RBFM.

- Decreased physical lookup distances and thus network load in RBFM compared with Chord, but increased compared with DHash++.

- Reduced link failure and increased node lifetime in RBFM due to lower nodes' reduced load.

Indeed, the results support the assumption that better finger characteristics do indeed result in improvements in global behavior.

7.1.3 Underlay Comparison

For the underlay comparison, the overlay lookup hops generated by RBFM in the existing OnNET++ simulation were passed into a MATLAB environment called SONIR (Self-Organizing Network with Intelligent Relaying) [ZGA$^+$10] for the simulation of underlay routing. SONIR is a purpose-built framework for simulating cluster-based ad-hoc networks with cooperative multi-hop relaying. The coordinates and resource levels of the nodes used in SONIR were identical to those used for the overlay simulation. The underlay simulation divided the nodes into location-based clusters, either with or without resource awareness for the choice of clusterheads. Nodes then collect information needed for routing via extensive piloting, so that the route from a source to a destination can be established by one of multiple possible routing protocols. Of those routing protocols, AODV (Ad-hoc On-demand Distance Vector) was used, since it is a popular, scalable and loop free reactive routing protocol widely used in both simulation tasks as well as in real applications [BBB09]. Expectations for this simulation include:

- Improved location awareness with fewer underlay hops per lookup and total underlay hops for RBFM compared with Chord, slight increase compared with DHash++.
- Even further location aware improvement for cluster-based approaches.
- Less use of resources per lookup for both normal and cluster-based RBFM when compared to Chord and (cluster-based) DHash++.

7.1.4 Approach Comparison

For the approach comparison, nodes with four resource availability levels are used to compare Chord, RBFM, HRM, the hybrid RBFM/HRM from Section 5.4, and a basic two-tier hierarchical overlay. Thus, the four resource levels are assigned to either four hierarchical layers (as in HRM, thus called HRM4), two hierarchical layers (as in the two-tier overlays), or one layer (as in Chord and RBFM). Due to the small variation caused by different finger maintenance intervals and stretch constants in RBFM in comparison to the larger variations between approaches, a single fixed RBFM configuration is used with stretch $c = 90$ and quadratic finger maintenance. The HRM4 and hybrid HRM approaches, on the other hand, use the load-heavier linear finger maintenance. HRM4 assigns nodes to layers with $x_H = x_R$ and uses finger ranges as described in Section 5.3.2:

$$x.Frange = \begin{cases} x.I.closestInt,\ x_R = 1 \\ m,\ x_R \in \{2,3\}. \end{cases}$$

The basic location aware two-tier hierarchies place weak nodes in the leaf layer and strong nodes in the single super peer layer. Using four resource levels ($l_{max} = 3$), nodes are assigned to layers in two fashions: For HRM2:0-123, level 0 nodes are assigned to the leaf layer and level 1,2, and 3 nodes to the super peer layer, while for HRM2:01-23 level 0 and 1 nodes are assigned to the leaf layer and level 2 and 3 nodes to the super peer layer. In contrast to the hybrid approach, nodes within the super peer layer are unaware of their varying resource levels and choose their fingers based only on physical distance. For the hybrid approach, only layer 0 nodes are assigned to the leaf layer to provide a large number of nodes from which

		Approach Comp.			Layer Comparison		
		HRM4	HRM2:0-123	HRM2:01-23	$h_{num} = 5$	$h_{num} = 4$	$h_{num} = 3$
(Expected) Number of nodes in hierarchy	Layer 4	-	-	-	27	-	-
	Layer 3	439	-	-	43	27	-
	Layer 2	780	-	-	76	62	27
	Layer 1	1756	2976	1220	171	142.5	100
	Layer 0	7024	7024	8780	683	768.5	873

Table 7.3: In the layer comparison, nodes from five resource availability levels are distributed amongst five, four, and three hierarchical layers.

resource and location aware fingers can be chosen in the super peer layer. Thus, it is called HRM2:0-123 with RBFM and uses the same setup as HRM2:0-123 with the addition of a stretch-constant $c = 90$ for its RBFM fingers. The expected number of nodes per hierarchy layer upon network initialization for HRM4 and the two-tier hierarchies are listed in Table 7.3. Expectations for this simulation include:

- Hierarchical approaches should be best a relieving weak (leaf) nodes of routing and maintenance load, thus prolonging their average lifetimes. This, in turn, prolongs the network lifetime.
- Physical distances for HRM may be longer than for RBFM (due to analytical results) and even the hybrid hierarchies (since nodes also have more peers from which to choose links), causing an increase in overall network load. However, leaf HRM nodes should still have decreased load.
- RBFM has a very adaptable structure, since a node's links do not depend on its resource level, and may thus adapt better to network dynamics, and a lower lookup failure rate.

7.1.5 Layer Comparison

The layer comparison is based on nodes from five resource availability levels ($l_{max} = 5$) that are distributed among three, four, and five hierarchical layers ($h_{num} \in \{3, 4, 5\}$). In order to compare hierarchies with varying numbers of layers, we must ensure that the underlying network starts in the same state for each configuration. Therefore, nodes' initial resource values and the number of nodes with inexhaustible resources must remain identical. Thus, instead of setting $l_{max} = h_{num} - 1$, resource values are *always* assigned across five resource availability levels for $h_{num} = 3, 4$, and 5, and then varying resource value thresholds assign nodes to hierarchical layers. In this way, five resource levels are, for example, assigned to three hierarchy layers. A node x that has been assigned the resource level x_R chooses its resource value at random from the corresponding interval listed in Table 7.2. Each value of h_{num} uses its own set of resource value thresholds to determine which hierarchy layer each node is placed in. The expected number of nodes per hierarchy layer upon network initialization is listed in Table 7.3 for $h_{num} \in \{3, 4, 5\}$. Note that the assignment of nodes to layers follows a quadratic Zipf distribution for $h_{num} = 5$ only.

Maintenance accounts for a large portion of total network traffic and is distributed differently among the layers than application lookup routing. In order to compare the scenarios where either maintenance or lookup load dominated, we use four different application loads with a constant maintenance load (as in Table 7.2). The ratio of application lookup load to maintenance load was thus varied using the following configurations:

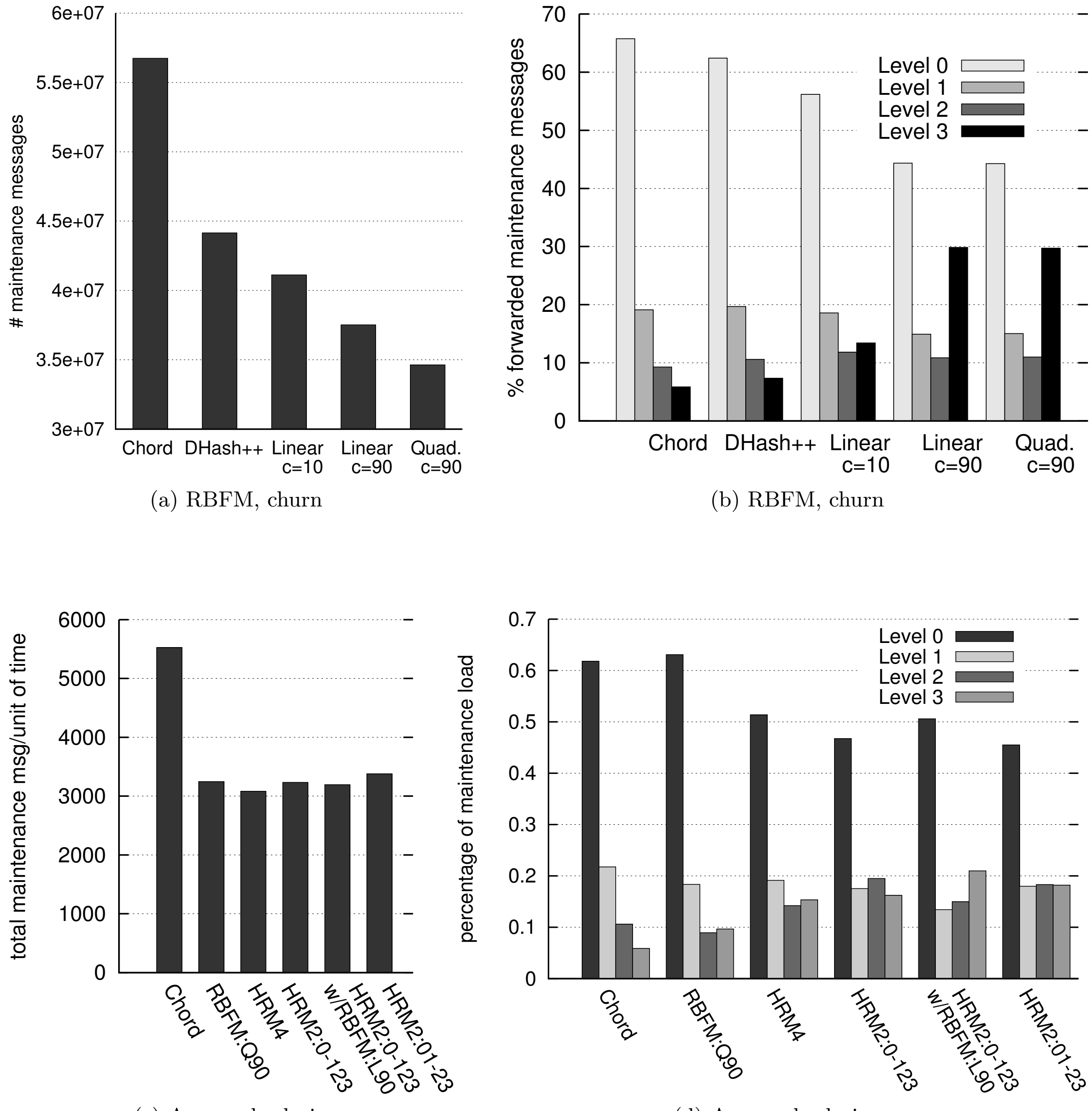
Notation	Delay	Total Application:Maintenance Message Ratio	
		static configuration	drain configuration
low	60 sec.	4:5	3:5
mid	45 sec.	1:1	9:11
high	35 sec.	7:5	1:1
high+	10 sec.	24:5	7:2

While results were expected to vary for different configurations, the overwhelming negative effect of additional hierarchy layers was surprisingly clear. Expectations for this simulation include:

- Fewer layers should incur fewer lookup hops, thus reducing overall network traffic.
- Fewer layers give nodes more peers from which to choose nearby layer fingers, thus improving location awareness.
- Fewer layers provide poorer resource awareness, since nodes are differentiated on a less fine scale. Thus, load may be less differentiated between resource levels for fewer layers. Furthermore, lookup failure rate may be higher, since peers are less prepared for individual nodes' failure.
- Higher application load increases the importance of resource aware routing. The varying load patterns of maintenance and lookup routing may cause various number of layers to have better node lifetimes.

7.2 Results

Results are discussed per evaluation measure across all simulation sets. Figures have thus also been grouped by measure and are marked with their respective simulation set (RBFM, Underlay, Approach, Layer) and scenario (static, churn, drain).



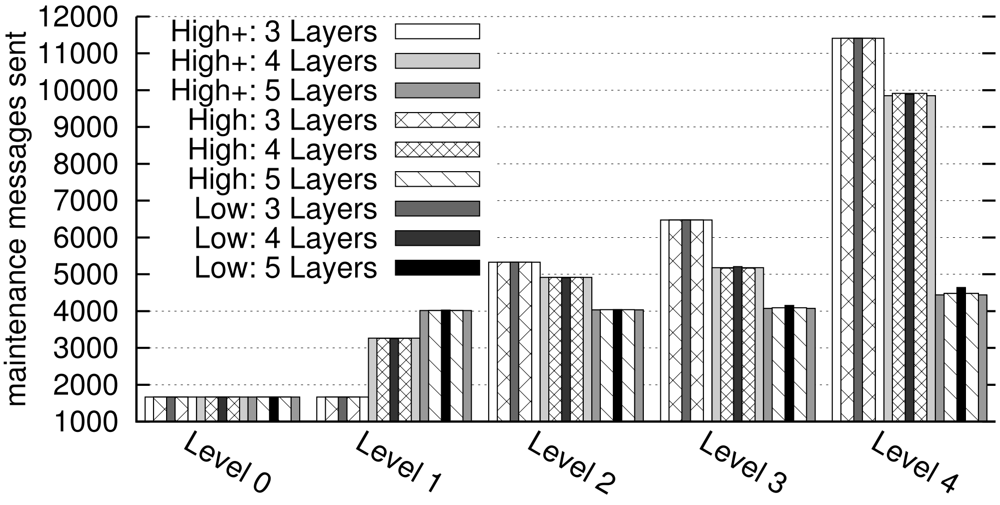
(a) RBFM, churn

(b) RBFM, churn

(c) Approach, drain

(d) Approach, drain

Figure 7.1: **Maintenance load** Quadratic finger maintenance and $k = 1$. (a) Total number of maintenance messages sent. (b) Percent of the total *forwarded* maintenance messages that each level forwarded. (c) Total number of maintenance messages sent per second in system. (d) Percentage of total messages sent per level.



(a) Layer, static

Figure 7.2: **Maintenance load** Average number of maintenance messages sent per node with resource availability levels 0 to 4. Maintenance is more or less constant regardless of the application load.

7.2.1 Maintenance Load

Figures 7.1 and 7.2 show the maintenance load results, ranging from the total number of maintenance messages sent, to the percentage of maintenance messages sent per resource level, to the percentage of maintenance messages that were forwarded (i.e. not delivered in one overlay hop) per resource level.

Figure 7.1a demonstrates the RBFM reduction in maintenance messages, with a decreasing tendency for increasing stretch (i.e. increasing resource-awareness integrated into the resource distance) and infrequent finger maintenance interval. Considering the large number of nodes in the bottom level, level 0 is responsible for the majority of the maintenance load in each system. The number of maintenance messages per second varies depending on the protocol and node failure, but Chord consistently has the highest maintenance load. Figure 7.1b, on the other hand, shows only the maintenance traffic which is not delivered within one hop and therefore creates extra network routing/lookup load. The figure indicates that the percentage of hops routed over lower levels is significantly reduced for RBFM, depending on the stretch. As expected from the analysis, a substantial portion of forwarding responsibilities have been transfered from lower level to higher level nodes.

Meanwhile, Figure 7.1c demonstrates that the difference in maintenance load between the flat and hierarchical structures is in fact small, despite the more frequently required maintenance in the hierarchal DHTs for leaf nodes and upper layer predecessors and successors. Figure 7.1d, on the other hand, shows the percentage of total maintenance load per level (as opposed to forwarded load in Figure 7.1b). Interestingly, RBFM level 0 nodes actually have a slightly higher overall percentage of maintenance load than in Chord, while level 1 loads have a slightly lower percentage of load. This is however, perhaps due to the lifetime lengths of nodes and therefore the number of nodes that persist in level 0. As we will see in Figure 7.6b, level 0 Chord nodes have substantially shorter lifetimes and thus create a substantially lower number of messages in relation to the other levels. In this we see a reoccurring problem in level-based measures in dynamic scenarios, where multiple variables make reliable interpretation difficult. Note, however, that there is a significant drop in total level 0 maintenance load for the hierarchical approaches which cannot be similarly explained, but rather is the result of leaf nodes' low number of links.

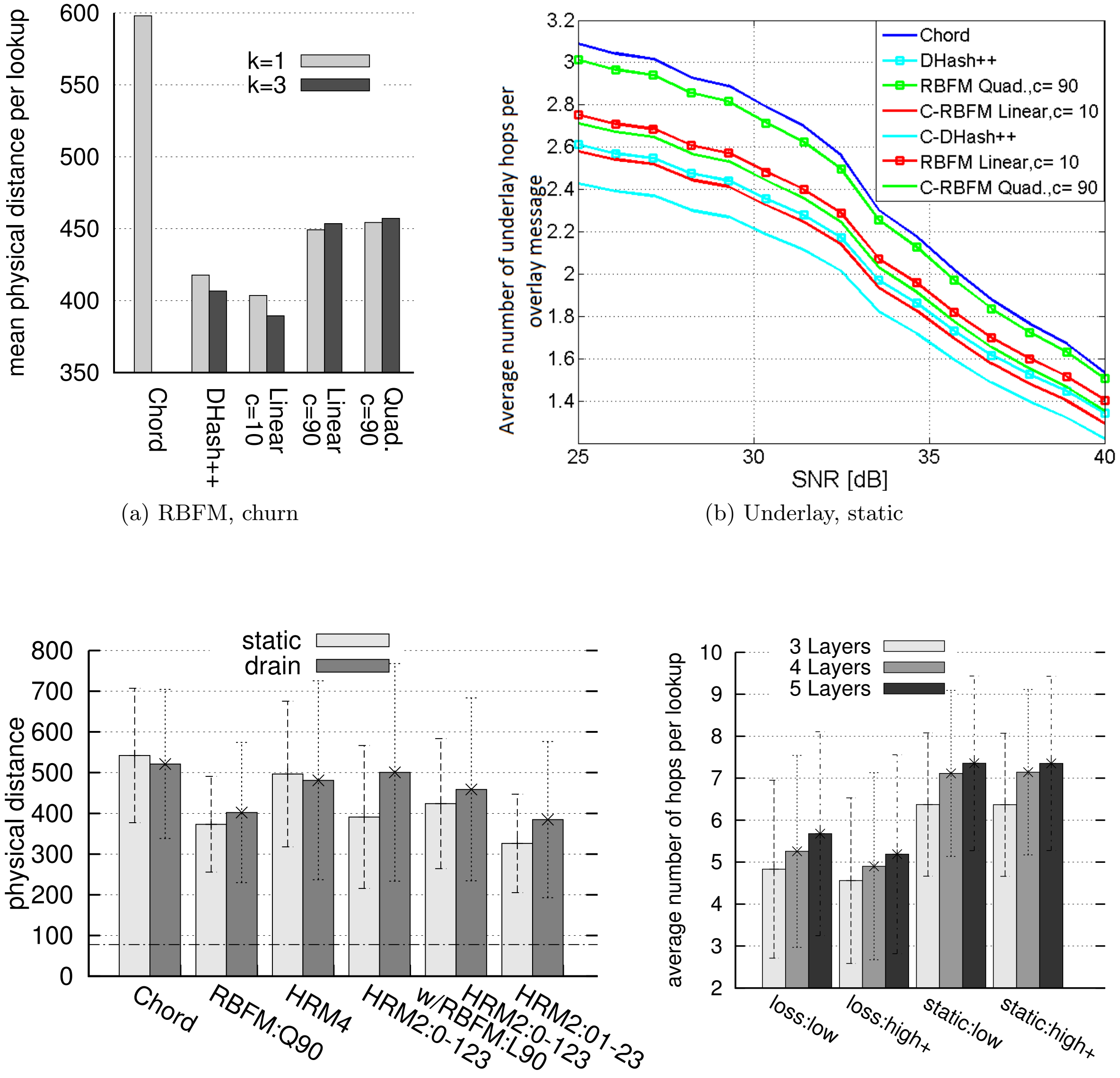
Figure 7.2 compares the number of maintenance messages sent for varying numbers of hierarchy layers and maintenance to application load ratio. As might be expected, the maintenance is not significantly effected by application load. However, while maintenance load remains low for level 0 nodes, it increases strongly for level 1 nodes with increasing number of hierarchy layers. On the other hand, level 2, 3, and 4 nodes' maintenance load is strongly decreased. Thus, splitting the network into more hierarchy layers decreases the maintenance load of strong nodes while increasing the load of weak upper layer nodes. This may, however, be beneficial by freeing top layer nodes' capacities for heavier lookup responsibilities.

7.2.2 Lookup Distance

For both the massive failure and the churn scenarios in RBFM, the differences between the approaches' various average hop counts for lookups were nearly negligible. The average hop count varied by at the most 5% with 6.5 to 6.8 hops, and the standard deviation of hops per lookup was, at its highest, 1.8. The mean physical distance traveled by all application lookups is shown in Figure 7.3a. While the increased physical distance for increased stretch (and thus increased resource awareness) in RBFM was expected, the lower value of the linear RBFM with stretch $c = 10$ than DHash++ was not. Since RBFM with any stretch constant > 0 forfeits a bit of location awareness, this must be related to the network churn. With slightly shorter lifetimes (see Figure 7.6a), DHash++ nodes have less time to find and establish the best possible fingers, which then also fail sooner. After all, a node can only choose a finger from the peers which it knows through the information piggybacked on messages. This indicates that lower node lifetimes also inhibit the systems' ability to implement location (or resource) awareness.

