Peer-to-Peer Concepts for Emergency First Response

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades
Doctor rerum naturalium (Dr. rer. nat.)

von

Dipl. Wirtsch. Inform. Dirk Bradler

geboren in *Erbach (Odenwald)*

Referenten
Prof. Dr. Max Mühlhäuser (TU Darmstadt)
Prof. Dr. Jussi Kangasharju (University of Helsinki)

Tag der Einreichung: 26.04.2010
Tag der mündlichen Prüfung: 02.06.2010

Darmstadt 2010
Hochschulkennziffer D17

Acknowledgements

This work was only possible with the support and encouragement of my family and friends and colleagues over the last years, which I would like to acknowledge here. First and foremost, I would like to thank my advisor, Max Mühlhäuser, for his support, patience, excellent advice and his encouragements. I am also grateful to Jussi Kangasharju (University of Helsinki), who supervised large parts of my work and acts as second referee. I am grateful to all at Telecooperation and RBG for providing me with a friendly and supportive place to work. Especially, I would like to thank Thorsten Strufe for all the fruitful discussions and support. I am also grateful to all the other members in the Telecooperation Group who proofread papers and provided feedback (Sebastian Ries, Alexander Behring, Immanuel Schweizer, Mohammadreza Khalilbeigi to name only few of them).

I would also like to thank Lachezar Krumov from Algorithmik for his support. Furthermore, I would like to acknowledge Joanna Geibig (HU Berlin), Aleksandra Kovacevic (KOM/TU Darmstadt), Kalman Graffi (KOM/TU Darmstadt), Christof Leng (DVS/TU Darmstadt) and Wesley Terpstra (DVS/TU Darmstadt) for their advice.

I am highly grateful to my sister and my father for their support and patience during the course of this work. Finally, many thanks are due to Oxana for her unlimited support and for standing all the stressful time with me when finishing this thesis.

Abstract

Peer-to-peer (P2P) technology has already been established in several application domains, e.g., IP telephony, file sharing, and content distribution. It is considered to be distributed, ad-hoc, robust and scalable approach for digital information transfer. In this thesis, the P2P communication paradigm is proposed as an alternative communication approach in the first response application domain.

Disaster relief efforts, after larger scale catastrophes, suffer from damaged or destroyed communication infrastructure. Satellite connection equipment is hardly available and costly, and therefore rescue workers tend to send foot messengers to relay messages between the on-site organizations.
A P2P-inspired communication approach for first responders would relieve the overloading of communication channels, can function as a completely self-contained method and remove the need for centrally managed communication approaches, which might be damaged or destroyed after catastrophes.

In this thesis, we investigate a breakdown of the P2P-inspired communication approach in four distinct layers. We identify the key challenges for each layer and propose novel approaches for the most important challenges in each layer.

The main contributions are: (i) a systematic breakdown of the communication concept in four distinct layers, and (ii) a mechanism called 'BridgeFinder', which increases the robustness of the communication network. The overlay network (iii) 'Pathfinder' provides key functionalities like routing, lookup and exhaustive search. The application-level multicast (iv) provides an efficient way of sending messages to multiple recipients. The novel mechanism (v), called DCC, provides reliable command and control structure management in a distributed fashion.
Besides the technical concept, we developed a simulation environment with working prototypes for each contribution. Further, we evaluated the robustness, scalability and efficiency of these communication approaches. We show that they meet the stipulated requirements of first responders and perform at least equally or better than the current approaches.

Zusammenfassung

Peer-to-Peer-Technologie (P2P) ist bereits heute in unterschiedlichen Anwendungsgebieten etabliert, z.B. im Bereich IP-Telefonie, gemeinsamer Dateizugriff und 'Content Distribution'. P2P ist bekannt als dezentraler, spontan verfügbarer, robuster und skalierbarer Ansatz für den digitalen Informationsverkehr. In dieser Arbeit wird das P2P-Kommunikationsparadigma als alternativer Kommunikationsansatz für den Bereich der Katastrophenhilfe vorgeschlagen.

Im Falle größerer Katastrophen werden Hilfsmaßnahmen durch beschädigte oder zerstörte Kommunkationsinfrastruktur deutlich erschwert. Satellitenverbindungen sind selten verfügbar und kostspielig, dadurch tendieren Ersthelfer dazu Nachrichten mittels Boten den beteiligten Organisationen vor Ort zuzustellen.
Ein P2P-basierter Kommunikationsansatz für Ersthelfer kann die Überlastung der bestehenden Kommunikationsinfrastruktur mindern und als eigenständige Kommunikationsmethode fungieren. Es wäre keine zentrale Verwaltung notwendig, welche beschädigt oder zerstört werden könnte.

In dieser Arbeit wird eine Aufteilung des P2P-basierten Kommunikationsansatz in vier separate Schichten vorgeschlagen. Die Aufgaben und Herausforderungen jeder Schicht werden identifiziert und neue Ansätze für die wichtigsten Herausforderungen vorgeschlagen.

Die Hauptbeiträge dieser Arbeit sind: Eine systematische Aufteilung des Kommunikationsansatzes in vier separate Schichten (i) sowie der 'BridgeFinder' Mechanismus (ii), welcher die Robustheit des Netzwerks erhöht. Das Overlay Netzwerk (iii) 'PathFinder' bietet eine einzigartige Kombination der Schlüsselfunktionalitäten Routing, Key-Lookup und Volltextsuche. Das Application-Level Multicast (iv) bietet eine effiziente Möglichkeit Nachrichten an Empfängergruppen zu senden. Der neuartige Mechanismus (v) 'DCC' ermöglicht ein verlässliches Management der Organisationsstruktur in einem verteilten System.
Neben dem Konzept wird eine Simulationsumgebung zur Evaluation der vorgeschlagenen Methoden vorgestellt. Es wurden Robustheit, Skalierbarkeit und Effizienz der erarbeiteten Mechanismen evaluiert. Die erhobenen Anforderungen der Ersthelfer konnten durchgehend erfüllt werden, die entwickelten Ansätze sind darüber hinaus in den relevanten Bereichen performanter als die bisher vorgeschlagenen Mechanismen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Peer-to-peer (P2P) technologies have been established in several application domains, e.g., IP telephony, file sharing, and content distribution. The most prevalent P2P application is Skype [Tho06], which is used for instant messaging and video conferences. In addition e.g., Gnutella [KM02] and eMule [KB] also found their way out of the laboratory and are used for file sharing purposes. Applications for TV streaming such as PPLive and TVU, which are both based on P2P technology have left beta status and are available to end users.

The advantages of the P2P approach are on the one side the self configuration properties and the elimination of a single point of failure in the form of a central server, but more important is the scalability of P2P systems. As long as a peer is connected to the overlay network it donates parts of its resources to the P2P system. Therefore, even P2P networks with several millions of peers are possible, because every peer brings its required resources on its own.

In this thesis, the P2P communication paradigm is proposed as an alternative communication approach in the application domain of first responders. Using a P2P-based communication approach for first responders, it is possible to relieve the overloading of communication channels and to remove the need for centrally managed communication approaches, which might be damaged or destroyed after catastrophes. The proposed P2P system is intended to be used after the existing communication infrastructure has been heavily damaged, e.g., after an earthquake or a hurricane. The need for reliable first response communications was one of the first findings of the 9/11 attack on the Pentagon. In order to set realistic requirements to address the needs of first response helpers, this work builds on analyzed findings of disaster reports and interviewed first response helpers [Dir08].
In this thesis, a first response communication system is developed and evaluated by extensive simulations. The main contribution is a P2P approach for communication requirements at the incident area. Communication takes place primarily over wireless links and between small handheld devices carried by the personnel. In addition, there may be some optional servers (JAN towers) handling other tasks.

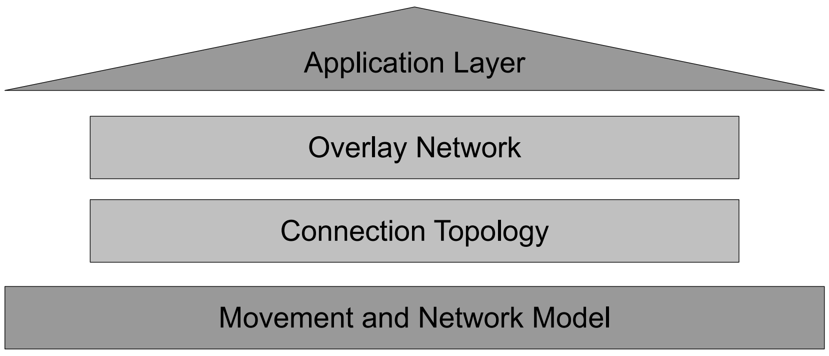


Figure 1.1: Layered Communication Model for First Responders

In recent larger scale disasters, communication could only be kept alive by explicitly assigning foot messengers to carry messages from source to target. The goal is to replace this last resort communication approach with a distributed, reliable, self-organizing and versatile communication system. The challenge is to provide reliable communication, even though parts of the network may be damaged or destroyed.