In contrast to the RBFM results, the underlay results in Figure 7.3b based on a static network do indeed reflect that DHash++ is more location aware than RBFM. Furthermore, each cluster-based approach uses significantly fewer underlay hops than its cluster-naive counterpart. The signal to noise ratio indicates how sparse the network is, with lower signal to noise ratios indicating sparser networks. The location aware benefits are even more pronounced for sparse clustered networks in which more underlay hops are necessary.

In the approach comparison, average lookup hop lengths ranged from 6.5 to 8.5 hops. The hierarchical approaches tended to have approximately one hop more than the other approaches, presumably from the final and/or initial hops to and from leaf nodes, as discussed in Chapter 5. Figure 7.3b suggests, again as expected, that the location awareness provided by RBFM mostly surpasses the location awareness of a hierarchical structure with sparse hierarchy layers from which to choose links. Note that the drawn line represents the actual mean physical distance between the sender and receiver of a lookup. HRM2:01-23 does, however, surpass RBFM, since it chooses links based only on location within its parent hierarchy layer. The difference between HRM2:01-23 and HRM2:0-123, on the other hand demonstrates how more



(a) RBFM, churn

(b) Underlay, static

(c) Approach, static and drain

(d) Layer, static and drain

Figure 7.3: **Lookup distance** (a) Mean physical distance traveled by a lookup. (b) Number of cluster-based underlay hops per overlay hop. (c) Average number and standard deviation of overlay hops per application lookup. (d) Mean physical distance traveled by a lookup.

possible nodes from which to choose fingers do not necessarily improve location awareness: HRM2:0-123, although it has more nodes in its upper layer, has a higher variation of physical lookup lengths, perhaps due to the increased number of hops necessary to forward messages in this more populated parent layer.

Increasing the number of hierarchy layers also increases the number of necessary hops per lookup as shown in Figure 7.3d. Note that the hop number is higher for the static than for the drain scenario, presumably because the overall size of the upper overlay which routes lookups is quickly reduced as nodes loose energy and switch to lower layers in the drain scenario (see Figure 7.6c). In fact, the drain scenario with higher application load and thus lower lifetimes has an even lower number of hops per lookup.

7.2.3 Lookup Failure

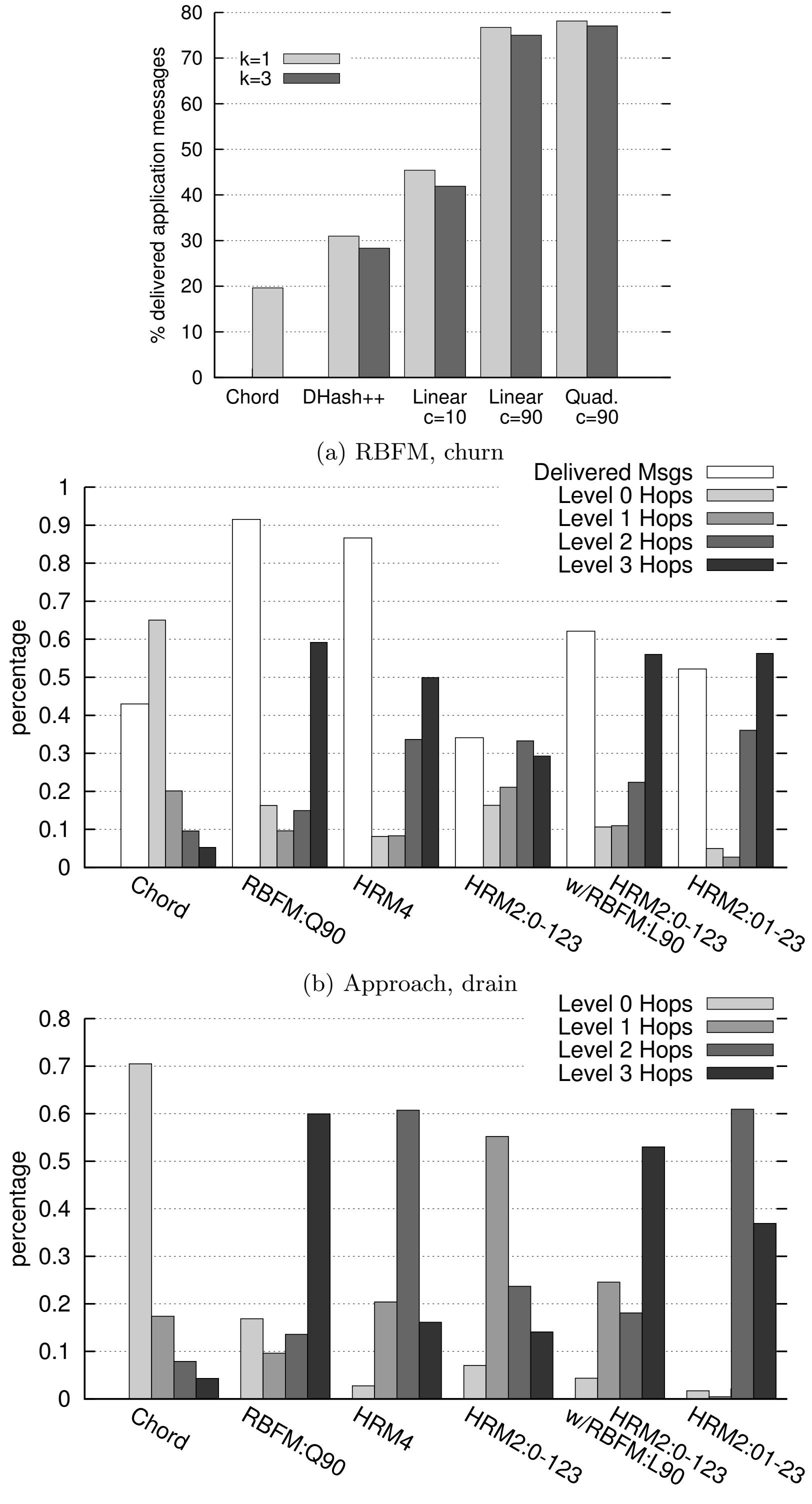
One of the most direct result for robustness is the application lookup deliverability. The results for RBFM with churn in Figure 7.4a were slightly surprising in that the resource-naive DHash++ performed significantly better with respect to resources than Chord although one could expected both to behave similarly. However, this improvement can be explained by DHash++'s prospective links list. Nodes with higher resource levels also have longer lifetimes, making then more well known to other nodes by their continually backpacked information. Since this information can propagate longer, these higher level nodes will invariably take up a larger portion of the prospective fingers lists. and thus, increase their number of incoming links. The result is an indirect resource-awareness which improves lookup deliverability.

One result from the RBFM with massive failures is also noteworthy: Message failures were comparable at around $20 - 25\%$ for Chord, DHash++, and RBFM with a linear finger maintenance interval for stretch $c = 10, 90$, but surprisingly poor $(45 - 55\%)$ for the quadratic finger maintenance function for $c = 90$. This indicates the importance of adequately frequent finger maintenance for all levels when large scale failures can be expected.

The lookup failure rates for the basic two-tier hierarchies in Figure 7.4b were surprisingly high, with lookup success rates reduced to under 65%. While Figure 7.6b shows how their averaged node lifetimes are actually longer, the lack of more than leaf-parent resource awareness appears to affect them just as strongly as the completely resource naive Chord.

7.2.4 Node Load

Figures 7.4b and 7.4c provide an overview of the lookup hop load distribution among the resource levels. Only successful lookups are included in these figures. Figure 7.4c clearly shows how strongly each approach prefers a single resource level for its lookup hops and demonstrates how important it is to design an overlay with the nodes' resource availabilities in mind. For example, RBFM's lookup hop distribution is more suitable for networks in which top level nodes can handle unlimited load while HRM4 is more suitable for systems with strong nodes in level 2 that should be used to reduce top level load. Note the high load on level 1 nodes for the two-tier approach HRM2:0-123 in Figure 7.4c. This may cause a high rate of node movement between the super peer and leaf layers, triggering HRM2:0-123's low deliverabiliy rate. Similarly, HRM4 distributes more load to level 2 nodes at the cost of their average lifetimes (relative to the other levels) as seen in Figure 7.6b.

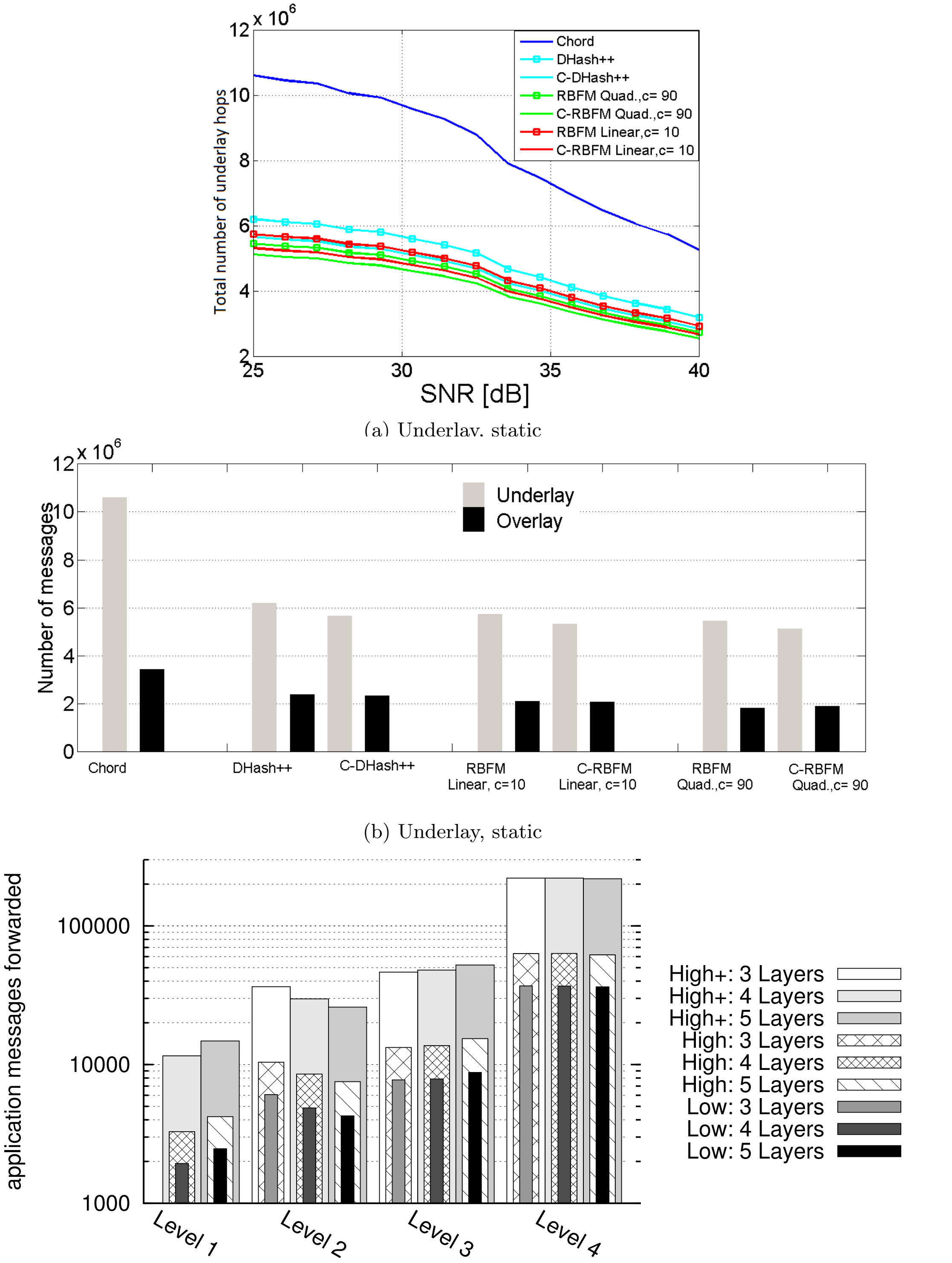


(a) RBFM, churn

(b) Approach, drain

(c) Approach, static

Figure 7.4: **Lookup failures/Node load** (a) Percentage of delivered application lookups using prospective links lists with one and three entries. (b) Percentage of delivered application messages and percentage of total lookup hops resolved per resource level. (c) Percentage of total lookup hops resolved per resource level. Each approach delivered more than 99% of lookups.



(a) Underlay, static

(b) Underlay, static

(c) Layer, static

Figure 7.5: **Node load** (a) Total number of underlay hops for application and maintenance messages. (b) Number of underlay hops shown with corresponding overlay hops for fixed SNR (25 dB). (c) Average number of application messages forwarded per node. Nodes in hierarchy layer 0 (i.e. level 0 nodes, level 1 nodes for 3 layers) do not forward messages.

Figure 7.5a reflects the findings regarding the mean number of underlay hops per single overlay hop, but additionally includes maintenance messages. The corresponding cumulative number of total overlay and underlay messages sent is shown in Figure 7.5b. These reflect what is expected of ad hoc networks, namely that location awareness provides an overall reduction of network load. Clearly, the lowest load is incurred by the cluster-aware approaches.

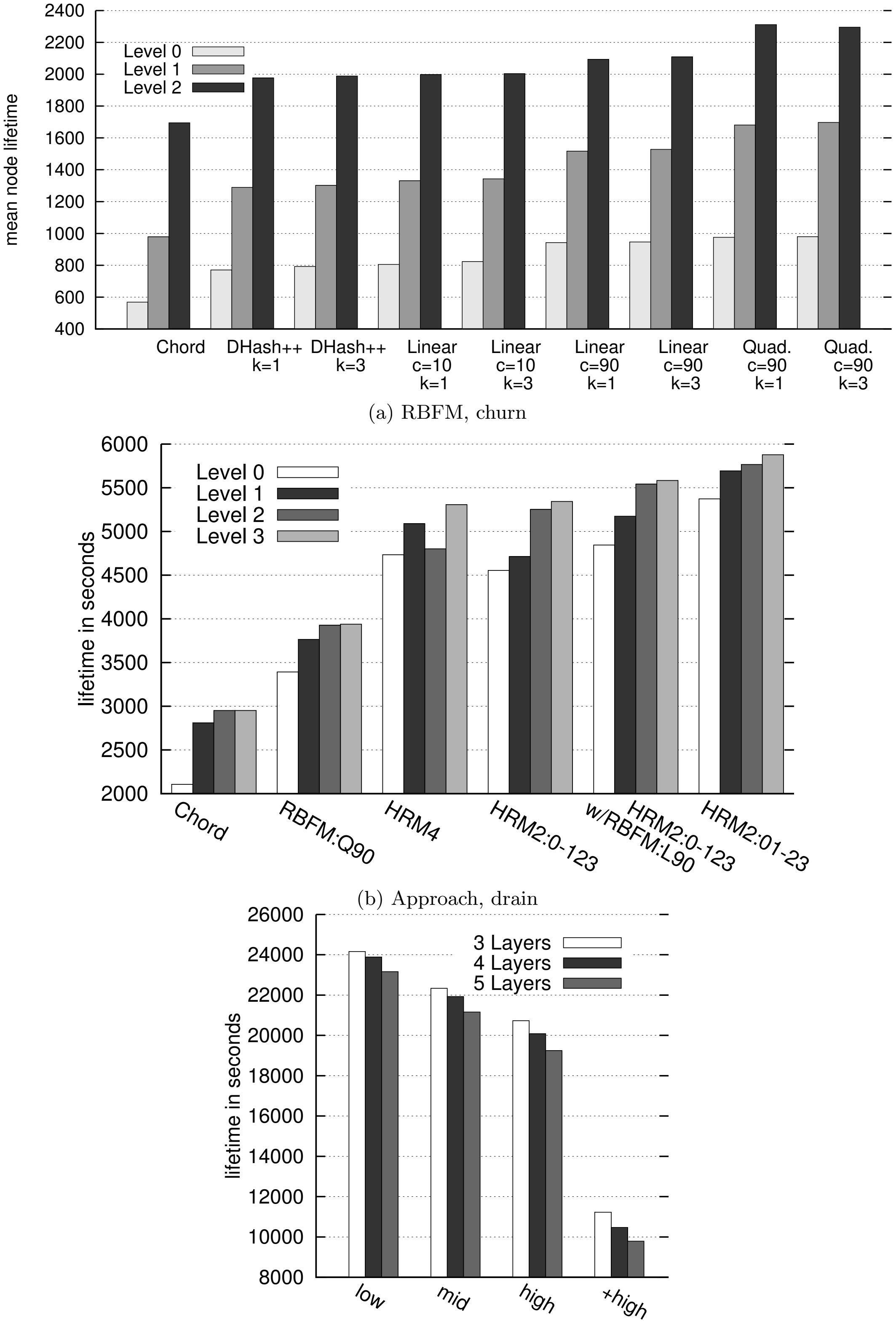
The total number of application messages routed in HRM is shown in Figure 7.5c for varying numbers of hierarchy layers. It shows how the increasing and decreasing tendencies for application load per node over the levels and the number of layers h_{num} are preserved for varying application:maintenance ratios, and similar to the shifts in maintenance load from Figure 7.2. Note that the increases and decreases in the level maintenance and application load for increasing h_{num} can be explained by the shifts of nodes to higher hierarchy layers caused by the chosen thresholds. Level 1 nodes, for example, are continually shifted upwards in the hierarchy, thus assuming more load. And while level 2,3, and 4 nodes also shift upwards in the hierarchy, their load is increasingly shared with the numerous level 1 nodes that have shifted up to take more load, and thus deceased. Level 3 nodes experience a slight increase in application load for increasing h_{num}, perhaps due to the lack of nodes shifting up into its layers (there are fewer level 2 nodes to shift upwards) as level 3 nodes shift up themselves.

7.2.5 Node Lifetime

The mean node lifetime for RBFM in Figure 7.6a has a similar behavior with regard to DHash++ and Chord as with the lookup failure rate in Figure 7.4a: DHash++, although it is not resource aware, has longer node lifetimes. This again can be explained by the use of prospective links lists, since finger maintenance is often routed directly to an existing finger in the prospective links list, generating less maintenance overhead and thus less resource usage (as also seen in Figure 7.1a). Otherwise, the highest mean node lifetime is achieved by the RBFM configuration which sends the most infrequent of the tested maintenance messages: the quadratic finger maintenance interval.

The lifetimes of each of the approaches before half of its nodes fail, as shown in Figure 7.6b, vary strongly. Despite a very infrequent finger maintenance interval for Chord, set here intentionally to 240 seconds to reduce maintenance load, it cannot compare to the alternative approaches. Note that while HRM4 performs significantly better than RBFM, the two-tier approaches perform even better, most likely due to the higher average hop length of lookups in HRM (approximately one hop more per lookup). While Figure 7.6b demonstrates how well the traditional two tier approaches prolong node lifetimes, Figure 7.4b shows that these two-tier approaches' performance suffers given high failure rates, reducing the success rate of lookups to under 65%.

When the number of hierarchy layers is considered, the upwards shift from level 1 nodes that reduces the load on level 2-4 nodes with an increasing number of layers, ultimately decreases these level 1 nodes' and thus the network's lifetime. As we see in Figure 7.6c, this upwards shift occurs regardless of the application load. Weaker nodes' lifetimes are bounded by the periodic maintenance and application messages that they (and every other node) send, providing an upper bound for network lifetime. Thus, if additional load is added to these weak nodes, this bound is reduced, shortening the possible network lifetime.



(a) RBFM, churn

(b) Approach, drain

(c) Layer, drain

Figure 7.6: **Node lifetime** (a) Mean node lifetimes per resource level for $k = 1$. (b) Mean node lifetimes per resource level. Level 3 is also total system lifetime. (c) Mean network lifetime until half of the nodes have failed.

7.2.6 Links

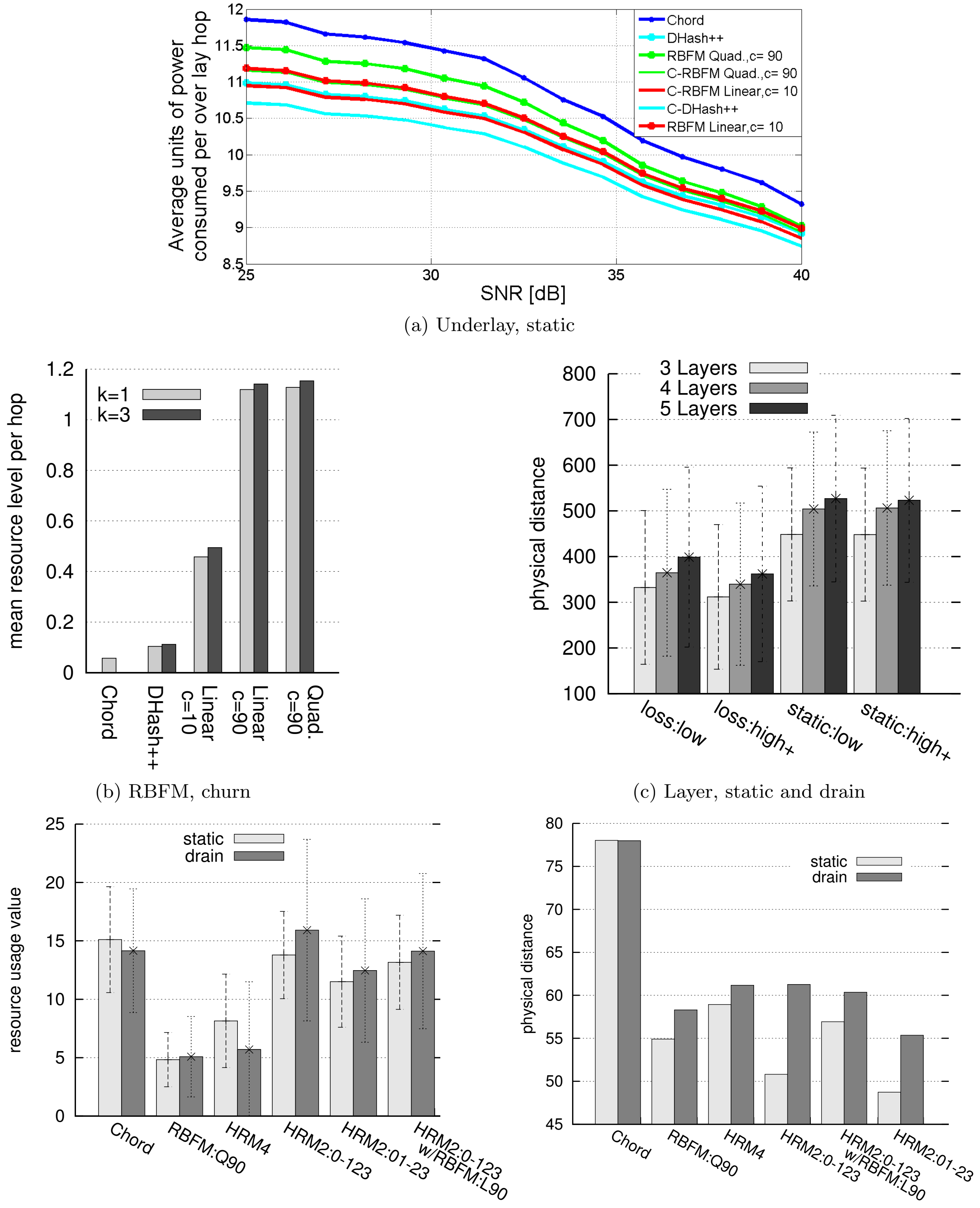
Links are considered with respect to both their physical distance and resource levels. However, "hops" might be a better term, since it is the mean distance and resources for hops taken that is observed. This means simply that links that are more heavily used influence the mean more significantly, giving us a weighted mean for the links.

The increase in mean resource levels per hop in Figure 7.7b for RBFM surpassed the expectations from the analysis. This increase is due to a combination of the higher resource fingers as in Figure 6.1 and the fact that higher fingers are used more frequently in routing. Note the increase in mean resource level of DHash++ compared to Chord, which was also apparent for mean node lifetimes and the lookup success rates. Furthermore, the resource awareness clearly depend on stretch but is not significantly effected by the finger maintenance interval or size of the prospective links lists.

Figures 7.7a and 7.7d measure the resources used by the hops of a path using a resource utilization measure. The resource utilization for a hop to node x is $l_{max} - x_R$. So a node with resource level 2 in a 5-layer system ($l_{max} = 4$) has resource utilization 2, while the accumulated resource utilization of a path along top level nodes will have resource utilization 0 regardless of its length. Figure 7.7a shows the accumulated resource utilization of the underlay hops necessary for a single overlay hop. Contrary to expectations, RBFM did not have a lower resource utilization that DHash++, although it uses significantly more resource-aware hops as seen in Figure 7.7b. However, the resource levels of underlay nodes used relays for the multi-hop underlay route of a single overlay hop are not considered in the choice of fingers in RBFM. Thus, the physically shortest paths, which have fewer intermediate underlay nodes, provide the best resource utilization. Alternatively, the cluster-aware approaches further decrease these path distances, thus yielding less resource usage. However, the best approach regarding resources remains the resource unaware C-DHash++. This illustrates how underlay routing must play a role in establishing connections in an ad hoc network to best utilize resources.

The average physical distance of single lookup hops shown in Figure 7.3c reflects what we expect: while RBFM:Q90 has a large pool of nodes from which to choose links with strong and physically close nodes, HRM4's nodes have less choice for their links which are drawn from specific (less populated) hierarchical layers. HRM4 actually performs only slightly more location-aware than the completely location-naive Chord - further reducing its usability for mobile network scenarios with ad hoc routing where location-awareness is integral. Multi-tiered hierarchical approaches for MANETs require additional location-awareness such as PIS or PRS. On the other hand, level 2 and 3 nodes in HRM2:01-23 choose links from these two upper levels based only on distance, providing upper nodes with physically close links. Thus, depending on the tolerable lookup failure rate and desired distribution of load between the upper level nodes, HRM2:01-23 and RBFM are best suited for systems where the physical routing distance plays a central role.

An increase in the number of hierarchy layers also results in higher physical distances per lookup hop, as shown in Figure 7.7c. This increase in hop distances is presumably due to the number of nodes in the top two hierarchy layers, which perform a large portion of the routing (see Table 7.3). Indeed, the average physical distance per hop varies only negligibly between the scenarios.



(a) Underlay, static

(b) RBFM, churn

(c) Layer, static and drain

(d) Approach, static and drain

(e) Approach, static and drain

Figure 7.7: **Links** (a) Mean power consumed by underlay nodes for a single overlay hop. (b) Mean resource level per hop of RBFM is nearly twenty times that of Chord and ten times that of DHash++. (c) Mean physical distance and standard deviation per application hop. (d) Mean and standard deviation of resource usage per overlay hop. (e) Mean physical distance per application hop.

7.3 Summary

The majority of results reflect the expectations outlined in Section 7.1 and from the analysis in Section 6.3, with lower lookup failure and higher node lifetimes for RBFM and HRM configurations. Considering only node and network lifetimes together with the percentage of delivered lookups, RBFM and HRM4 clearly outperform the other approaches with over twice the lifetimes compared with resource naive approaches. While the hierarchical approaches all provided the longest lifetimes, HRM4 alone retained a lookup success rate over 65% (at 85%).

Node lifetimes effect not only the stability and the cumulative storage space in the system, but also its ability to implement awareness. When discovering suitable nodes in a non-deterministic fashion, longer lifetimes mean that nodes have more opportunities to find nearby or strong peers. Similarly, any awareness that uses a non-deterministic approach such as piggybacked information may inadvertently improve resource awareness (in the sense of node-lifetime), since longer lived nodes' information is propagated longer, thus increasing their prevalence in other peers' links. Of course, for application lookups which are non-uniformly distributed to nodes, nodes with high lookup rates similarly propagate their information more extensively.

Results also demonstrate that an increase in the number of nodes from which to choose nearby links does not necessarily decrease path lengths, since they are often accompanied by a higher number of necessary (underlay) hops to fulfill a lookup. Moreover, in order to integrate any form of resource awareness on an ad hoc network, the resources along the underlay routes must be taken into consideration. Otherwise, the shortest lookup paths yield the most conservative resource usage.

With regards to the number of hierarchy layers used, three central conjectures can be derived from the results:

- The more hierarchy layers used, the longer lookup path lengths become, both in hops and physical distances. However, more hierarchy layers also facilitate a more fine grained allocation of load to resource levels. In choosing an optimal number or hierarchy layers, the benefits of the varying load-to-level distributions must be waged against the overall increase in load for additional layers.
- The weakest nodes easily dominate system performance. A target minimum lifetime for the weakest nodes could help the system optimization process by decoupling the robustness problem from the maximization of weakest nodes' lifetimes.
- A system-specific optimal threshold for determining which nodes act as leaf nodes and which act as upper layer nodes could also help maximize network lifetime, but has not yet been explored. For example, for the simulated configurations, an increase in the numbers of hierarchy layers resulted in lower maintenance load at stronger nodes and higher load at weak upper layer nodes. However, this likely depends on the allocation of nodes to layers.

Considering this last conjecture, if the threshold is too low (i.e. there are too few leaf nodes), then weak nodes are assigned higher layers and fail earlier; if it is too high (i.e. too many leaf nodes), then strong nodes have fewer nodes from which to choose nearby links and higher load that causes earlier failure. However, the optimal threshold depends on the send and receive drain variables as well as the actual resource values of each resource level and the expected query load. If concrete values for these parameters are known, then, with the help of the expected maintenance and lookup load per node (the lookup load per node can be roughly derived from the fingers' resource levels and the expected query load), the expected number of sent and received messages could be derived to provide expected lifetimes.

Although HRM ultimately provides a robust basis for dynamic mobile (resource restricted) networks with long node lifetimes, it is inferior to the flat RBFM in lookup hop lengths, physical distances, and success rates. However, optimization techniques targeted at multi-tiered hierarchical approaches could cause significant improvements. While optimal thresholds for leaf nodes could improve lookup success rates, routing optimization could reduce the extra hops generated from routing between hierarchy layers and significantly reduce network load, thus further prolonging lifetimes and reducing lookups' physical distances.

Chapter 8

Related Work: Replication

The choice of a structured network in the form of a DHT ensures the necessary scalability, load balancing, and data availability through guaranteed upper bounds on lookup lengths. However, without additional replication, data is neither placed physically near to where it is needed nor is it distributed to stronger more reliable nodes to ensure both data persistence and lower query load on weaker nodes. Thus, data replication is viewed here as more than a backup or faster accesses strategy: it is integral to the resource and location awareness of the DHTs developed in Chapter 5. Replication in peer-to-peer (content distribution or data management) systems focuses heavily on a smaller set of issues than traditional data management systems.

Traditional (distributed) systems are primarily concerned with the consistency of the data, and focus heavily on update propagation, concurrency control, failure detection and recovery, and the handling of read transactions to ensure that the most frequent version of data items is available and being used [LS13]. Dynamic peer-to-peer systems, on the other hand, in which nodes are much less reliable than the servers in a (distributed) data management system with centralized control, much invest significantly more effort to ensure that data is not lost when transient peers leave the system. Moreover, controlling consistency becomes an increasingly difficult task without a central coordinator for the potentially high number of updates, failures, and transactions. In fact, Brewer's CAP theorem [GL02] rules out the simultaneous existence of (strong) consistency, availability, and partition tolerance, where consistency in this context is understood to be a single view of data from any source. However, availability and partition tolerance are two important characteristics of peer-to-peer data management: Data should be found even if nodes have failed or a part of the network nodes have been partitioned from the remainder of the network. In relaxing the consistency requirements, peer-to-peer systems must use at most a form of weak consistency called eventual consistency: if an update is made to a data object and no further updates are performed, then the data will eventually be updated on every node within a given time frame. On the other hand, traditional solutions based on central (transaction) coordination such as locking mechanisms become impracticable with data objects distributed to high numbers of peers with costly and slow communication paths so that approaches must be found that enable distant, transient peers to reliably coordinate joint tasks. Thus, peer-to-peer systems must find more sophisticated solutions for dealing with the issues of replica consistency while also determining in a self-organized fashion how many replicas are needed (depending, for example, on the churn rate), which nodes they should be stored on, and how these copies can be found. However, this work focuses primarily on peer-to-peer replica placement issues instead of concurrency control, and standard concurrency approaches are not further discussed.

8.1 Goals

Many existing systems already employ replication that addresses the general peer-to-peer requirements (i)-(iv) self-organization, scalability, load balancing, and data consistency from Chapter 1, several of which are discussed in Sections 8.2 and 8.3. As with other DHT design issues, these requirements also pose tradeoffs for replication. While, for example, having enough replicas to effectively reduce the lookup path lengths increases system scalability (both load and latency-wise), and replicating popular data objects to multiple nodes can reduce the responsible node's overall load if the replicated objects can be effectively found and thus improve load balancing, both approaches may decrease data consistency if the increased number of replicas are not reliably updated. But as the number of replicas increases, so does the necessary effort to maintain updates. The main peer-to-peer replication issues that existing approaches have addressed in order to pursue these requirements have been discussed in several replication surveys [VLO10, PGVA08, KMP99, DB09] and include (note that the replica characterizations placed in *italics* are from [VLO10]):

Replica quantity. How many replicas are needed to assure a desired availability rate? With system dynamics, too few replicas may cause data to be lost when nodes fail, but too many replicas cause unnecessary storage and update load and may result in outdated data lookups. Should each data object have the same replication rate (*uniform* replication) or should this rate depend on, for example, the object's lookup rate or how important the information is (e.g. *proportional* or *square root* replication)?

Replica location. Which nodes should a data object be replicated to? Replicas that are close in the physical or key space to the object's owner (*predecessor* or *successor* replication), while easier to update, may become simultaneously unavailable if massive failures occur. On the other hand, replicas that are placed along frequent lookup paths for that object (*path* replication) or close to the querying node(s) (*owner* or *client* replication [She10]) must be accompanied by strategies for locating these replicas. Furthermore, these replicas may no longer be beneficial should the query origin shift. Placing replicas randomly, for example with a hash function, (*random* replication) requires no additional overhead to determine suitable replica nodes and may be found at good rates for random distributions of queries over nodes, but fail to adapt to non-uniform query patterns.

Replica maintenance. How often should replicas be initiated or updated? Should the update frequency depend, for example, on when the data object is updated, when conflicts for the data object occur, the number of replicas, the popularity of the data object, or the dynamics of the system? If replicas are updated or maintained on a time schedule *expiration updates* more frequently than the object is queried or changed, then unnecessary traffic increases the total system load. However, if replication or updates are too infrequent, then data may be lost from failure of replicas' nodes and queries may return outdated replicas. On the other hand, if replicas are updated *pessimistically* as soon as the data object is altered, but the replicas are seldom queried, unnecessary load is incurred. However, if replicas are updated *optimistically* only after conflicts have occurred, then nodes may receive outdated objects and sophisticated conflict detection and resolution mechanisms must exist. Further examples include *location-based updates* and *connectivity-based updates* [VLO10, PGVA08].

Query routing. Simply because a data object has been highly replicated does not mean that any replicas will be found during a lookup. In order to be found, and thus enable shorter more efficient lookups and load balancing, lookup mechanisms must be in place which reroute original lookups to replicated copies. However, these mechanisms must ensure that the lookup successful, which is especially challenging in highly dynamic systems where replica nodes are transient. Moreover, the replica lookup mechanism must decide which replica to use, depending on, for example, key or physical distance, load, or reliability.

However, traditional DHT replication approaches consider neither the resource awareness (v) nor the location awareness (xi) requirements. While approaches with a strong focus on load balancing use techniques that might be altered for resource awareness, for example by replacing node load parameters with node resource availability, location awareness must still be integrated. On the other hand, mobile ad hoc network (MANET) replication focuses strongly on location awareness as well as the power availability of its mobile nodes, but is almost exclusively built on unstructured networks. Yet, the DHT approaches developed in Chapter 5 provide resource and location aware structures that can be exploited to provide novel resource and location aware replication mechanisms.

8.2 Unstructured Systems: MANET Replication

Since MANETs are based on ad hoc communication between nodes, with each overlay hop equal to a physical underlay hop, location awareness is an inherent MANET characteristic. Thus, the MANET replication mechanisms are characterized using other categories, such as the following three categories, the first of which is particularly interesting for our scenario [PGVA08, DB09]:

Power-awareness. Is the power availability of replica peers, querying peers, or routing peers considered? Replicas should be placed at strong peers to reduce the replication and query load on weaker nodes. Examples include E-DSM, which considers two types of peers: small mobile hosts (SMH) and large mobile hosts (LMH) and reduces transmission costs by buffering data and transaction messages to be broadcast together [MC04].

Partition-awareness. Are partitions anticipated by the replica mechanism and replication performed accordingly to ensure data availability post-partition? Some such approaches predict when network partitioning will occur by using node movement patterns, for example Wang and Li with their reference velocity group mobility model (RVGM) [WL02], or evaluating link robustness, such as Hauspie et al. [HSC01]. However, MANET partitioning detection is not necessarily applicable to DHT scenarios, since it based on the assumption that all hops are direct physical transmissions between nodes.

Real-time-awareness or scalability. Can information be delivered quickly and before deadlines given by real-time applications? This issue is a reflection of performance with regard to network size, i.e. scalability. However, it is less interesting for DHTs, since they provide guarantees on lookup path complexity.

Further replication mechanisms are categorized as non-location-aware, non-partition-aware, and non-real-time-aware, including REDMAN, which stores replica indexes with hints to replicas' locations on nodes between the data object's owner and the replica peers [BCM05]. Due to the distinctly different character of MANET networks, their replication approaches can be used at most as inspiration for DHT replication. Indeed, their explicit location-based replica positioning and power (i.e. resource) awareness are, besides data redundancy, the main goals of our replication.

Derhab and Badache [DB09] offered a discussion of general evaluation metrics relating to the replication issues from Section 8.1 and awareness-specific evaluation metrics in their survey of MANET replication. The general and power-aware metrics, in contrast with the replication approaches themselves, can be directly applied to the DHT scenario. They found the primary general quantitative evaluation metrics to be the replication cost (e.g. in quantity of replicas or space), update cost (e.g. total messages or hops for updates), query costs (e.g. total hops for queries), data availability (e.g. probability that a random data object can be retrieved within a given time frame), and data consistency (the retrieved data object is the most recent version). Meanwhile, the power-aware metrics were energy balancing, the costs associated with selecting replica peers, the power-availability of replica peers, and whether energy-related failures of replica nodes can be predicted.

8.3 Structured Systems: DHT Replication

Replication in structured systems varies widely and only several replication techniques are discussed here. PAST [DR01] and CFS [DKK$^+$01] are read-only distributed file systems mentioned in Chapter 4 and based on Pastry and Chord, respectively. In PAST, replicas are stored at the k-nearest neighbors in the keyspace, quite similar to the original Chord. This provides physical dispersion of the replicas, thus protecting against massive failures, and assigns roughly the same load to each node. However, it does not consider the popularity of files, so that load balancing (even regardless of resource-awareness) is not increased. A popular or important file has the same number of replicas as a file that is never queried. In CFS, data is saved on a block level, with blocks of data and metadata. These blocks are replicated using both the k-nearest neighbors and *path* replication along lookup search paths. Load balancing is thus improved, since popular blocks are replicated more often along their lookup paths.

Beehive [RS04] is designed to work on many of the original DHTs such as Chord and Pastry. It uses a combination of *predecessor* and *path* replication, replicating to all of a single node's neighbors on a lookup path before replicating to the next node along the lookup path. Thus, all of the owner's neighbors receive replicas, as do all of the neighbors of the last hop of a lookup, etc. The more popular a data object is, the further along these lookup paths the replicas will be places and the more replicas will be created. By replicating to all neighbors instead of just those in the lookup path, this approach reduces the expected lookup hop length for lookups originating at other nodes as well. Nodes decide autonomously using target parameters whether fewer or more replicas are needed and perform these adjustments themselves. Updates are disseminated along the tree structure by each (non end-point) replica node broadcasting to its neighbors. While Beehive provides an interesting replica fanout structure, the owner of a data object has no influence on where and how it will be replicated, which would make the integration of location and resource awareness difficult.

Ghodsi et al. [GAH07] introduce a replication mechanism which divides nodes into groups and then replicates the same data to each of the nodes within a single group, namely all data object held within the group. The groups can, for example, be determined using identifier ranges and the number and size of the groups can be altered to adjust the load and replication degree. While this approach is resource and location naive, it could easily be made aware through a resource and location aware selection of the groups.