1.1 Scientific Contribution and Outline

In this thesis, we explore a breakdown of the P2P-inspired communication approach in four distinct layers. After identifying the key challenges for each layer, we propose novel approaches for the most important challenges.
In order to provide larger scale evaluations of first response communication approaches, simulations of recent or artificial disasters are performed. We developed a simulation framework, called the First Response Communication Sandbox (FRCS) [BPSM08], which combines both, mobility of first responders and the behaviors of communication devices.
Based on the requirements of first responders and the conducted simulations, this thesis presents a P2P-based communication approach, which is designed especially for the hostile environments that result from catastrophes.

Figure 1.1 shows the Layered Communication Model used in this thesis. Each layer features its specific tasks and challenges.
The fundament is the 'Movement and Network Model' layer. It provides both, movement data from all participants of the scenario, and a basic connection graph of all devices within communication range. On the next layer, called 'Connection Topology', all possible communication partners within range are preselected. The challenge is to choose qualified neighbors from the large number of available nodes and to protect the network from partitioning. On top, the 'Overlay Network' layer provides search and lookup functionalities among all connected nodes. The challenge is, on the one side, to save bandwidth and on the other side to provide

fast access to all available resources. The topmost layer is the ’Application’ layer. P2P applications are allowed to access methods provided by the layers below. Nevertheless, a consistent view of the provided resources is the prevalent challenge in the Application layer.

Each layer of the communication model is designed especially for the harsh environment found in first response situations. For each layer, at least one contribution is presented in detail. From bottom to top, the most important contributions presented are:

1. On the ’Movement and Network Model’ layer: A simulation environment for the Movement and Network Model layer, which estimates movement of first responders and civilians and simultaneously simulates the corresponding mobile communication infrastructure.

2. On the ’Communication Topology’ layer: An approach called BridgeFinder, which provides a mechanism for detecting critical paths between sparsely connected clusters of mobile peers.

3. On the ’Overlay Network’ layer: An Application-Level Multicast in the Overlay Network layer. A message with optional acknowledgment can be sent to either all involved parties or to a defined subset of peers within the network. The approach guarantees that no duplicate messages occur, which saves bandwidth and energy.

4. On the ’Overlay Network’ layer: A P2P overlay network called Pathfinder, which adapts itself dynamically to the current stability of the network. It is a contribution for the Overlay Network Layer.

5. On the ’Application’ layer: P2P functionality, which is able to provide hierarchical data access in structured P2P overlay networks. It is a contribution to the Application layer.

1.2 Publications

The main contributions of this thesis have been published as research contributions in computer science conference proceedings.
An overview of the current application domains of P2P-technology has been published in [GKM$^+$07] The general approach of this work has been published in [BSAL09]. In detail, the requirement analysis have been published in [Dir08], followed by the analysis of existing P2P-overlay networks [BKM08]. A P2P mechanism for sending multicast messages has been published in [BKM09b]; alternative approaches are presented in [KTL$^+$09]. The simulation environment has been published in [BPSM08] and [BKM09a], as well as an example An example application for first responders in [BKK09]. The mechanism for identifying critical

nodes within the network called BridgeFinder, and a P2P-overlay network, which combines structured and unstructured search methods, are currently in submission.

1.3 Thesis Structure

The thesis is structured as follows:
Chapter 2 shows the requirements found typically in first response situations. Chapter 3 presents the derived coarse communication concept, which is recommended for communication in larger scale catastrophes.
In Chapter 4, the foundation for the simulation approach of rescue units is elicited and implemented. It is used for simulation of communication patterns during a disaster scenario. Chapter 5 systematically analyses how to identify articulation points of sparsely connected node clusters; these points are crucial for keeping the network connected. Chapter 6 analyses the applicability of existing P2P overlay networks for the harsh environment common to first response situations. The findings lead to the development of a novel P2P overlay called PathFinder. In Chapter 7, a new mechanism for an efficient application-level multicast is developed and evaluated. Chapter 8 shows one implemented mechanism, built on the proposed P2P communication approach. It maintains command and control hierarchy in a distributed environment.
Chapter 9 concludes this thesis with a summary and provides directions for future research opportunities.