Controlled Update Propagation (CUP) [RB03] uses *path* replication, and each node decides autonomously whether to keep a replica of a data object based on possible benefits related to the object's popularity. Popularity is based on the node's query history for the object within a sliding window, and the more popular an object is, the more nodes will decide to store a replica. In order to receive a replica, a node must establish an update channel along the lookup route to the data's owner through which the owner will send updates as soon as they occur. Each node tracks which of its neighbors needs which replicas and thus forwards updates only where they are needed. This approach would be difficult in highly dynamic networks, where nodes are highly transient and links often change, thus breaking the link-based update tree, and the owner has, similar to Beehive, little control over the placement of the replicas.

Chen et al. [CKK02] use another variation of *path* replication, but suitable replica nodes are selected with the help of constraints. Each replica node maintains a dissemination tree of replica nodes further along the lookup path for forwarding updates, thus offering more stability in dynamic scenarios than CUP. In addition to update messages, the owner node periodically sends heartbeat messages just as replica nodes send refresh messages, an perform necessary updates to the dissemination tree. This approach is particularly easy to adapt to resource and location awareness by using replica constraints such as the distance from queries' origins or nodes' resource availability.

Datta et al. [DHA03] suggested a more flexible replica maintenance structure, with each node holding lists of all known replicas as opposed to dissemination trees. These lists can contain nodes from which replicas have been received, nodes on which replicas have been placed, and nodes whose replicas have only been discovered through joint replica update messages. This approach focuses on maintenance instead of placement, with a push phase for disseminating updates and a pull phase for temporarily disconnected or unavailable nodes to receive missed updates. During the push phase of an update, upon receiving the update each node then propagates it to a random number of nodes in its replica list. Update messages include the updated data object, its version, a list of already updated nodes (to which a forwarding node appends itself), and an update round counter.

Tempo [SHD$^+$06] takes a decidedly different approach to replica dissemination, although it also does not determine the nodes at which replicas are to be placed. Assuming a network with highly transient nodes and thus high failure rates, Sit et al.'s goal was to add replicas at a rate at least as high as the rate at which replica nodes fail in order to maintain a given number of replicas and avoid data loss. This runs the risk of swamping the system with replicas that must be stored and maintained, so parameters for each node's bandwidth budget for replication, each node's storage budget for replication, and a global number of replications needed per object (R_{max}) are introduced. They suggest adding new replicas of a data object at a high rate at the beginning and then tapering the rate off once R_{max} replicas have been established. Additionally, each node is only permitted to place further replicas a fixed number of times, and only in accordance with its bandwidth and storage budget.

While each of these approaches was designed for a structured system, note that many are not necessarily dependent on the underlying system being structured. Especially approaches that focus on maintenance such as Datta et al. [DHA03]. BORG (Borg-gOverning Replication objects in larGe environments) [KSHK08], which focuses on highly dynamic massively distributed systems, is yet another maintenance and update approach that is orthogonal to its underlying system. The nodes to which a data object is replicated arrange themselves in a ring, with shortcuts similar to Chord, over which updates are propagated. Thus, a node may belong to multiple rings if it maintains multiple replicas. Each node has a rank which expresses its performance (CPU, bandwidth, and/or data access time) and reliability (average online availability), and the node with highest ranking (i.e. the strongest node) within a replication ring is designated its temporary master. The temporary master is responsible for coordinating the propagation of updates throughout the replication ring and adding new nodes to the replication ring if necessary. This approach is clearly resource aware with its use of node ranks, and could be combined with location aware replica node selection to create resource *and* location aware replication.

8.4 Open Questions

Of the four main issues surrounding peer-to-peer replication (replica quantity, replica location, replication maintenance, and query routing), replication approaches typically focus on only a subset. Thus, for example, a replica maintenance approach can be combined with an approach to replica location and query routing. In this work, the emphasis is placed on three of these four issues which strongly influence availability and resource and location awareness: replica quantity, replica location, and query routing. Replica maintenance approaches such as BORG or that from Datta et al. [DHA03] can be used to supplement the aware placement of replicas with robust maintenance.

The open issues addressed in the following chapters target the highly dynamic nature of the use case scenarios along with the resource and location awareness deficiencies of DHTs. Since the DHT approaches developed in Chapter 5 do provide resource and location aware structures (i.e. links) which can be further exploited to find suitable replica nodes, replication techniques developed specifically for these novel approaches may offer added benefits. The open issues are therefore viewed from the perspective of the developed DHTs:

Replica quantity. How can nodes' varying resource levels be utilized to vary the number of replicas necessary? When replicas are located at nodes with high resource levels, fewer replicas may be necessary to ensure the same level of availability. How does an increase in the number of replicas used per data object effect the resource usages of nodes system-wide? Can fewer replicas result in higher availability, since there is less resource drain caused by replication lookups and management?

Replica location. Replicas should be placed physically near to where they will be needed in order to provide availability should partitioning happen and to reduce lookups' distances to the closest replica. Placing replicas at the locations from which a data object was input into the DHT, from which it is queried, and along which the queries travel can help to improve location awareness. Furthermore, placing replicas at robust nodes may improve availability and reduce storage, query, and routing load on weaker nodes. How can the existing location and resource aware links be used to select suitable replica nodes? How does the use of stronger nodes effect the overall resource usage distribution? What mechanisms can be used to ensure that strong nodes are not overloaded with replication load?

Query routing. How well replicas are found by lookups effects not only the overall load, but also which nodes receive that load. If queries can be routed to both nearby *and* strong nodes, then load on weaker nodes is reduces twofold. How can query routing be designed to prefer strong and nearby replicas? Replicas on nodes with lower resource levels may be less reliable, and routing to them may cause extra overhead if they have failed or are overloaded. Yet, routing to stronger replica nodes may require longer lookup paths. Can a threshold be found for the decision of whether to route queries to nearby or strong replicas?

Chapter 9

Resource and Location Aware Replication

Replication is necessary in any DHT to ensure the availability of data when nodes leave the network, but even more important in highly dynamic networks with limited resource availability. In this case, replication can provide the resource and location aware data placement by copying data to strong nodes and nodes that are close to where they are needed in case partitioning should occur. Indeed, since the approaches in Chapter 5 explicitly avoid resource and proximity identifier selection (RIS and PIS) due to their scalability issues, data replication can provide this "missing" aware placement. Thus, this work's replication's focus on replica quantity, location, and routing must tackle three central goals:

Availability. The set of replicas must ensure a minimum data availability probability ε that any data object can be found at any time. Since nodes have varying resources and failure probabilities, the number of replicas necessary to guarantee availability depends on the strength of the nodes on which they are placed.

Resource Awareness. The use of resources should be minimized. This means not only replicating to strong nodes when possible but also minimizing the total number of replicas and thus the network traffic generated from maintaining these replicas.

Location Awareness. Replicas placed and found physically close to where they are needed can reduce costly cross-network traffic. Furthermore, in the case of physical partitioning in a highly dynamic network, replicas should be found within physical proximity to where they are needed to improve availability. This may include the locations at which data was input into the system and the locations from which the data is queried.

The approaches introduced in Chapter 5 build links that are both resource and location aware. These links offer both efficient discovery of suitable replica nodes as well as one-hop overlay distances to them, so their utilization for replication should incur less organizational load while offering a high level of availability. Moreover, nodes within the higher levels have much higher incoming link degrees than lower level nodes. If these incoming links are aware of the replica placed at the end node, replicas on high level nodes will have a significant pool of routing hints to assist their discovery without significantly adding hint propagation load. Thus, this higher incoming degree can used for designing routing to the replicas.

9.1 Assumptions

In keeping with the DHTs' goals of scalability and self-organization, a decentralized approach to replication is chosen in which the owner of a data object d's key, referred to as $owner(d)$, is responsible for coordinating the number and location of replicas of d. For HRM, this means that leaf nodes also coordinate replication, although they do so with the help of their parent nodes. A global desired availability probability ε must be known to each of the nodes. While this probability could also easily be varied for different data objects, it is assumed here that it is constant. In order to assess how many replicas are necessary to ensure this availability probability, the probabilities of its replica nodes being available must be known. Nodes' availability can be described using a probabilistic model or a time slots model [RDB10]. In the probabilistic model, each node has a known probability of being available after a given period of time (for example in [BLF09]), while in the time slot model, the time periods that a given node is/will be available are deterministically given (for example periodically [SENB09]). We assume that no information is available about when exactly nodes will be available, but rather that available nodes' chances of being available in the future are correlated with their resource levels, and thus use a probabilistic model.

However, nodes within a system rarely know what their future failure probabilities will be in order to estimate the required number of replicas, but the nodes in the proposed systems do know their resource levels. Furthermore, the use of resource levels as node availability levels facilitates the use of the existing resource aware structure. Thus, it is assumed that a node's resource level is correlated with its failure probability. This scenario therefore focuses on a subset of possible resources such as battery power or node lifetimes, although it can be analogously used for other resources to allocate replicas to strong - if not long lived - nodes.

In order to simplify this requirement, it is assumed that each node x with resource level $x_R = \ell$ that is part of the network at a fixed time t_0 has the same probability of being available at time $t > t_0$:

$$p_{t,\ell} := P(x \in A_{t_0,t} | x_R = \ell),$$

where $A_{t_0,t}$ is the set of live nodes at time t. Further simplifying, consider that a set of replica nodes $\Sigma(d)$ on which each data object d is placed is maintained after a fixed interval of time s_3. Thus, the owner responsible for selecting replica nodes need only know the probability of the replica nodes being alive at time $t = t_0 + s_3$. We therefore consider only each node x's failure probability after this fixed time s_0, which depends only on x's resource level:

$$p_\ell := P(x \in A_{t_0,t_0+s_3} | x_R = \ell), \tag{9.1}$$

The probability of node x with not being available after this time interval is $1 - p_{x_R}$. The probability that the set of nodes $\Sigma(d)$ to which data object d has been replicated will have at least one node still available after s_3 is:

$$P(|\Sigma(d) \cap A_{t_0,t_0+s_3}| \geq 1) = 1 - \prod_{x \in \Sigma(d)} (1 - p_{x_R}).$$

If we assume that a massive failure occurs within the s_3 time interval, then the failure probabilities for resource levels according to the example scenario in Chapter 2, Equation (2.3) would be

$$\begin{aligned} p_\ell &= P(x \in A_{t_0,t_0+s_3} | x_R = \ell) \\ &= P(F_x | x_R = \ell) \\ &= \gamma / \left(N \sum_{j=0}^{l_{max}} P(x_R = j)^2 \right) \cdot P(x_R = \ell). \end{aligned}$$

Of course, these probabilities need not preclude a massive failure scenario. Varying battery powers in varying resource levels, for example, provide expected lifetimes for nodes, which could also be interpreted as failure probabilities. Or the Pareto distribution of node lifetimes as in Equation (2.2) could be adapted to reflect the lifetimes, and thus failure probabilities, of the individual resource levels. However, concrete failure probabilities are not assumed and are not further discussed in this work. Hence, it is assumed in the following that the failure probabilities of resource levels are given and that higher resource levels have lower failure probabilities.

9.2 Number and Location of Replicas

Each data object d must guarantee a minimum level of availability, which may be achieved with the help of its owner node $owner(d)$'s links. The node which inputs d into the system is referred to as $input(d)$ and the set of nodes which have placed lookup queries for d is $\Lambda(d)$. Since nodes' failure probabilities depend on their resource levels, the number of replicas necessary to achieve an availability probability $> \varepsilon$ depends on *which* nodes are chosen to replicate. Thus, the number and location of replicas depend directly on one another.

9.2.1 Availability and Resource Awareness

If replica maintenance is performed at time intervals s_3, then the probability that at least one node in $\Sigma(d)$ is still available after s_3 must be $> \varepsilon$.

$$1 - \prod_{x \in \Sigma(d)} (1 - p_{x_R}) > \varepsilon. \tag{9.2}$$

Furthermore, if the set $\Sigma(d)$ is to be chosen from $owner(d)$'s set of links $L(owner(d))$ (i.e. fingers or layer fingers and inter-layer links), then these together must provide sufficient availability:

$$1 - \prod_{x \in L(owner(d))} (1 - p_{x_R}) > \varepsilon.$$

Since top level nodes are considered inexhaustible in the scenarios used in this work, a single top level node would be sufficient to provide absolute availability ($\varepsilon = 1$). This, however, is an unrealistic assumption in the real world, in which no node or set of nodes can actually achieve absolute availability. Nonetheless, top level nodes have lower failure rates.

Fortunately, in RBFM, each node has links to higher level nodes. Figure 6.2 shows that, for the used example scenario, the expected resource level of a finger in a finger interval containing

2^4 nodes is already the second highest resource level $3 = l_{max} - 1$ for stretch $\tilde{c} = 1$. On the other hand, every upper layer node in HRM has a top layer (and thus top level if $x_H = x_R$) inter-layer link. However, HRM leaf nodes have only parent nodes within the upper layers. To overcome this deficit, each leaf node maintains a list of its parent node's current links, which is updated once every replica maintenance interval s_3 by the parent nodes adding its links list to a leaf maintenance message. This parent link list is then used by the leaf node to coordinate replication placement.

So for both RBFM and HRM, each node knows a set of strong nodes with which to fulfill (9.2). As explained below, a link or other node x nearby to the peer $input(d)$ which initiates the put request for d is first inserted into the set of replicas $\Sigma(d)$, after which $owner(d)$ picks a minimum sized subset of its link nodes (or its parents' link nodes) to add to $\Sigma(d)$ which together with x fulfill (9.2). Since nodes with higher resource levels have lower failure probabilities, a greedy selection can be used. Thus, if $\Sigma(d)$ does not yet fulfill (9.2), the highest level link node not yet included in the replication set is added to $\Sigma(d)$ and condition (9.2) is again tested. If multiple link nodes share the same resource level, the node physically closest to $owner(d)$ is chosen. Thus, the number of replicas is minimized while maximizing the resource levels of the used replica nodes. See Algorithm 8 for the initial replication steps a node takes after receiving a put request for a data object.

Algorithm 8 Selecting a set of replica nodes upon an initial put

```
procedure INITIALIZEREPLICATION(data, request)
    if isLeafNode(thisNode) then
        linkSet.input(parentsLinksList)
    else
        linkSet.input(interLayerLinkList)
        → only for HRM
        linkSet.input(fingerList)
    end if
    originReplica = findStrongestLocalNode(linkSet, request.origin)
    linkSet.remove(originReplica)
    if originReplica.undefined then
        originReplica = findStrongestLocalNode(request.hintList, request.origin)
        → request.origin is in request.hintList
    end ifreplicaSet.input(originReplica)
    while !fulfillsAvailabilityRequirement(replicaSet) & linkSet.size > 0 do
        nextNodeList = findStrongestNodes(linkSet)
        nextNode = physicallyClosestNode(nextNodeList, thisNode)
        replicaSet.input(nextNode)
        linkSet.remove(nextNode)
    end while
    replicaLists.input(data, replicaSet, request.origin)
    replicateTo(data, replicaSet)
end procedure
```

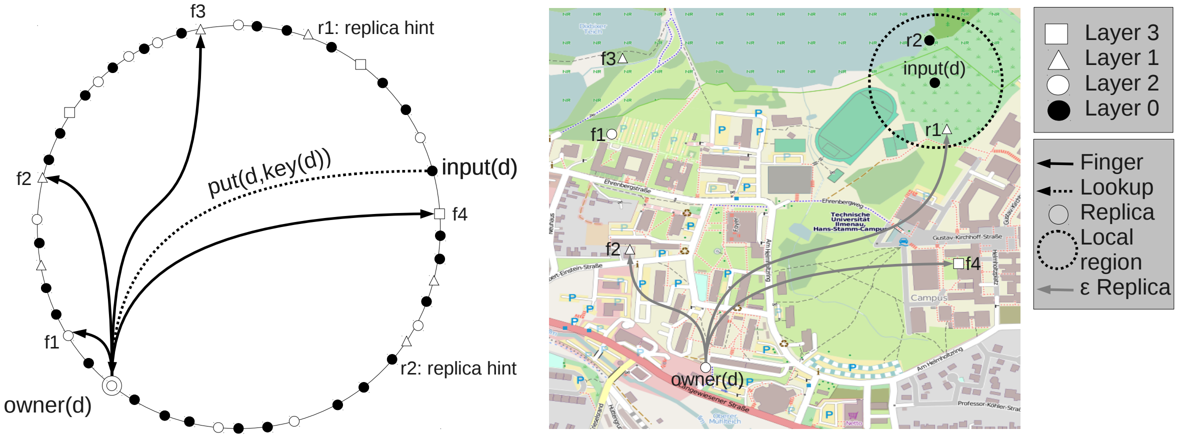


Figure 9.1: Data object d is stored at $owner(d)$ after it receives a put request from $input(d)$. Node $owner(d)$'s fingers and two replica hints from the put request message are shown on the left. The chosen replicas which fulfill the availability probability requirement are shown at the right as ϵ replicas and include the replica placed locally to $input(d)$.

9.2.2 Location Awareness

While the fundamental availability level is reached by nodes replicating to their links (or their parents' links), this provides neither varying numbers of copies depending on a data object's popularity nor would it ensure that data can be found physically close to where it is needed. In order to provide this location awareness and load balancing, a data object d is replicated to within a given radius of the node $input(d)$ that initiated d's put request and each of the lookup nodes in $\Lambda(d)$. To these means, each node x maintains an approximation $r(x)$ of the total physical network radius r by monitoring the node information piggybacked on network messages. Furthermore, a global proximity constant π dictates which fraction of the observed network is considered "local" to a node:

Definition 9.2.1 (Node locality.)**.** *Two nodes u and v are considered* local *by node x, which is referred to as $loc_x(u, v)$, if:*

$$d_{phy}(u, v) \leq r(x)/\pi.$$

In order to establish location awareness, node x places replicas of d locally for $input(d)$ and each node in $\Lambda(d)$, such that:

$$\forall z \in \{input(d)\} \cup \{\Lambda(d)\} : \min_{y \in \Sigma(d)} \{d_{phy}(y, z)\} \leq r(x)/\pi.$$

To facilitate the discovery of suitable replica nodes, each node x which initiates a request (put or get) piggybacks on the request message information about the strongest link that it considers local to itself, i.e. a node v with $loc_x(x, v)$. Furthermore, each hop along the lookup path also appends the message with its node information, resulting in a set hop nodes. Ideally, suitable nodes can be found in $owner(d)$'s links, but these when this is not the case, these *replica hints* within the request messages are used.

Thus, when node $owner(d)$ receives a put request for d, it first identifies if it has a link (or its parent has a link) which it considers local to $input(d)$. If this is the case, it adds the strongest

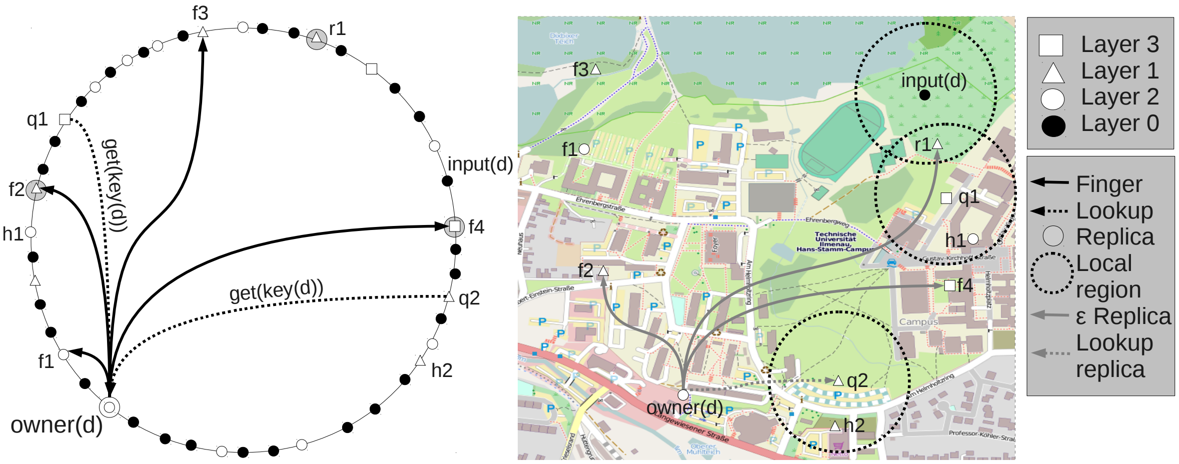


Figure 9.2: Data object d is stored at $owner(d)$ and is already replicated at three nodes. The new replicas are shown on the right, with ϵ replicas denoting the replicas chosen to fulfill the availability probability requirement and lookup replicas denoting the additional replicas chosen locally by the lookup locations.

of those local links to $\Sigma(d)$. If this is not the case, it adds the strongest (i.e. highest level) of the local links found in the put request's hint list, i.e. a node $y \in S$ the set of replica hints such that

$$loc_{owner(d)}(input(d), y) \text{ and } y_R = max\{z_R | \, z \in S \text{ and } loc_{owner(d)}(input(d), z)\}.$$

It then adds a minimum number of additional links as discussed above so that $\Sigma(d)$ fulfills (9.2). This initial replica choice is illustrated in Figure 9.1 for RBFM and an $owner(d)$ node with four fingers and two replica hints $r1$ and $r2$ contained in the put request's message. The strongest of these two hints is chosen for one replica and two further replica nodes are necessary to achieve the desired availability probability. See also Algorithm 8 for this initialization phase.

Replica maintenance, on the other hand, also incorporates the locations of the nodes which have requested lookups for d. Thus, each node x with a replica of d stores a list with each node y which has queried d and another list with a respective strongest local replica hint z per lookup-node y ($loc_x(y, z)$). This information is forwarded to $owner(d)$ along with the lookup load for d at the replica node x before the scheduled replica maintenance in a replica-alive message. Node $owner(d)$ stores a list of all querying nodes $\Lambda(d)$ for each data object d (which expire after a set time period) along with a list of all replica hints for these querying nodes.

When replica maintenance is performed, $owner(d)$ reevaluates the nodes on which replicas have been place. It chooses replica nodes to fulfill (9.2) similarly to the replication initialization, with the exception that the replica hint list also helps to find a suitable replica near $input(d)$ and the list of current replica nodes which are still alive helps find the remaining suitable replica nodes (see Algorithm 9). After establishing these necessary nodes, further replicas are placed local to each lookup initiating node $x \in \Lambda(d)$, if there are not already local replicas in $\Sigma(d)$ ($\nexists v \in \Sigma(d) : loc_{owner(d)}(x, v)$). Node $owner(d)$'s links (or parent's links) are first used to find the strongest local node, but if no suitable replica nodes are found the replica hint list is used. If local nodes are found for each node in $\Lambda(d)$, then the following holds:

$$\forall xin\Lambda(d)\exists y \in \Sigma(d) : loc_{owner(d)}(x, y).$$

Algorithm 9 Selecting a set of replica nodes with replica maintenance

```
procedure MAINTENANCEREPLICATION(data)
    liveReplicas = listsLiveReplicas[data] → list of replica node that are still alive
    queryLocation = listQueryLocations[data] → list of where data has been queried from
    replicaHints = listReplicaHints[data] → list of nodes gathered from lookup requests
linkSet.input(liveReplicas)
    if isLeafNode(thisNode) then
        linkSet.input(parentsLinksList)
    else
        linkSet.input(interLayerLinkList) → only for HRM
        linkSet.input(fingerList)
    end if
    originNode = replicaLists(data).getOrigin
    originReplica = findStrongestLocalNode(linkSet, originNode)
    linkSet.remove(originReplica)
    if originReplica.undefined then
        originReplica = findStrongestLocalNode(replicaHints, request.origin)
    end if
    if originReplica.undefined then
        originReplica = originNode
    end ifreplicaSet.input(originReplica)
    while !fulfillsAvailabilityRequirement(replicaSet) & linkSet.size > 0  do
        nextNodeList = findStrongestNodes(linkSet)
        nextNode = physicallyClosestNode(nextNodeList, thisNode)
        replicaSet.input(nextNode)
        linkSet.remove(nextNode)
    end while
    for queryNode in queryLocations do
        if containsLocalNode(replicaSet, queryNode) == false then
            queryReplica = findStrongestLocalNode(linkSet, queryNode)
            if q thenueryReplica.undefined
                queryReplica = findStrongestLocalNode(replicaHints, queryNode)
                → replicaHints includes query nodes
            end if
            linkSet.input(queryReplica)
        end if
    end for
    replicaLists.update(data, replicaSet)
    replicateTo(data, replicaSet)
end procedure
```

While this procedure may cause replica nodes to change after a given replica maintenance period, this is necessary in order to find the best possible replica nodes while adjusting to higher data popularity. Note that a replica node that remains online is only replaced when a *better* replica node with higher resource level or closer physical distance to $owner(d)$ has been found.

Algorithm 8 shows this basic procedure for selecting a new set of replicas upon replica maintenance while Figure 9.2 illustrates what this might look like. The owner of data object d is shown with its fingers, lookup nodes (shown only for lookup directly to $owner(d)$) and one respective replica hint, and existing replicas on the left. On the right, the newly chosen replicas after replica maintenance are shown. Node $r1$ is within the local region of both $input(d)$ and lookup node $q1$, so no new replica is needed for $q1$. However, $q2$ requires a new replica within its local region and is chosen itself due to its resource level.

9.3 Routing

Nodes with high resource levels are preferred for replication and also have high incoming link degrees. By informing nodes of the replicas held by their fingers and inter-layer links, the information about strong nodes' replicas is thus widely disseminated. To keep such information fresh, each finger (inter-layer link) maintenance request response includes all of the keys located at the respective node. Each finger table (inter-layer link list) contains - in addition to links' keys, addresses, locations, and resource levels - a list of the replicas' keys held at each link.

In addition to this information about fingers' (inter-layer links') replicas, local lists of replicated keys are also maintained. A node x keeps a *local replica list* of all known local replica nodes and their keys, where local is understood as above defined. By including this information on maintenance messages to links (including predecessor and successor links) within a physical distance radius of $2r/\pi$, these lists are propagated on nodes within a given region. To prevent the use of stale replica hints within the local replica list, replica entries are given time-to-live values and deleted after they expire.

Lookup routing for a data object d remains simple. After a node x has determined that it and its direct successors are not the owner of d, it then checks its finger lists, inter-layer link lists, and local replica lists for replicas of d. If one or more replicas are found, the physically closest replica is chosen as the new destination of the lookup. Otherwise, the lookup is forwarded on using the standard routing procedure (for either RBFM or HRM).

9.4 Adaptability

In a highly dynamic scenario, replicas must be regularly maintained - even in the absence of updates - to determine which replica nodes are still alive. Replica nodes which have failed must be replaced in order to maintain the desired level of availability. The replica maintenance suggested here finds not only replacements for failed replica nodes, but also replaces replica nodes as stronger or physically closer nodes are found or replica nodes' resource levels decline. Replica information propagated to finger tables or inter-layer link lists is either refreshed with finger maintenance or is removed once the finger is discovered failed. Similarly, information propagated into local replica lists expires after a set time and is only refreshed when its respective replica node directly refreshes the information to one of its links. This refreshed information is then again propagated throughout the local region with the standard link maintenance messages.

Popular data objects may cause very high load at some or all replica nodes, requiring additional load balancing. The additional replicas placed locally by query nodes provide a rudimentary increase in the number of replicas depending on the data objects popularity. However, to ensure that nodes are nor overwhelmed by lookup load, each replica node maintains a lookup request counter for data object d which it relays to $owner(d)$ to determine if load must be further balanced. For example, if load is above a given threshold, then an additional replica node may be added for the following maintenance period. Alternatively, if a nodes load is above a given threshold it may refuse to replicate data and be removed from $\Sigma(d)$.

9.5 Summary

In this chapter, a replication technique was introduced that uses nodes' existing links to increase the systems' resource and location awareness while reducing the number of replicas which are necessary to achieve a given availability probability ε. In both RBFM and HRM, nodes have links (in expectation) to strong nodes on which replicas are placed to achieve this necessary minimum availability ε. To place additional replicas close to where they are needed, the concept of two nodes being considered *local* to each other is defined as having physical distances within a fraction of the perceived physical radius of the network. A replica for data object d is placed local to the node which initiated its put request (this replica is included in the set of nodes which fulfill the availability requirement) as well as local to every node which has performed a lookup for d. This provides both load balancing for popular data objects and location awareness with respect to where the data is needed.

The following chapter provides a mathematical analysis of the expected number of replica nodes necessary to fulfill the availability probability ε, the expected replication load on each node per resource level, and the probability of finding replicas via the replica hints in nodes' finger lists. Analog to the overlay simulation, similar results would be expected in a simulative evaluation regarding these three measures of primary interest. The only relevant measures not covered in the analysis concern the added location awareness through the locally placed replicas. However, assuming that local replicas are in fact placed near querying nodes and maintained in local replica lists which are propagated within a node's local area, repeated queries are found locally within a single physically close hop, thus fully fulfilling their purpose. This replication technique is therefore assessed purely analytically with respect to the most relevant measures regarding the load which it incurs.

Chapter 10

Replication Analysis

In order to mathematically analyze replication, many assumptions must be made in order to simplify the replication model. A node x's number and location of replicas of data object d depend on, among other things: x's non-deterministic resource and location aware links' resource levels and distances; the location from which d's put request was issued and the location of queries for d; x's non-deterministic network radius estimation $r(x)$; the links of $input(d)$; the nodes of $\Lambda(d)$ which are sent as replica hints; and the hops along the put request and lookup paths which are saved as replica hints. To simplify these very complex dependencies, the following analysis focuses on the *ε-replicas*, i.e. the replicas which are chosen to fulfill the availability requirement and which are primarily placed on owner $owner(d)$'s links. The replicas which are added after these ε-replicas in order to ensure that each node in the querying set $\Lambda(d)$ has a replica that is considered local are referred to as *local-replicas*.

The number and locations of ε-replicas, with the exception of the replica local to $input(d)$, are determined primarily by the structure of the overlay and desired availability probability ε. In contrast, the number and locations of local-replicas depend heavily on the location of the ε-replicas, the query load, the nodes through which lookups are routed, and the local estimation of the network radius. For the sake of simplicity, the local-replicas are ignored in the following analysis, and the necessary ϵ-replicas' distributions are simplified.

Thus, the $input(d)$-local replica is left out of the ϵ-replicas, which are assumed to be placed only on owner $owner(d)$'s strongest links. Moreover, links are assumed to take unbounded replica load, so that replicas are not rejected by overloaded nodes. With this assumptions, we look at the number of replicas which are necessary to achieve the desired availability probability for single data objects (stored under a single key value) and the replication load that is thus incurred by nodes in varying resource levels. However, applications may in fact need entire sets of data objects to be available with a given probability $\hat{\varepsilon} > 0$. Note that this can be achieved analogously by determining the necessary availability probability for each data object ε (and thus the number of necessary replicas) with the help of $\hat{\varepsilon}$ and the size n of the data set: $\varepsilon = \hat{\varepsilon}^{1/n}$. Furthermore, the replica load placed on each node via the ε-replicas is considered, thus providing an estimate for the portion of keyspace that each node is ultimately responsible for.

It is assumed that the local replica lists are corrects and complete and that a lookup thus terminates as soon as it is within the local distance to a replica. This means alternatively that we are only interested in "new" query locations for d, since previous locations route lookups within one hop to the correct local-replica. However, in order to consider how likely the ε-replicas are found by a "new" lookup, further simplification is necessary: We assume that d's replicas are uniformly distributed in the physical and key spaces. Then we can assess the probability that a random node x's finger list will contain a replica of d. Although local-replicas increase the probability of replica discovery, they are ignored in these considerations due to the unknown and variable nature of their numbers, which depend on the frequency and locations of queries as well as the chosen proximity constant π.

10.1 Number of Replicas per Data Object

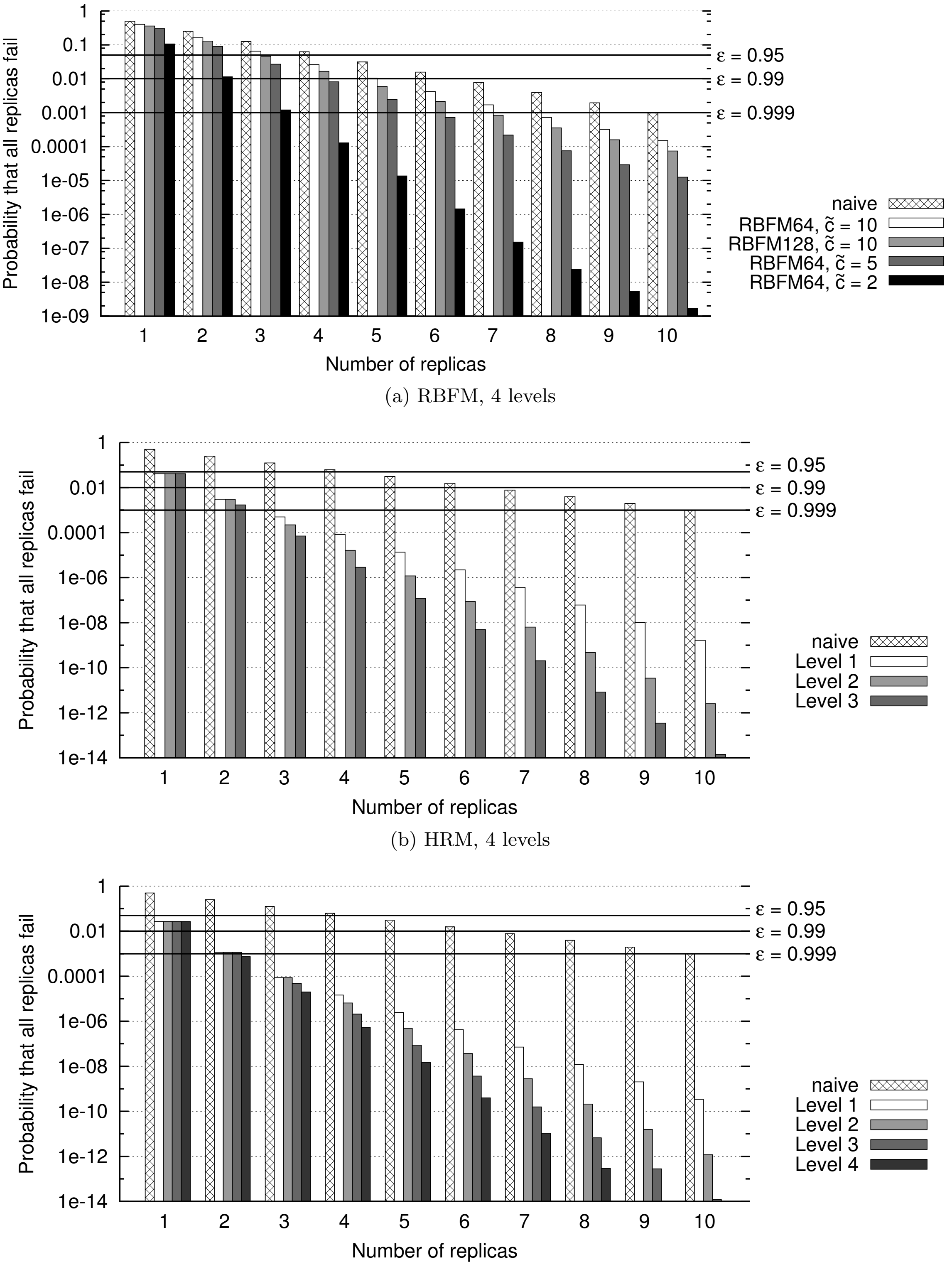
Recall that the owner node places replicas in a greedy fashion on its links with respect to the links' resource levels. In RBFM, fingers from further finger intervals (i.e. from $B_{x,i}$ for i large) have higher expected resource levels. For simplicity sake, we assume that $owner(d)$ adds first the furthest finger $owner(d).F[m]$ to $\Sigma(d)$, and only adds the i$^{\text{th}}$ finger if all further fingers have already been added, i.e. $owner(d).F[j] \in \Sigma(d) \, \forall i < j \leq m$. Of course, in reality, a finger $owner(d).F[i]$ may have a higher resource level than a further finger $owner(d).F[j]$ with $j > i$. In this way we assume that the choice of replica nodes is not greedy. Thus, using fingers' individual failure probabilities from Section 6.1.3 Equation (6.7) and assuming that we have taken the furthest ϕ fingers as replicas, the probability that at least one of these replicas does not fail must be larger than ε:

$$1 - \prod_{i=m-\phi+1}^{m} P(f_i \text{ fails}) = 1 - \prod_{i=m-\phi+1}^{m} \left(\sum_{j=0}^{l_{max}} P(F_x | x_R = j) \cdot P(R_{k_i,min} = j) \right) > \varepsilon. \quad (10.1)$$

Note that f_i and f are used here, as in Chapter 6 as abbreviations for $x.f_i.node$ and $x.f.node$ for either a random or specific node x. Figure 10.1a shows the probability that all nodes replica nodes fail for the scenario used in Section 6.1.3 with variable ϕ. Recall that this scenario used 10,000 nodes in four resource levels, of which 5,000 fail according to a quadratic Zipf distribution. Again, the stretch, which determines the amount of resource vs. location awareness in finger choices, is varied. In order to provide more pessimistic failure probabilities, a maximum number of nodes that have been seen for each finger interval is also used. Thus, RBFM64 assumes that at most 64 nodes are known (i.e. have been "seen" through piggybacked node information) for any finger interval, while RBFM128 assumes a maximum of $128 = 2^7$ and is shown for comparison only. Note that in this scenario, the 32$^{\text{nd}}$ finger interval contains an expected $10,000/2 > 2^11$ nodes. The naive approach shown for comparison is unaware of the resource level differences between nodes so that each node has a failure probability of 0.5. The respective failure probability for all ϕ replicas is thus 0.5^{ϕ}. Three respective values of $\varepsilon = 0.95, 0.99$, and 0.999 are shown in the figure.

Clearly, and as expected, RBFM requires significantly fewer replicas to ensure a fixed availability probability. For a 95% availability, RBFM64 with $\tilde{c} = 5$ (the approach with middle resource awareness of those tested) requires only 3 replicas while the naive approach requires 5. This difference becomes more pronounced as the required availability rises, with RBFM64 needing only 7 replicas for 95% while the naive approach needs 10.

HRM, on the other hand, has very different links than RBFM. We assume here for simplicity that $l_{max} = h_{max}$ and $x_H = x_R$ for all nodes x. Each upper layer node x has inter-layer links



(a) RBFM, 4 levels

(b) HRM, 4 levels

(c) RBFM, 5 levels

Figure 10.1: The necessary number of replicas is determined by the desired availability probability ε.

in every upper layer $\ell \neq x_H$ and layer fingers in layer x_H, and replicates first to the highest layers. So x replicates first to its inter-layer links $x.I[\ell]$ with $\ell > x_H$ and then within its own layer on layer fingers. For layers $\ell > 0$, the probability that ϕ replicas of a single $owner(d)$ node fail depends on the resource level $owner(d)_R$ and must again be smaller than $1 - \varepsilon$:

$$P(\phi \text{ replica nodes fail with } owner(d)_R = \ell) = \begin{cases} \prod_{i=l_{max}-\phi+1}^{l_{max}} P(F_x|x_R = i), & \phi \leq l_{max} - \ell \\ P(F_x|x_R = \ell)^{\phi-(l_{max}-\ell)} \prod_{i=y_R+1}^{l_{max}} P(F_x|x_R = i), & \text{otherwise} \end{cases} \tag{10.2}$$

This probability is derived by simply multiplying the failure probabilities of the higher inter-layer links with the failure probabilities of the layer fingers. Note that we assume that layer 0 nodes use the same replica nodes as their parents, so they have been omitted. Furthermore, the layers $< h_{max}$ are considered for an arbitrary number of fingers although the number of fingers may be restricted by *x.I.closestInt* (we see, however, that few layer fingers are actually needed). Figures 10.1b and 10.1c show analog scenarios to Figure 10.1a for HRM with four and five layers (i.e. levels), respectively. Since all HRM nodes replicate first to the deterministic top level inter-layer link, the failure probability with only one replica is already under 5%. The 5-layer hierarchy provides quicker reduction of the failure probability due to the lower failure probabilities of level 5 nodes and higher number of strong inter-layer links, but both approaches achieve under 0.1% failure probability with only three replicas per data object, regardless of the data owner $owner(d)$. Note that while HRM requires fewer replicas for a fixed availability probability than RBFM or a naive approach, this is due to the high replica load placed on the top layer nodes.