Chapter 2

First Response Communication Requirements

In this Chapter, requirements leading to the thesis's proposed communication architecture for first responders are explained in greater detail.
First, by surveying the demand of first responders through extensive telephone interviews and from existing reports, the most important use cases are identified. Based on these findings, non-functional requirements and design requirements demanded by national authorities are analyzed. Finally we summarize the most important findings, which lead to the coarse architecture proposed in this thesis.

The remainder of this Chapter is structured as follows. In Section 2.1, we present the current state of disaster relief efforts and demonstrate the need for a distributed communication approach. In Section 2.2 and Section 2.3 we summarize the these use cases and the evaluation methodology. The non-functional requirements are presented in Section 2.4. Finally, in Section 2.5 we present the System-of-Systems concept proposed by homeland security. The Chapter concludes with a summary of requirements (Section 2.6), which lead to the proposed architecture and mechanisms put forth in this thesis.

2.1 Introduction

Working communications are extremely important in disaster management and first response scenarios. In this Chapter, the main requirements of a communication system for first response scenarios are elicited. Flexible communication architectures are vital for larger scale disaster management, and existing communication mechanisms have several shortcomings in light of the heterogeneity of first response groups. The need for reliable first response communications was one finding discovered after the 9/11 attack on the Pentagon: [bTSC02]

> "Because radio communications were overloaded and ineffective, Captain Liebold sent two firefighters on foot to record the identification

number and location of every piece of equipment on the Pentagon grounds. In the first few hours, foot messengers at times proved to be the most reliable means of communicating."

This Chapter presents the requirements of the development of a first response communication sandbox, intended to simulate the situation that occurs after the existing communication infrastructure has been heavily damaged.
A P2P requirement analysis for first responders with prototyped software was done by the University of Virginia [A.S03a], they identified three main issues in current first response approaches and developed a prototype for a P2P based first response solution. Further implementation of this P2P approach [A.S03a] is completed using hypercast, GPS capabilities, multicast streaming video and access control mechanism. Nevertheless, the goal was to develop a prototype, architecture design and larger scale evaluations were out of scope.

Our requirements are based on reports and extensive telephone interviews with rescue professionals, as well as on the findings of recent larger scale disasters. In order to evaluate our proposed approach, we developed the First Response Commmunication Sandbox (FRCS) in order to measure metrics derived from the proposed requirements and to evaluate new approaches for tackling communication in first response situations. Both pieces, the collected use cases combined with FRCS allow for the evaluation and comparison of different disaster management solutions under exactly the same laboratory conditions. FRCS is used in the entire thesis and is presented in detail in Chapter 4.

2.2 Identified Use Cases

In this section, the basic requirements and use cases for a first response communication system are presented. Both, the requirements and use cases system from the results presented in the literature [A.S03a] [J. 02], as well as extensive telephone interviews conducted with professionals working on first response were analyzed. The FRCS [BPSM08] [BKM09a] is intended for simulating distributed rescue communication. The first response team on-site needs a reliable group communication system to establish communication which supports the organizational hierarchy [MB07]. On-site communications refer to the communication needs of the first response team. They take place mainly over wireless links and between small handheld devices carried by the personnel. In addition, there could be some servers handling other tasks. A distributed approach can be used as communication infrastructure in unreliable and rapidly changing environments. In fact, Groove Office [Gro04] has already been applied successfully during first response situations. Nevertheless, Groove had to provide a central backup server in order to make the approach scalable.

Our goal is to augment or replace foot messengers with a distributed communication system. Based on extensive telephone interviews, several use cases that the system

ID	**Name**	**Description**	**Example**
R1	Broadcast/ Multicast	Messages need to be sent to groups of peers, optionally with acknowledgment.	Operation Control sends evacuation message to first response team
R2	Shift Change	Immediate churn occurs, due to massive logins of new shift members and leaving peers from the previous shift.	Firefighter Team A must rest and responsibility is taken over by firefighter Team B
R3	Locality Awareness	Mobile user enters a certain area and automatically receives location aware news	A medic is walking toward a building and receives a warning not to enter the building
R4	Resource Awareness	Additional hardware is provided, and the network adapts to the new resources and optimally utilizes the additional hardware	Operation control provides additional hardware, and the network restructures itself using the new hardware to stabilize the overlay
R5	Active Search	Search for an object, optionally within location range	A first response team is searching for an available paramedic nearby
R6	Hierarchy Maintenance	In all catastrophic scenarios, there is a strict organization hierarchy for all helpers.	A new team at the site it is immediately assigned to a supervisor and is able to receive orders

Table 2.1: Use Case Table

would need to address are identified. These are listed in Table 2.1. An example for each use case is provided as well.