10.2 Portion of Keyspace per Node

The focus here is placed on the portion of the keyspace that each node is responsible for with its replicas. Since keys are uniformly distributed, each node has an expectedc:dhtAnalysis $2^m/N$ keys for which it is the owner. It is assumed that the data's' keys are also uniformly distributed so that each node is owner to the same expected number of data objects. For simplicity, we consider that there is a data object on every key, and consider the expected total number of data objects that a each node maintains as ε-replicas. This depends only a node's resource level (again $x_R = x_H$ for HRM) and can be best expressed as a factor of $2^m/N$, its load without replicas. For example, if node x has replication factor 0.5, then it maintains $0.5 \cdot 2^m/N$ replicas in addition to its own $2^m/N$ keys and its load is 1.5 times more than without replication.

RBFM

For simplicity, assume that a fixed number of replicas are used for each RBFM node according to the availability probabilities in Equation (10.1) and seen in Figure 10.1a. Each node thus sends ϕ_{min} replicas to its furthest fingers with:

$$\phi_{min} = \min \left\{ \phi \middle| 1 - \prod_{i=m-\phi+1}^{m} P(f_i \text{ fails}) > \epsilon \right\}.$$

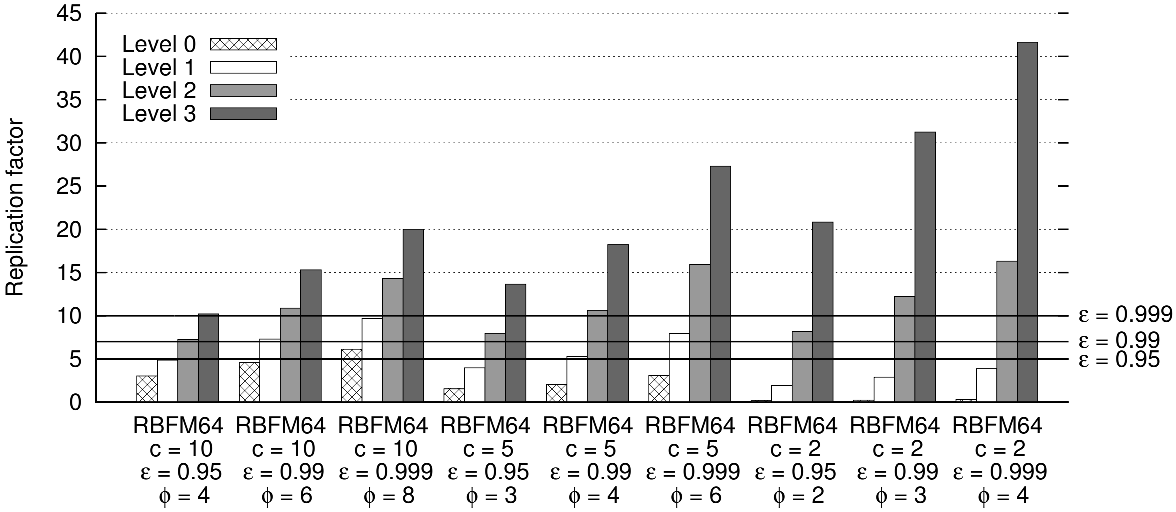


Figure 10.2: The replica factor for 10,000 nodes, 32-bit keyspace, $l_{max} = 3$, $r = 10$, $\rho = 2$, variable stretch $\tilde{c} = 10, 5$, and 2, and variable required availability probabilities. The number of replicas used to achieve this availability varies from $\phi_{min} = 2$ to 8. The replica factor used by every node for a resource naive approach is shown with lines for $\varepsilon = 0.95, 0.99$ and 0.999.

Then the expected number of replicas of a single data object d which are placed in level ℓ can be determined using the probabilities that the owner $owner(d)$'s fingers belong the varying resource levels from Equations 6.1 and 6.7:

$$E(\#\text{replicas on level } \ell) = \sum_{i=m-\phi_{min}+1}^{m} 1 \cdot P(R_{k_i,min} = \ell).$$

Note that each of the 2^m d's has the same expected number of replicas per level in RBFM, regardless of $owner(d)$. Since there are an expected $N \cdot p_\ell$ nodes in level ℓ, each node thus has total expected replica load:

$$\frac{2^m \cdot E(\#\text{replicas on level } \ell)}{N \cdot p_\ell}.$$

Expressed as a replication factor, we have thus:

$$r_\ell = \frac{1}{p_\ell} \sum_{i=m-\phi_{min}+1}^{m} P(R_{k_i,min} = \ell).$$

The respective replication factors for the scenarios in Figure 10.1a are shown in Figure 10.2. Recall that a fixed number of fingers ϕ_{min}, whose values can be read from Figure 10.1a and are noted for each availability requirement ε. The replication factor for a naive approach is simply the number of links necessary to ensure the availability probability and can also be read from Figure 10.1a for this scenario. While top level nodes have up to over four times replication load for the very resource aware approach with $\tilde{c} = 2$, level 0 and 1 nodes have significantly less load even for the mildly resource aware approach with $\tilde{c} = 10$. This demonstrates how well the replication technique alleviates weak nodes of replication responsibilities while highlighting the importance of load distribution mechanisms for possibly overloaded top level nodes.

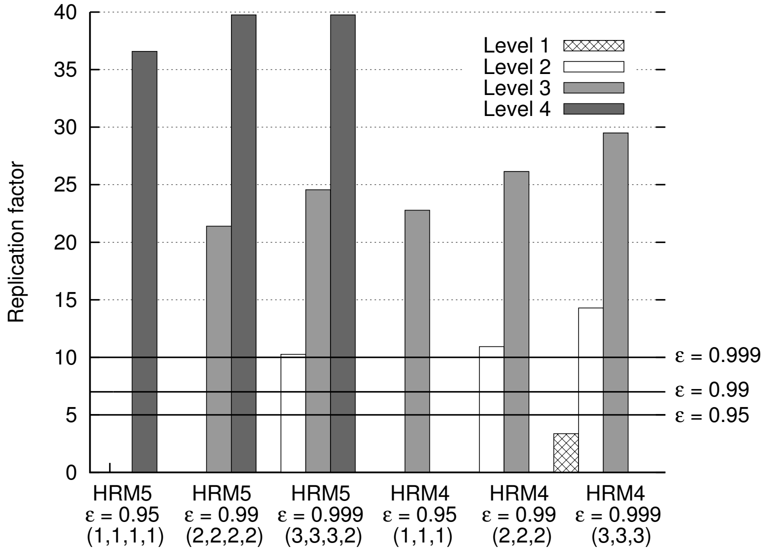


Figure 10.3: The replica factor for $l_{max} = 3$ and 4, 32-bit keyspace, $\rho = 2$, and variable required availability probabilities. The number of replicas used to achieve this availability varies depending on the data owner $owner(d)$'s resource level and ranges from $\phi_{min} = 1$ to 3. The values $(\phi_{min,1}, \phi_{min,2}, \phi_{min,3})$ or $(\phi_{min,1}, \phi_{min,2}, \phi_{min,3}, \phi_{min,4})$ are denoted along with the availability probabilities. The replica factor used by every node for a resource naive approach is shown with lines for $\varepsilon = 0.95, 0.99$ and 0.999.

HRM

In contrast to RBFM, replicas in HRM are placed differently depending on the owner $owner(d)$'s resource level. For simplicity, a fixed number of replicas is assigned for each data object d, although this number $\phi_{min,\ell}$ depends also on the owner $owner(d)$'s resource level in HRM, according to (10.2):

$$\phi_{min,\ell} = \min\{\phi | P(\phi \text{ replica nodes fail with } owner(d)_R = \ell) \leq 1 - \varepsilon\}.$$

Each level distributes replicas in a different fashion, first placing replicas on higher inter-layer links and then on the own layer fingers. However, level 0 nodes use their parent's links in HRM to find replica nodes. Since level 0 nodes are also distributed uniformly through the key space to parent nodes, of the expected $N \cdot p_0$ level 0 nodes, each upper level ℓ has an additional

$$\frac{p_\ell}{1 - p_0}(Np_0)$$

level 0 (i.e. leaf) nodes that analogously place replicas. Thus, there are a total of

$$Np_\ell + \frac{p_\ell}{1 - p_0}(Np_0) = \frac{Np_\ell}{1 - p_0}$$

nodes which place replicas in the fashion of level ℓ nodes, which corresponds to a total number of keys which are replicated in this fashion:

$$m(\ell) = \frac{2^m}{N} \cdot \frac{Np_\ell}{1 - p_0} = \frac{2^m p_\ell}{1 - p_0}.$$

The expected total number of replicas $n_i(\ell)$ which are placed on level ℓ by a level i depends on whether $\ell > i$ or $\ell = i$ and the number of required replicas:

$$n_i(\ell) = \begin{cases} m(i) \cdot 1, & i < \ell \text{ and } \phi_{min,i} > l_{max} - \ell \\ m(i) \cdot (\phi_{min,\ell} - l_{max} + \ell), & i = \ell \text{ and } \phi_{min,\ell} > l_{max} - \ell \\ 0, & \text{otherwise} \end{cases}$$

Note that the conditions $\phi_{min,\ell} > l_{max} - \ell$ refer to the number of replicas which are placed first on stronger links. Then the total number of replicas (i.e.size of keyspace) that are placed on level ℓ nodes is $\sum_{i=1}^{l_{max}} n_i(\ell)$ and the number of replicas placed on a single node in level ℓ is

$$\frac{1}{Np_\ell} \sum_{i=1}^{l_{max}} n_i(\ell).$$

The replication factor for level ℓ can thus be simplified to:

$$r_\ell = \frac{1}{p_\ell(1-p_0)} \sum_{i=1}^{\ell} \tilde{n}_i(\ell) \quad \text{for} \quad \tilde{n}_i(\ell) = \begin{cases} p_i, & i < \ell \text{ and } \phi_{min,i} > l_{max} - \ell \\ p_\ell(\phi_{min,\ell} - l_{max} + \ell), & i = \ell \text{ and } \phi_{min,\ell} > l_{max} - \ell \\ 0, & \text{otherwise.} \end{cases}$$

The replication factors corresponding to the scenarios in Figures 10.1b and 10.1c are shown in Figure 10.3. The numbers of necessary replicas per resource level $\phi_{min,\ell}$ as taken from Figures 10.1b and 10.1c are shown for the 3 and 4 upper levels as tuples $(\phi_{min,1}, \phi_{min,2}, \phi_{min,3})$ and $(\phi_{min,1}, \phi_{min,2}, \phi_{min,3}, \phi_{min,4})$, respectively. In contrast to RBFM, the required number of replicas is low due to the definite replicas to nodes in top levels via the inter-layer links and level 0 nodes are never given replicas. While this results in higher replication factors for the top level nodes, especially for few numbers of replicas, level 1 nodes are largely spared replication load. The difference between the replication factors for top level nodes for $l_{max} = 4$ and 5 is due to the dwindling number of nodes in the top level (i.e. layer) as the overall number of levels is increased. These fewer nodes handle the same set of replicas and thus have more replicas per nodes. HRM underlines the importance of upper load limits for strong but overburdened nodes even more so than RBFM.

10.3 Finding Replicas on Fingers

We now consider for RBFM how probable it is that replicas are found via the replica lists integrated into nodes' finger lists. This loosely reflects the probability of finding a replica along the lookup path of a *new* query node, since a repeated query is fulfilled directly from the querying node's local replica list. Of course, observing only the replicas in nodes' finger lists ignores the possibility of finding a replica in a local replica list along the lookup path, so the actual probability of finding a replica would be higher, albeit difficult to estimate.

So consider a random node $x \notin \Sigma(d) \cup \{owner(d)\}$ along the lookup path for data object d. Recall that x has at most m fingers $x.F[i]$ for $i \in \{1, \ldots, m\}$ that link to nodes $x.F[i].node$, which is denoted simply as f_i or f. The probability that at least one of x's finger nodes f

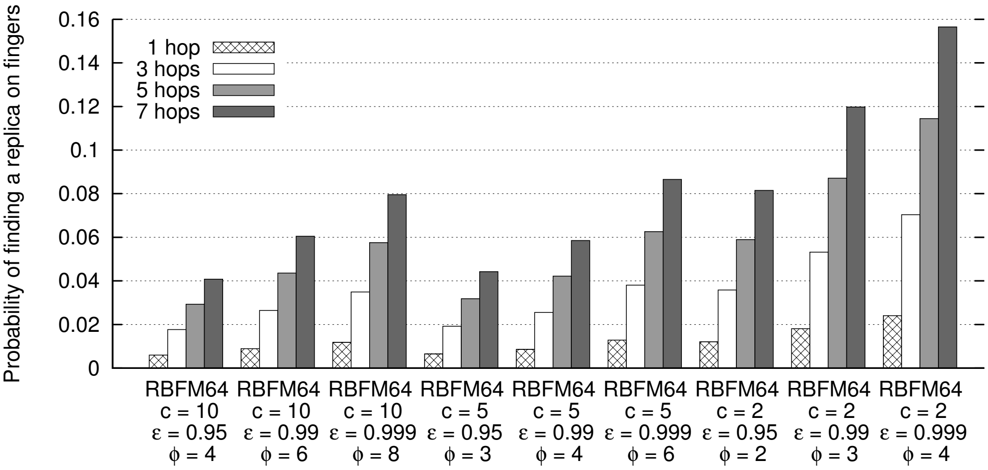


Figure 10.4: Using the same scenario as Figure 10.2, the approximated probabilities that a node along a lookup of variable hop length will have a finger with a replica of the sought data object.

maintains a ε-replica of d is

$$P(\exists f \in \Sigma(d)) = 1 - \prod_{i=1}^{m}(1 - P(\exists f_i \text{ and } f_i \in \Sigma(d))). \tag{10.3}$$

The probability that a single finger node f_i has a replica of d is furthermore:

$$\begin{aligned} P(\exists f_i \text{ and } f_i \in \Sigma(d)) &= P(\exists f_i) P(f_i \in \Sigma(d) | \exists f_i) \\ &= P(\exists f_i) \sum_{\ell=0}^{l_{max}} P(R_{k_i,min} = \ell) P(f_i \in \Sigma(d) |\, R_{k_i,min} = \ell), \end{aligned}$$

where again k_i is the number of nodes in finger interval $B_{x,i}$. In order to estimate th conditional probability $P(f_i \in \Sigma(d) |\, R_{k_i,min} = \ell)$, we make the unrealistic assumption that d's replicas are uniformly distributed in the key space. We furthermore assume that each data object is replicated to a fixed number of nodes ϕ_{min} and use the expected number or nodes in $B_{x,i}$ for $k_i = \lfloor N2^{i-m-1} \rfloor$. Since they are placed on $owner(d)$'s fingers, the expected number of replicas in a given resource level ℓ is thus:

$$\sum_{j=m-\phi_{min}+1}^{m} P(R_{k_j,min} = \ell).$$

The conditional probability is then approximated by the expected number of replicas of d in resource level ℓ divided by the expected number of nodes in resource level ℓ:

$$P(f_i \in \Sigma(d) |\, R_{k_i,min} = \ell) \simeq \frac{\sum_{j=m-\phi_{min}+1}^{m} P(R_{k_j,min} = \ell)}{Np_\ell}.$$

Considering the minimum expected finger interval $\lceil m + 1 - log(N) \rceil$ which actually contains nodes, we obtain thus the following approximation for (10.3):

$$P(\exists f \in \Sigma(d)) \simeq$$

$$1 - \prod_{i=\lceil m+1-log(N)\rceil}^{m} \left[1 - \sum_{\ell=0}^{l_{max}} \left(P(R_{k_i,min} = \ell) \frac{\sum_{j=m-\phi_{min}+1}^{m} P(R_{k_j,min} = \ell)}{Np_\ell} \right) \right] \quad (10.4)$$

Furthermore, if the nodes along a lookup path of length *hops* are considered independent of one another, then the probability that at least one replica will be found on at least one of the path's nodes' fingers is:

$$1 - (1 - P(\exists f \in \Sigma(d)))^{hops}.$$

Figure 10.4 demonstrates this approximated probability of finding a replica for varying lookup hop lengths $hops = 1, 3, 5$, and 7 for the scenarios in Figure 10.2, where one hop corresponds to the simple case from (e:repanalfindingrep). As expected, the probability of finding a replica increases for the number of ε-replicas used and the lookup hop length, but note how it also increases as resource awareness increases (i.e. $\tilde{c}$ decreases). However, these probabilities are rather small and illustrate the importance of the additional local-replicas and the propagation of replica information to nearby nodes, which should significantly increase the replica hit rate.

10.4 Summary

This analysis of the replication techniques using simplified models and assumptions demonstrates how many replicas might be expected to satisfy the availability requirements, how much replication load nodes in the varying resource levels are assigned for these replicas, and how probable these replicas are found during simple lookups. While RBFM tends to require more replicas than HRM since its nodes' links have non-deterministic resource levels that are not guaranteed strong, the replication factor on its strongest nodes tends to be lower due to the participation of nodes from all levels in storing replicas. As the stretch is adjusted to increase resource awareness in RBFM, fewer replicas are necessary and more replica load is transfered from weak nodes to strong nodes. HRM, on the other hand, consistently has very high replication factors on its strongest nodes, with between 20 to 40 times the normal load in the given scenarios. This is the result of each data object being first replicated to level l_{max} and the numerous nodes in level 0 not being used at all for replica storage. HRM is thus, as for the overlay comparisons, better suited to networks in which the lowest level nodes have very low capabilities and should be spared possible load.

These load-oriented results' primary contribution - other than verifying that the replication technique's resource awareness - was to illustrate the importance of mechanisms which enable nodes to cap their total replica load in order to prevent strong nodes from becoming overloaded. Such mechanisms distribute as much replica load as possible to the highest levels, in which replicas are also more easily found, and spread the excess load downward through the levels. On the other hand, the ability for new lookup requests to find replica nodes is limited when only considering the ε-replicas and the replica hints found in nodes' finger lists, with the used scenarios showing between 2% and 12% successful replica discovery for lookup paths with 5 hops. This emphasizes the importance of both local-replicas, which increase the total number of replicas and thus significantly improve the chances of finding replicas, and further replica information propagation, for example through local replica lists.

Chapter 11

Conclusion

The focus of this work lay in the resource-based groundwork established in Chapters 2 and 3; the novel resource and location aware DHTs presented in Chapter 5; the DHT comparison results with regard to the number of hierarchy layers and the tradeoffs between resource and location awareness in Chapters 6 and 7; and the resource and location aware replication from Chapter 9. More specifically, the main contributions are:

Awareness Framework. Both resource and location usage taxonomies are introduced to provide a basis for describing existing work, comparing varying approaches, and defining relevant new work. The importance but lack of popular evaluation metrics that are used on a resource-level-basis for assessing resource awareness is established.

Novel resource and location aware DHTs. These approaches use both resource and location awareness while fulfilling the remaining use case requirements. Both flat and hierarchical structures are used so that differences caused by network structure might be assessed. A cluster-based approach is developed to compare and improve underlay routing.

DHT Comparisons. The simulative evaluation reflects the analysis results, showing how the developed DHTs have up to over twice the node lifetimes for weak nodes while experiencing success rates over twice as high as the resource and location unaware alternatives. The most interesting evaluative results include:

- The resource-level-based measures demonstrate that various approaches prefer different levels for routing responsibilities, showing how important these trends towards levels might be if nodes' resource level distributions and strengths are known.
- Surprisingly, resource and location awareness do not always behave as tradeoffs, and can even indirectly improve each other.
- Underlays require an underlay-aware approach more sophisticated than latency, as overlay improvements are otherwise unsatisfactorily reflected in the used cluster-based underlay.
- An increase in hierarchy layers increases the necessary hop count but can reduce the load for nodes in upper layers. On the other hand, a flat (i.e. one layer) system increases the lookup routing success through its simplified structure, but has lower lifetimes than a hierarchical approach with leaf nodes.

Replication. A replication technique that improves resource and location awareness by using existing aware links is introduced. This form of replication presents a work-around to the poorly scalable RIS and PIS approaches. Its lower overall load and added resource awareness is demonstrated in a mathematical analysis.

These contributions provide the foundation for further research in the area of awareness in dynamic distributed data management systems by classifying the differences between approaches, performing comparisons of possible solutions, and deriving initial conclusions about tradeoffs and structural decisions. After recapping this work's development in Section 11.1, the open questions from Chapters 4 and 8 which were targeted in this work are discussed in Section 11.2, and an excerpt of possible future work encouraged by these results is presented in Section 11.3.

11.1 Summary

In order to address resource and location aware, robust decentralized data management, much preliminary work was devoted to understanding the dimensions of the given problem and how previous approaches have dealt with similar problems. In Chapter 1, various use case scenarios were described in which data management systems on peer-to-peer networks require both nodes' varying resources and their locations to be considered in order to provide robustness, i.e. high data availability. These scenarios included systems built on both intact infrastructures such as the heterogeneous cloud scenario as well as ad hoc networks such as the disaster scenario. However, these underlaying networks have inherently different requirements considering resource and location awareness. The focus of this work was devoted to the intact infrastructure case in which nodes can be viewed independently of their underlay, but with an interest in the applicability of these systems being used on ad hoc networks. Requirements were derived from the use case scenarios and stemming from what is considered to be *robustness* in decentralized data management. Of these requirements, self-organization and scalability were viewed as prerequisites for all design issues, while the remaining requirements may vary for different design issues.

After establishing the scenarios and requirements on which this work is founded, Chapter 2 establishes a working definition of two of the central requirements: resource and location awareness. In order to differentiate between various approaches to and implementations of resource and location awareness, taxonomies are established for each to define their use and integration. With the help of these taxonomies, the use of resources and location in this work are then specified. Foundational assumptions such as the distribution of resources to nodes are also discussed.

With an established approach to resources and location, Chapter 3 considers how peer-to-peer systems can meet the derived requirements. General peer-to-peer challenges are discussed and placed in relation to the specific use case scenario requirements. Furthermore, the use case scenarios are classified, where possible, with regard to a popular peer-to-peer application taxonomy, thus narrowing the possible solution design space. Of the design decisions, unstructured systems were ruled out so that the focus is turned to structured systems, which are almost exclusively distributed hash tables. After an introduction to DHTs, their possible design issues are discussed in relation to resource and location awareness. Going further, possible tradeoffs for the derived requirements are considered for each of the DHT design issues. While most of the DHT design issues are addressed together in the following chapters, replication is addressed in separate chapters. The derived requirements and the design issues that were used in this work are listed in Table 11.1.

Integration	Requirement	Addressed by design issues
Prerequisite for all design issues	(i) self-organization	all
	(ii) scalability	all
Overall system goals, vary for design issues	(iii) load balancing	links, routing (HRM), maintenance, replication
	(iv) data consistency	replication
	(v) resource awareness	links, routing, maintenance, replication
	(vi) location awareness	links, routing, maintenance, replication

Table 11.1: Requirements met primarily through links, routing, maintenance, and replication.

Concrete related work and applications for DHTs are presented and characterized with respect to the introduced resource taxonomy and resource classifications RIS, RNS, RRS, and RSD, among other things, in Chapter 4. Various approaches to resource and location awareness in DHTs are discussed and an extended peer-to-peer structural taxonomy is suggested for differentiating between the structural approaches to resource awareness. MANETs in particular were discussed as the forerunners of location awareness, although they rarely also employ resource awareness. Finally, the recurring measures applied in the evaluation of the numerous mentioned approaches are identified and mapped to peer-to-peer challenges which they aim to assess, thus establishing an evaluative foundation. Of the countless open questions regarding awareness in DHTs, a focus is then set on structural aspects, in particular how well hierarchal structures can accommodate the requirements derived from the use case scenarios.

In Chapter 5, several novel resource and location aware DHTs are introduced. The two main DHTs, RBFM and HRM, use a flat and a multi-tiered hierarchical structure, respectively. While RBFM is a rather simple extension of the existing DHash++ and can easily be applied to almost any DHT, HRM is specifically designed for the use case scenarios and bears less resemblance to existing systems. Although they use distinctly different links and routing, both maintain the standard (expected) lookup hop length of $O(\log N)$. Two adaptations of these approaches are also described, with $C-DHash++$ and $C-RBFM$ making use of cluster-head information in a cluster-based ad hoc routing scenario and a hybrid approach combining RBFM and HRM to a two-tiered hierarchical approach.

The following analysis of RBFM and HRM in Chapter 6 starts by summarizing the measures that are used in both the mathematical analysis and simulative evaluation. Resources, distances, maintenance and failures are then analyzed on a per-link basis, since the analysis of complete lookup hops is impracticable due to the dependencies between single hops in a lookup. Each of these measures yields a probability or expected value that depends on many different variables, including the distribution of nodes in space and the distribution of resources to nodes. Thus, for illustration, the measures are calculated and shown for a simple, fixed scenario. These results provided expectations for the following simulative evaluation in Chapter 7.

The evaluation used a combination of the static, churn, drain, and massive failure behavior scenarios for four different comparisons. These comparisons observed RBFM with varying parameters to DHash++ and Chord; RBFM and the cluster-based approaches C-DHash++ and C-RBFM on a cluster-based ad hoc underlay; HRM, the hybrid DHT, and two-tier hierarchies against the flat RBFM; and varying numbers of hierarchy layers in HRM. Using varying simulation setups, the expected behavior was tested for a slew of measures which were often meaningless on their own. The node lifetime and lookup failure rates were particularly interesting, since they directly reflect the system's robustness and capability to deal with nodes with varying resource strengths. The results show that while HRM could more successfully incorporate nodes' resource availability differences and thus provide long lifetimes while sustaining an acceptable lookup failure rate, RBFM more easily adapted to network changes and thus provided lower lookup failure rates and distances.

Although these DHTs proved promising for highly dynamic scenarios, they can only provide robust data availability with the help of replication. The goals of and existing approaches to replication in peer-to-peer systems are outlined in Chapter 8. The focus of this work is narrowed to replica quantity, location, and routing, with the goal of utilizing the existing structure of RBFM and HRM's links to placed links physically near to where they are needed in a fashion that assures their availability. Solutions for unstructured MANET systems are briefly summarized, as they have provided a large amount of resource aware replication strategies. Unfortunately, they are largely unsuitable to structured scenarios, while DHT replication strategies, on the other hand, have typically ignored nodes' varying resources.

A replica strategy geared specifically towards RBFM and HRM is then described in Chapter 9. In order to ensure a set availability probability of data, the failure probability of nodes must be known within given time periods. And in order to effectively use the resource and location aware link structure of RBFM and HRM, the use case scenario is restricted to resources that correlate with failure probabilities. The number of replicas thus depends on the resource levels of their replica nodes while the position of the replica nodes depend on the data owner's, inputting node's, and querying nodes' locations. This replication strategy adds an element of proximity data placement without searching for new replica nodes by utilizing the existing links which prefer both strong and nearby nodes.

Chapter 10 considers the introduced replica strategy with a mathematical analysis. Using simplified conditions, the number of replicas necessary to fulfill the required availability probability, the replica load placed on varying resource levels, and the probability of finding a replica on a random lookup are discussed. In order to illustrate the meaning of the resulting approximations, the measures are calculated for the same fixed scenario as in the DHT analysis in Chapter 6. The results demonstrate the reduced number of required replicas for the resource aware replica placement and the importance of using upper bounds on replica load for top level nodes.

11.2 Addressed Questions

Many of the open questions discussed in Chapters 4 and 8 have been addressed throughout this work, while others were not in this work's scope. While the replication specific questions were addressed through the replica design in Chapter 9, the more general questions were addressed throughout the DHT analysis and evaluation. The following three questions in particular were addressed.

How do resource and location awareness effect each other? The use of prospective links lists collected via information piggybacked on messages caused stronger nodes to be preferred in a churn scenario even for the resource unaware DHash++. This indicates that such gathered information about nodes may unintentionally cause resource awareness when high resource nodes are more active or live longer, thus propagating more of their own node information throughout the network. On the other hand, resource awareness on the overlay level can actually have an adverse effect on total resource awareness in ad hoc networks if the resources are not also considered on the underlay level, since longer paths may then be chosen which route through more (low level) nodes. In order to facilitate total resource awareness in an ad hoc network, it is necessary that the underlay hops' locations are integrated in the overlay design. And interestingly, resource awareness can also improve location awareness in churn scenarios where nodes are highly dynamic. When links are established to both close and stable nodes (as opposed to simply the nearest node), then those links are longer lived so that fewer reconfiguration phases are necessary in which new suitable nodes must be found. Thus, we have seen for multiple situations that resource and location awareness can actually positively effect each other instead of posing a tradeoff.

How do the peer-to-peer structural taxonomy variations influence resource and location awareness? This work concentrated primarily on the topic of flat vs. hierarchical structure, as opposed to single vs. multiple or vertical vs. horizontal overlays. A horizontal approach was chosen to increase robustness by removing the bottleneck gateway peers, and the single overlay used could easily be replaced by a horizontal multi-overlay system. The hierarchical structure in question did in fact offer longer node lifetimes due to the finer differentiation between nodes' resource levels. However, in comparison to the flat RBFM, the hierarchical HRM also led to higher hop counts per lookup due to the passing of messages between layers; longer physical link and lookup distances due to the smaller set of nodes from which to choose close links within a single hierarchy layer; and higher lookup failures due to links being broken. Furthermore, node lifetimes and location awareness decreased as hierarchy layers increased, suggesting that lower numbers of hierarchy layers are able to provide better awareness. The number of hierarchy layers thus effects not only the distribution of load to the individual resource levels but also the degree of possible resource or location awareness.

Which structure is best for the derived requirements? While the multi-tiered hierarchical approach HRM provided less location awareness, thus making it less suitable for ad hoc networks, it is more capable of relieving leaf nodes of load. Thus, leaf nodes and with them the entire network, have longer lifetimes. RBFM, on the other hand, can be configured to incorporate more or less resource or location awareness and can easier adapt to a changing network. With its flat structure, nodes that change levels do not change their outgoing links or responsibilities, which led to fewer lookup failures in simulation. Depending on the differing strengths of the nodes, a flat or hierarchical structure may be better.

11.3 Future Work

The two-tier approaches performed well on many measures, especially location awareness, but failed to provide the robustness required for a system with high churn rates. The improvement of these approaches' robustness could provide promising resource and location aware alternatives. Similarly, the multi-tier hierarchy suffers primarily from the extra hop counts necessary to traverse hierarchy layers. If a routing workaround to this problem that does not require extra hops for additional hierarchy layers can be found, then HRM would have an even larger resource awareness advantage.

The approach and layer comparisons showed how the choice of a threshold with which weak nodes are allocated to either a leaf or upper layers in HRM affects the distribution of network load and ultimately the nodes' and network's lifetime. An optimal threshold would maximize node lifetimes and thus increase robustness. While previous work from Zoels et al. has addressed similar questions, it focused on either balancing load between *super-peers* [ZDK07] or minimizing the total network traffic by finding an optimal ratio of leaf to super-peer nodes [ZHDK09]. However, results presented here demonstrated that when nodes are drained by activities, then lifetimes are determined not by the total network load but rather by the load incurred by the network's weakest nodes. This could be addressed by either finding optimal thresholds to maximize network lifetime, or using minimum target lifetimes for weakest nodes to further optimize the hierarchical system. In the second case, optimal thresholds between the remaining hierarchy layers could also be sought for more sophisticated scenarios; for example, scenarios that place upper bounds on the permissible load per time unit for varying resource levels.

An optimal configuration of the system could be explored for the cases that nodes' replication distributions are either completely known or unknown. For an example of a system in which distributions are completely known, consider a cloud use case scenario like that from Section 1.2.4 in which nodes are not restricted by their battery power but rather classified by their computing power, energy consumption, and network connections - such as Amazon's Dynamo system with DHT-based key-value storage in a cloud [DHJ^{+}07]. On the other hand, consider a standard unregulated peer-to-peer system of heterogeneous nodes for an example of a system with unknown replication distribution. Recall that this work assumed a Zipf distribution for resource availabilities, but does not specifically configure the system to work optimally for this distribution. However, for systems in which the resource distribution is known, optimal values for the number of resource levels, allocation of nodes to resource levels, number of hierarchy layers, various maintenance parameters, numbers and ranges of fingers, and the concrete number of replicas might be found. And when there is no distribution that can be assumed for the system, approaches for determining good configurations without the help of a priori information must be found.

Finding an optimal configuration is also a necessary step when considering how to best adapt the approaches introduced here to specific cloud and sensor-net use cases. For the cloud scenario without restricted battery power (as mentioned above), much information may be available about the locations and strengths of systems nodes. However, weak nodes with unbounded power supplies are restricted by their computing power instead of the total number of messages they have sent and received. So the suggested approaches must be adapted to specifically incorporate node computing load as its resource as well as the known distributions of resources and node failure probabilities. Sensor-nets, on the other hand, have both restricted battery and computing power and operate on ad hoc networks with a primarily static structure. The adaptation to this scenario should thus requires stricter underlay-based location awareness, which could be achieved through specialized routing and replication techniques.

Currently unexplored scenarios could also benefit from the interpretation of nodes' suitability to perform specific tasks as resources. In this case, the DHT overlays could also be used to distribute (computational) tasks to nodes with varying capabilities to perform them, for example via the resource and location aware links. Specific use cases would have to be identified, the resource notation and usage adapted to incorporate multiple dimensions which reflect nodes' capabilities to perform tasks, and techniques for distributing tasks developed.

Multiple resources such as sensor-nets' battery and computing power may also benefit from a more sophisticated use of replication notation (R.5). Tuples that store values for multiple resources could, for example, enable nodes to prefer or weight other peers' resources differently depending on their needs, whole systems to prefer varying resources for different times or scenarios, or each node to communicate which resource is being over or under-loaded. Functions, on the other hand, could provide information about the fluctuation in a node's resources. For example, a node that moves in a periodic fashion may have bandwidth availability that depends on its location, which is given by a function over time. Such resource functions could be used in a fashion similar to path predictions in delay-tolerant networks, with peers predicting when a specific node will have strong resources. This is relevant not only for establishing and routing over links, but also for determining the number and location of replicas necessary to provide availability guarantees.

Thus, the potential for future work extends in varying directions, including the search for more efficient hierarchical routing, an optimal allocation of nodes to hierarchy layers, and the examination of the effects of different resource distributions on the system. While a Zipf distribution with arbitrary power was assumed for the resource distribution in this work (quadratic Zipf distributions were used when a concrete distribution was necessary), this distribution may also be formally known or completely unknown. This relates closely to the possible future application of the presented approaches in specific cloud and sensor-net scenarios, in which resource distributions may in fact be known. Furthermore, an examination of the possible use of more sophisticated tuple and function notations for resource awareness could provide more variable resource awareness for multi-dimensional or time-dependent resources.

Bibliography

[Abe01] K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems*, volume 2172 of *Lecture Notes in Computer Science*, pages 179–194. Springer Berlin Heidelberg, 2001.

[Abe02] K. Aberer. Scalable Data Access in P2P Systems Using Unbalanced Search Trees. In *Workshop on Distributed Data and Structures (WDAS'02)*, 2002.

[ACMD$^+$03] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: A Self-organizing Structured P2P System. *SIGMOD Record*, 32(3):29–33, 2003.

[ACMHP04] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. V. Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In *ISWC'04*, pages 107–121, 2004.

[AH00] E. Adar and B. A. Huberman. Free Riding on Gnutella. Technical report, Xerox PARC, 9 September 2000. `http://www.firstmonday.dk/issues/issue5_10/adar/index.html`.

[AKU03] A. Anagnostopoulos, A. Kirsch, and E. Upfal. Stability and efficiency of a random local load balancing protocol. In *In Proceedings FOCS*, pages 472–481, 2003.

[ALAS05] M. Artigas, P. Lopez, J. P. Ahullo, and A. Skarmeta. Cyclone: A novel design schema for hierarchical dhts. In *P2P'05*, pages 49–56, 2005.

[ALS07] M. S. Artigas, P. G. Lopez, and A. F. Skarmeta. A comparative study of hierarchical dht systems. In *Proceedings of the 32nd IEEE Conference on Local Computer Networks*, pages 325–333, 2007.

[ARK$^+$05] F. Araujo, L. Rodrigues, J. Kaiser, C. Liu, and C. Mitidieri. Chr: a distributed hash table for wireless ad hoc networks. In *ICDCS Workshops '05*, pages 407 – 413, 2005.

[ATS04] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, December 2004.

[BAS04] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *SIGCOMM '04*, pages 353–366, 2004.

[BBB09] M. Bouhorma, H. Bentaouit, and A. Boudhir. Performance comparison of ad-hoc routing protocols aodv and dsr. In *Multimedia Computing and Systems, 2009. ICMCS'09. International Conference on*, pages 511–514. IEEE, 2009.

[BBKK10] M. Bienkowski, A. Brinkmann, M. Klonowski, and M. Korzeniowski. Skewccc+: a heterogeneous distributed hash table. In *Proceedings of the 14th international conference on Principles of distributed systems*, OPODIS'10, pages 219–234, Berlin, Heidelberg, 2010. Springer-Verlag.

[BBST01] C. Batten, K. Barr, A. Saraf, and S. Trepetin. pstore: A secure peer-to-peer backup system. *Unpublished report, MIT Laboratory for Computer Science*, pages 130–139, 2001.

[BCM05] P. Bellavista, A. Corradi, and E. Magistretti. Comparing and evaluating lightweight solutions for replica dissemination and retrieval in dense manets. In *Computers and Communications, 2005. ISCC 2005. Proceedings. 10th IEEE Symposium on*, pages 43–50. IEEE, 2005.

[BGK$^+$02] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Fifth Int. Workshop on the Web and Databases (WebDB'02)*, pages 89–94, 2002.

[Bit13] Bittorrent, 2013. `http://www.bittorrent.com`.

[BK09] E. H. T. B. Brands and G. Karagiannis. Taxonomy of p2p applications. In *GLOBECOM Workshops, 2009 IEEE*, pages 1–8, 2009.

[BLF09] S. Bernard and F. Le Fessant. Optimizing peer-to-peer backup using lifetime estimations. In *Proceedings of the 2009 EDBT/ICDT Workshops*, pages 26–33. ACM, 2009.

[BLJ05] D. A. Bryan, B. B. Lowekamp, and C. Jennings. Sosimple: A serverless, standards-based, p2p sip communication system. In *Advanced Architectures and Algorithms for Internet Delivery and Applications, 2005. AAA-IDEA 2005. First International Workshop on*, pages 42–49. IEEE, 2005.

[BQ04] F. Bustamante and Y. Qiao. Friendships that last: Peer lifespan and its role in p2p protocols. In F. Douglis and B. Davison, editors, *Web Content Caching and Distribution*, pages 233–246. Springer Netherlands, 2004.

[BRB10] M. Brückner and L. Ribe-Baumann. Decentralized, resource-aware information management and delay tolerant networks in command-and-control. In *GI Jahrestagung (2)*, pages 175–180, 2010.

[BS06] S. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–11, 2006.

[BS08] S. A. Baset and H. Schulzrinne. Openvoip: An open peer-to-peer voip and im system. *Proc. of SIGCOMM (demo)*, 2008.

[Btd13] BTDigg DHT search engine, 2013. `http://btdigg.org`.

[CBB$^+$03] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. B. Zdonik. Scalable distributed stream processing. In *CIDR*, volume 3, pages 257–268, 2003.

[CCN$^+$06] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by dhts. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 351–362, New York, NY, USA, 2006. ACM.

[CCR05] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 85–98, Berkeley, CA, USA, 2005. USENIX Association.

[CCRK04] M. Costa, M. Castro, A. Rowstron, and P. Key. Pic: practical internet coordinates for distance estimation. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 178–187, 2004.

[CDCR02] M. Castro, P. Druschel, Y. Charlie, and H. A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical Report MSR-TR-2002-82, Microsoft Research, 2002.

[CDHR02] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting network proximity in distributed hash tables. Technical report, Microsoft Research, 2002.

[CDK$^+$03] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *ACM SIGOPS Operating Systems Review*, number 5, pages 298–313. ACM, 2003.

[CDKR02] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[CF05a] C. Cramer and T. Fuhrmann. Isprp: a message-efficient protocol for initializing structured p2p networks. In *Performance, Computing, and Communications Conference, 2005. IPCCC 2005. 24th IEEE International*, pages 365 – 370, april 2005.