The use case **R1** - *Broadcast/Multicast* is necessary for sending important messages to a group of users. The challenge of broadcasting information is to reach every node regardless of its current position and its connection point. Table 2.2 shows an extract of the 24h-Timeline of the 11/9 Pentagon incident [bTSC02] The listed actions describe ‘multicast’ situations, i.e., messages that need to be delivered to a group of people or to all devices connected to the incident network.
A typical use case scenario is one in which the officer in charge wants to send an evacuation message to the first response team located in an endangered area.

The use case **R2** - *Shift Change* occurs regularly if one rescue team needs rest and its members’ roles are taken over by another team. The communication system must ensure that roles, open tasks, and solved issues are seamlessly adapted by the

Timeline	Reported Actions
10:15 a.m.	Chief Schwartz orders full evacuation because of warning of approaching hijacked aircraft
10:38 a.m.	Chief Schwartz sounds the all-clear, ending the evacuation
11:30 a.m.	Chief Schwartz establishes the ICS Operations Section at the Pentagon Heliport
2:00 p.m.	Second threat of unidentified aircraft causes full evacuation

Table 2.2: Arlington County Log Excerpt of After Action Report

new team.
A common scenario is when rescue team A needs to rest, and responsibility is taken over by team B.

In the use case **R3** - *Locality Awareness*, a digitally marked spot is visible to all members of the network. If another team approaches e.g., a bridge, it will be notified before it attempts to cross the bridge.
One critical scenario occurs when a paramedic is walking towards a building that may possible collapse soon; he should receive a warning to not enter the building.

The *Resource Awareness* use case **R4** ensures the optimal utilization of the available hardware. Communication is heavily influenced by heterogeneous hardware with dynamic fluctuation of resources.
During an operation, control provides additional hardware, and the network restructures itself using the new hardware for stabilizing the overlay.

Use case **R5** - *Active Search* covers the need to search for people based on their roles, distinct persons, or digitally marked tools. Usually the area that a first response team can observe is visually limited, so digital support for locating objects and people would improve the overview.
A typical active search scenario, is for example, searching for an available paramedic nearby.

R6 - *Hierarchy Maintenance* is crucial to the success of first response. Strict hierarchical decision making and reporting have become the standard for disaster management. It is important to mention that the typical hierarchy dynamically changes during an incident.

Table 2.3 shows an extract of the 24h-Timeline of the 9/11 Pentagon incident [bTSC02]. Disaster relief teams arrive incrementally at the site, and their communication devices should assign them to a supervisor immediately in order to receive orders. In the presented excerpt, 12 changes in the dynamic hierarchy were required within 50 minutes.

Timeline	Reported Actions
9:40 a.m.	Captain Chuck Gibbs arrives at the Pentagon
9:40 a.m.	Captain Mark Penn arrives at Arlington County EOC
9:41 a.m.	Battalion Chief Bob Cornwell arrives at the Pentagon and assumes Incident Command
9:41 a.m.	ACFD Truck 105 arrives at the Pentagon
9:42 a.m.	ACFD Captain Edward Blunt arrives at the Pentagon and establishes EMS Control
9:43 a.m.	MWAA first responders arrive at the Pentagon
9:48 a.m.	Assistant Chief James Schwartz arrives and assumes Incident Command
9:49 a.m.	FBI Special Agent Chris Combs arrives and is FBI representative to Incident Command
9:50 a.m.	Chief Schwartz establishes Fire Suppression Branch, River Division, EMS Division, and A-E Division
9:50 a.m.	Three area hospitals are prepared to receive patients
9:55 a.m.	Assistant Chief John White arrives and is assigned EMS Branch Commander
10:30 a.m.	Arlington County EOC is operational

Table 2.3: Arlington County Log of After Action Report

In addition, loose cross-organizational communication should be established, while maintaining the strict intra-organizational hierarchy [MB07].

2.3 Use Case Evaluation Methodology

A communication system for on-site communications for first response has a relatively small amount of users, on the order of a few hundreds or a couple of thousand, at most. Furthermore, the churn is expected to remain at moderate levels for most of the nodes.
Evaluation of a first response solution must cover both, the actual performance of typical use cases needed in catastrophic situations and a unified disaster scenario in order to create comparable results.
In order to get a fairly reliable comparison of the different use cases, a metric for each use case is introduced. The metric is completely independent of the chosen communication technology and software.