[CF05b] C. Cramer and T. Fuhrmann. Proximity neighbor selection for a dht in wireless multi-hop networks. In *P2P '05*, pages 3 – 10, 2005.

[Chi13] Chimera structured overlay network, 2013. `http://current.cs.ucsb.edu/projects/chimera`.

[CKK02] Y. Chen, R. H. Katz, and J. D. Kubiatowicz. Dynamic replica placement for scalable content delivery. In *Peer-to-Peer Systems*, pages 306–318. Springer, 2002.

[CMN02] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. *ACM SIGOPS Operating Systems Review*, 36(SI):285–298, 2002.

[Cod13] Codeen, 2013. `http://codeen.cs.princeton.edu`.

[Col13] Collanos workplace, 2013. `http://collanos-workplace.en.softonic.com`.

[cSp13] cSpace Project, 2013. `http://code.google.com/p/cspace`.

[CSWH01a] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[CSWH01b] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[DB09] A. Derhab and N. Badache. Data replication protocols for mobile ad-hoc networks: a survey and taxonomy. *Communications Surveys & Tutorials, IEEE*, 11(2):33–51, 2009.

[DCKM04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM '04*, pages 15–26, 2004.

[DHA03] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *ICDCS'03*, page 76, 2003.

[DHJ+07] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, volume 7, pages 205–220, 2007.

[DKK+01] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 202–215, New York, NY, USA, 2001. ACM.

[DLS+04] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a dht for low latency and high throughput. In *PROCEEDINGS OF THE 1ST NSDI*, pages 85–98, 2004.

[DR01] P. Druschel and A. Rowstron. Past: a large-scale, persistent peer-to-peer storage utility. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 75–80, 2001.

[dSLMM08] A. da Silva, E. Leonardi, M. Mellia, and M. Meo. A bandwidth-aware scheduling strategy for p2p-tv systems. In *Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on*, pages 279–288, 2008.

[eDo13] eDonkey2000, 2013. `http://edonkey2000.en.softonic.com`.

[EDPK09] M. El Dick, E. Pacitti, and B. Kemme. Flower-cdn: a hybrid p2p overlay for efficient query processing in cdn. In *EDBT '09*, pages 427–438, 2009.

[eMu13] eMule, 2013. `http://www.emule-project.net`.

[fac13] facebook, 2013. `http://www.facebook.com`.

[Far13] Faroo peer-to-peer web search, 2013. `http://www.faroo.com`.

[FJJ+01] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. Idmaps: a global internet host distance estimation service. *IEEE/ACM Trans. Netw.*, 9(5):525–540, 2001.

[FLC09] B. Fan, J. C. S. Lui, and D.-M. Chiu. The design trade-offs of bittorrent-like file sharing protocols. *IEEE/ACM Trans. Netw.*, 17(2):365–376, April 2009.

[FM03] M. Freedman and D. Mazières. Sloppy hashing and self-organizing clusters. In *In IPTPS*, pages 45–55, 2003.

[Fre10] M. J. Freedman. Experiences with coralcdn: A five-year operational view. In *Proc. 7th USENIX/ACM Symposium on Networked Systems Design and Implementation*, pages 95–110, 2010.

[Fre13] Freenet Project, 2013. `http://FreenetProject.org`.

[GAH07] A. Ghodsi, L. O. Alima, and S. Haridi. Symmetric replication for structured peer-to-peer systems. In *Databases, Information Systems, and Peer-to-Peer Computing*, pages 74–85. Springer, 2007.

[GB03] P. Ganesan and M. Bawa. Distributed balanced tables: Not making a hash of it all. Technical Report 2003-71, Stanford InfoLab, 2003.

[GBGM04] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *VLDB '04*, pages 444–455, 2004.

[GEBF+03] L. Garces-Erice, E. W. Biersack, P. A. Felber, K. W. Ross, and G. Urvoy-Keller. Hierarchical peer-to-peer systems. In *Proceedings of ACM/IFIP International Conference on Parallel and Distributed Computing*, pages 643–657, 2003.

[GGG+03] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03*, pages 381–394, 2003.

[GGGM04] P. Ganesan, K. Gummadi, and H. Garcia-Molina. Canon in g major: Designing dhts with hierarchical structure. In *ICDCS'04*, pages 263–272, 2004.

[GL02] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[GNZ06] S. Gurun, P. Nagpurkar, and B. Y. Zhao. Energy consumption and conservation in mobile peer-to-peer systems. In *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, MobiShare '06, pages 18–23, New York, NY, USA, 2006. ACM.

[GS05] P. B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *PROC. OF IEEE INFOCOM*, 2005.

[gtk13] gtk-gnutella, 2013. `http://gtk-gnutella.sourceforge.net`.

[HHL+03] R. Huebsch, J. M. Hellerstein, N. Lanham, B. Thau Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *'03*, pages 321–332, 2003.

[HJS+03] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *USENIX Symposium on Internet Technologies and Systems (USITS'03)*, 2003.

[HLZ+04] J. Hu, M. Li, W. Zheng, D. Wang, N. Ning, and H. Dong. Smartboa: constructing p2p overlay network in the heterogeneous internet using irregular routing tables. In *Proceedings of the Third international conference on Peer-to-Peer Systems*, IPTPS'04, pages 278–287, Berlin, Heidelberg, 2004. Springer-Verlag.

[HR02] S. Hand and T. Roscoe. Mnemosyne: Peer-to-peer steganographic storage. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 130–140. Springer Berlin Heidelberg, 2002.

[HSC01] M. Hauspie, D. Simplot, and J. Carle. Replication decision algorithm based on link evaluation for services in manet. *CNRS UPRESA*, 2001.

[I2P13] I2P Technical Introduction, 2013. `http://www.i2p2.de/techintro.html`.

[IB09] S. K. Ingmar Baumgart, Bernhard Heep. Oversim: A scalable and flexible overlay framework for simulation and real network applications. In *IEEE P2P'09*, 2009.

[IRD02] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 213–222. ACM, 2002.

[Jab13] Jabber, 2013. `http://www.jabber.de`.

[JBu13] JBuilder, 2013. `http://www.embarcadero.com/products/jbuilder`.

[JMW03] S. Jain, R. Mahajan, and D. Wetherall. A study of the performance potential of dht-based overlays. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 11–11, Berkeley, CA, USA, 2003. USENIX Association.

[JOK09] R. Jiménez, F. Osmani, and B. Knutsson. Connectivity properties of mainline bittorrent dht nodes. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*, pages 262–270. IEEE, 2009.

[JOV05] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. Baton: A balanced tree structure for peer-to-peer networks. In *In VLDB*, pages 661–672, 2005.

[KBC+00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, 35(11):190–201, 2000.

[KHKY09] O. Kassinen, E. Harjula, J. Korhonen, and M. Ylianttila. Battery life of mobile peers with umts and wlan in a kademlia-based p2p overlay. In *Personal, Indoor and Mobile Radio Communications, 2009 IEEE 20th International Symposium on*, pages 662–665, 2009.

[KK03] M. F. Kaashoek and D. R. Karger. Koorder: A simple degree-optimal distributed hash table. In *In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'2003*, 2003.

[KKF06] R. Kummer, P. Kropf, and P. Felber. Distributed lookup in structured peer-to-peer ad-hoc networks. In R. Meersman and Z. Tari, editors, *On the Move '06: CoopIS, DOA, GADA, and ODBASE*, volume 4276 of *Lecture Notes in Computer Science*, pages 1541–1554. Springer Berlin / Heidelberg, 2006.

[KKHY13] T. Koskela, O. Kassinen, E. Harjula, and M. Ylianttila. P2p group management systems: A conceptual analysis. *ACM Comput. Surv.*, 45(2):20:1–20:25, March 2013.

[Kle00] J. Kleinberg. The small-world phenomenon: An algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on theory of computing*, pages 163–170, 2000.

[KLKP08] S. Kaune, T. Lauinger, A. Kovačević, and K. Pussep. Embracing the peer next door: Proximity in kademlia. In *P2P '08*, 2008.

[KLL$^+$97] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *STOC*, pages 654–663, 1997.

[KMP99] G. Karumanchi, S. Muralidharan, and R. Prakash. Information dissemination in partitionable mobile ad hoc networks. In *Reliable Distributed Systems, 1999. Proceedings of the 18th IEEE Symposium on*, pages 4–13. IEEE, 1999.

[KN08] I. Kelenyi and J. Nurminen. Optimizing energy consumption of mobile nodes in heterogeneous kademlia-based distributed hash tables. In *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST '08. The Second International Conference on*, pages 70–75, 2008.

[KN10] G. Kreitz and F. Niemela. Spotify – large scale, low latency, p2p music-on-demand streaming. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–10, 2010.

[KR04] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA'04*, pages 36–43, 2004.

[KSHK08] D. Klan, K.-U. Sattler, K. Hose, and M. Karnstedt. Decentralized managing of replication objects in massively distributed systems. In *Proceedings of the 2008 international workshop on Data management in peer-to-peer systems*, pages 19–26. ACM, 2008.

[KSR$^+$07] M. Karnstedt, K.-U. Sattler, M. Richtarsky, J. Muller, M. Hauswirth, R. Schmidt, and R. John. Unistore: querying a dht-based universal storage. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 1503–1504. IEEE, 2007. `http://www.p-grid.org/implementation/extensions.html`.

[Lap85] J.-C. Laprie. Dependable computing and fault-tolerance. *Digest of Papers FTCS-15*, pages 2–11, 1985.

[Lim13] Former limewire homepage, 2013. `http://www.limewire.com`.

[Liu09] J. Liu. Mojito under churn, 2009. `http://www.cs.utexas.edu`.

[LKRG03] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *SIGCOMM*, pages 395–406, 2003.

[LS13] W. Lehner and K.-U. Sattler. Transactional data management services for the cloud. In *Web-Scale Data Management for the Cloud*, pages 59–90. Springer, 2013.

[LSM$^+$05] J. Li, J. Stribling, R. Morris, M. Kaashoek, and T. Gil. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 225–236 vol. 1, 2005.

[LSW06] T. Locher, S. Schmid, and R. Wattenhofer. equus: A provably robust and locality-aware peer-to-peer system. In *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, pages 3–11, 2006.

[LZT04] M. Landers, H. Zhang, and K.-L. Tan. Peerstore: Better performance by relaxing in peer-to-peer backup. In *Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on*, pages 72–79. IEEE, 2004.

[LZZ$^+$04] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster computing on the fly: P2p scheduling of idle cycles in the internet. In *Proceedings of the Third international conference on Peer-to-Peer Systems*, IPTPS'04, pages 227–236, Berlin, Heidelberg, 2004. Springer-Verlag.

[Man04] G. S. Manku. Balanced binary trees for id management and load balance in distributed hash tables. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, pages 197–205, New York, NY, USA, 2004. ACM.

[MBK07] B. Maniymaran, M. Bertier, and A.-M. Kermarrec. Build one, get one free: Leveraging the coexistence of multiple p2p overlay networks. In *ICDCS '07*, pages 33–33, June 2007.

[MBR03] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140, 2003.

[MC04] A. Moon and H. Cho. Energy efficient replication extended database state machine in mobile ad hoc network. In *IADIS International Conference on Applied Computing*, pages 224–228, 2004.

[MD04] A. Mislove and P. Druschel. Providing administrative control and autonomy in structured peer-to-peer overlays, 2004.

[Mil67] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *IPTPS*, 2002.

[MNR02] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st annual ACM symposium on Principles of distributed computing.* ACM Press, 2002.

[MPP10] G. Millar, E. Panaousis, and C. Politis. Robust: Reliable overlay based utilisation of services and topology for emergency manets. In *Future Network and Mobile Summit '10*, pages 1 –8, 2010.

[NW03] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '03, pages 50–59, New York, NY, USA, 2003. ACM.

[NZ01] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *In INFOCOM*, pages 170–179, 2001.

[NZ06] D. Novak and P. Zezula. M-chord: a scalable distributed similarity search structure. In *InfoScale*, page 19, 2006.

[OHY09] Z. Ou, E. Harjula, and M. Ylianttila. Effects of different churn models on the performance of structured peer-to-peer networks. In *Personal, Indoor and Mobile Radio Communications, 2009 IEEE 20th International Symposium on*, pages 2856–2860, 2009.

[PDH04] H. Pucha, S. Das, and Y. Hu. Ekta: an efficient dht substrate for distributed applications in mobile ad hoc networks. In *WMCSA '04*, pages 163 – 173, 2004.

[PGVA08] P. Padmanabhan, L. Gruenwald, A. Vallur, and M. Atiquzzaman. A survey of data replication techniques for mobile ad hoc network databases. *The VLDB Journal-The International Journal on Very Large Data Bases*, 17(5):1143–1164, 2008.

[PRR97a] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distribute d Environment. In *SPAA*, 1997.

[PRR97b] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–20, June 1997. `http://www.cs.utexas.edu/users/plaxton/ps/1997/spaa.ps`.

[Pug90] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6), 1990.

[RB03] M. Roussopoulos and M. Baker. Cup: Controlled update propagation in peer-to-peer networks. In *USENIX Annual Technical Conference, General Track*, pages 167–180, 2003.

[RB07] M. Rogers and S. Bhatti. How to disappear completely: A survey of private peer-to-peer networks. *networks*, 13:14, 2007.

[RB11] L. 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.

[RBS12] L. Ribe-Baumann and K.-U. Sattler. A hierarchical approach to resource awareness in dhts for mobile data management. In *2nd International Workshop on Information Management for Mobile Applications*, pages 5–12, 2012.

[RBS13] L. Ribe-Baumann and K.-U. Sattler. A hierarchical approach to resource awareness in {DHTs} for mobile data management. *Pervasive and Mobile Computing*, to appear 2013. `http://www.sciencedirect.com/science/article/pii/S1574119213000990`.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[RDB10] K. Rzadca, A. Datta, and S. Buchegger. Replica placement in p2p storage: Complexity and game theoretic analyses. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 599–609. IEEE, 2010.

[Ret13] Retroshare development blog, 2013. `http://retroshareteam.wordpress.com`.

[RFH$^+$01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *SIGCOMM'01*, 2001.

[RGJZ04] S. Ren, L. Guo, S. Jiang, and X. Zhang. Sat-match: a self-adaptive topology matching method to achieve low lookup latency in structured p2p overlay networks. In *IPDPS'04*, pages 83–91, April 2004.

[RGRK04] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *USENIX Annual Technical Conference (ATEC'04)*, pages 10–10, 2004.

[RH13] F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. In *Security and Privacy in Social Networks*, pages 197–223. Springer, 2013.

[RHKS02] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *INFPCOM 2002*, 2002.

[RKY$^+$02] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. Ght: a geographic hash table for data-centric storage. In *WSNA '02*, pages 78–87, New York, NY, USA, 2002. ACM.

[RLS+03] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[RM06] J. Risson and T. Moors. Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks*, 50(17):3485 – 3521, 2006.

[RQMH06] J. Risson, S. Qazi, T. Moors, and A. Harwood. A dependable global location service using rendezvous on hierarchic distributed hash tables. In *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, ICNICONSMCL '06, pages 4–, Washington, DC, USA, 2006. IEEE Computer Society.

[RRP+03] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108, New York, NY, USA, 2003. ACM.

[RS04] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *NSDI*, volume 4, pages 8–8, 2004.

[SENB09] M. Steiner, T. En-Najjary, and E. W. Biersack. Long term study of peer behavior in the kad dht. *IEEE/ACM Transactions on Networking (TON)*, 17(5):1371–1384, 2009.

[SGE+05] O. D. Sahin, A. Gulbeden, F. Emekçi, D. Agrawal, and A. El Abbadi. Prism: indexing multi-dimensional data in p2p networks using reference vectors. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 946–955. ACM, 2005.

[SGG02] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the Multimedia Computing and Networking*, 2002.

[SHD+06] E. Sit, A. Haeberlen, F. Dabek, B. gon Chun, H. Weatherspoon, R. Morris, M. F. Kaashoek, and J. Kubiatowicz. Proactive replication for data durability. In *Proceedings of the 5th Intl Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[She10] H. Shen. An efficient and adaptive decentralized file replication algorithm in p2p file sharing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 21(6):827–840, 2010.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01*, pages 149–160, 2001.

[TWS+09] Z. Tian, X. Wen, Y. Sun, W. Zheng, and Y. Cheng. Improved bamboo algorithm based on hierarchical network model. In *CCCM '09*, volume 1, pages 297 –300, 2009.

[TXM03] C. Tang, Z. Xu, and M. Mahalingam. psearch: Information retrieval in structured overlays. *ACM SIGCOMM Computer Communication Review*, 33(1):89–94, 2003.

[TXZ+05] R. Tian, Y. Xiong, Q. Zhang, B. Li, B. Y. Zhao, and X. Li. Hybrid overlay structure based on random walks. In *In Proc. of the 4th Intl. Workshop on Peer-toPeer Systems (IPTPS'05*, 2005.

[VLO10] Q. H. Vu, M. Lupu, and B. C. Ooi. *Peer-to-peer computing.* Springer, 2010.

[Vuz13] Vuze bittorrent client, 2013. `http://www.vuze.com`.

[WEZ+10] F. Wenzel, M. Erdik, J. Zschau, J. Fischer, I. Christ, and C. Kiehle. Edim - earthquake disaster information system for the marmara region, turkey. In *European Geosciences Union, General Assembly 2010*, 2.-7. May 2010.

[WL02] K. H. Wang and B. Li. Efficient and guaranteed service coverage in partitionable mobile ad-hoc networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 1089–1098. IEEE, 2002.

[WR03] M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. *SIGCOMM Comput. Commun. Rev.*, 33:101–106, January 2003.

[XMH03] Z. Xu, R. Min, and Y. Hu. Hieras: A dht based hierarchical p2p routing algorithm. In *ICPP'03*, page 187, 2003.

[XMK03] Z. Xu, M. Mahalingam, and M. Karlsson. Turning heterogeneity into an advantage in overlay routing. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1499–1509 vol.2, 2003.

[Yac13] Yacy decentralized web search, 2013. `http://yacy.net`.

[YCM06] A. Yip, B. Chen, and R. Morris. Pastwatch: A distributed version control system. In *NSDI*, 2006.

[Yel13] Yelp, 2013. `http://www.yelp.com`.

[YV11] A. Yu and S. Vuong. A dht-based hierarchical overlay for peer-to-peer mmogs over manets. In *IWCMC '11*, pages 1475 –1480, 2011.

[ZARBH13] B. Zafar, R. Alieiev, L. Ribe-Baumann, and M. Haardt. Dhts for cluster-based ad-hoc networks employing multi-hop relaying. In *The 7th Workshop on Wireless Mesh and Ad Hoc Networks*, to appear 2013.

[ZDK06] S. Zoels, Z. Despotovic, and W. Kellerer. Cost-based analysis of hierarchical dht design. In *P2P '06*, pages 233–239, 2006.

[ZDK07] S. Zoels, Z. Despotovic, and W. Kellerer. Load balancing in a hierarchical dht-based p2p system. In *Proceedings of the 2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, COLCOM '07, pages 353–361, Washington, DC, USA, 2007. IEEE Computer Society.

[ZGA$^+$10] B. Zafar, S. Gherekhloo, A. Asgharzadeh, M. T. Garrosi, and M. Haardt. Self-organizing network with intelligent relaying (sonir). In *Mobile Adhoc and Sensor Systems (MASS), 2010 IEEE 7th International Conference on*, pages 765–767. IEEE, 2010.

[ZHDK09] S. Zoels, Q. Hofstätter, Z. Despotovic, and W. Kellerer. Achieving and maintaining cost-optimal operation of a hierarchical dht system. In *Proceedings of the 2009 IEEE international conference on Communications*, ICC'09, pages 2194–2199, Piscataway, NJ, USA, 2009. IEEE Press.

[ZHS$^+$04] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004.

[ZKJ01] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-are location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.

[ZS05] T. Zahn and J. Schiller. Madpastry: a dht substrate for practicably sized manets. In *ASWN '05*, 2005.

[ZZJ$^+$01] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20. ACM, 2001.

[ZZZ$^+$03] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. D. Joseph, and J. Kubiatowicz. Approximate object location and spam filtering on peer-to-peer systems. In *Middleware 2003*, pages 1–20. Springer, 2003. `http://www.zhoufeng.net/eng/spamwatch`.