Table 2.4 shows the basic evaluation methodology for each use case.

Nevertheless, a comparison is only possible with a “standardized” disaster scenario. Otherwise the results would vary, depending on the number of communication devices, network coverage, movement speed, resources of the devices, etc.
In order to overcome these numerous configuration discrepancies, we propose a

ID	**Name**	**Evaluation Criterion**	**Description**
R1	Broadcast/ Multicast	Average success rate	Fraction of successfully delivered broadcast/multicast messages
R2	Shift Change	Average recovery time	Average time needed for the network peer recovery time
R3	Locality Awareness	Notification Delay	Average delay between event and notification
R4	Resource Awareness	Resource utilization of peers	Relative resource utilization distribution of peers
R5	Active Search	Success rate of search	Fraction of correctly answered search queries / key lookups
R6	Hierarchy Maintenance	Average restructuration time	Time needed to successfully re-establish the command and control hierarchy

Table 2.4: Use Case Verification Table

standardized scenario setup description [BPSM08]. The combination of proposed use cases and the standardized scenario setup simplify comparisons between different first response communication approaches.
Both, a set of necessary use cases and a scenario description for first response situations are defined. With these two requirements met, a fair comparison of first response solutions is possible. However, a universal definition of a standardized disaster situation to evaluate first response communication approaches is not yet available.

Nevertheless, standardized evaluation scenarios are likely to evolve, and with our proposed scenario description, this process can be accelerated.

2.4 Non-functional Requirements

For a P2P first-response communication system, the use cases and requirements R1-R6 are analyzed in Section 2. The primary use of a distributed communications infrastructure in first response is to organize on-site communications in situations where most of the existing infrastructure has failed. The on-site teams need a method of reliable group communication that supports the organizational hierarchy of the response teams. Communication takes place over wireless links and between small handheld devices carried by the personnel. The networks are typically augmented with access points or more powerful communication nodes (e.g., communication vans). The main contribution is to augment the classical first response communica-

tion approach, which is often supported by foot messengers, and replace it with a distributed communication system.
When a crisis occurs, first responders from different professions, organizations and jurisdictions meet. Helpers from fire departments, law enforcement agencies, emergency medical services (EMS) and government agencies must work together in order to manage the situation and to save lives. Depending on the scenario, hazardous material (HAZMAT) workers, waste disposal technicians or victim rescue teams are also needed. In order to work together successfully, first responders from all professions, organizations and jurisdictions need must be able to communicate with each other. They must be able to communicate with their colleagues, the command center and other first responders. The reliance on voice-oriented communication alone is one of the major drawbacks of current first responder systems identified by [A.S03a].
We divided the non-functional requirements in the main areas: ‘Communication Types and Devices Types’, ‘Information Infrastructure’, ‘Network Structure’, and ‘Communication Schemes.’

Communication Types and Devices Types The information content of voice alone is not rich enough and can easily be misunderstood. In order to support first responders appropriately, they need the option of exchanging miscellaneous types of information; floor plans and area maps can provide better orientation; images and videos of the incident scene can help the command center to make more informed decisions instead of simply relying upon voice messages [A.S03a]. For that reason, it is important to enrich the communication channel:

> **R7 (Communication Channel)** - First Response communication shall provide opportunities for text, voice and picture transmissions.

The lack of interoperability is also a major problem. Most agencies and departments use different systems and equipment for communication. Sometimes, not even police and firefighters from the same county are able to communicate inter-disciplinarily using their standard devices. Thus, it is important to provide inter-organization communication interfaces:

> **R8 (Interoperability)** - First response communication shall provide channels between organizations.

Information Infrastructure The third major problem is the limited situational awareness of first responders and command centers. Even if a strong method of communication is established, the command center cannot rely on reports from first responders alone. They require access to databases, maps, personnel location, etc. to evaluate the situation and make well-informed decisions rapidly based on reliable information and a complete picture of the incident. In order to meet any of these requirements, an incident scene network must be established. It is formed without any existing infrastructure and is dismissed when the mission is accomplished. Connectivity between all participants must be provided, as well as seamless communication across incompatible devices and agencies. The system is mobile and temporary in nature and needs to be scalable to the dimension of the incident. It must allow mobile users and cannot rely on specific users or specific relay stations. Users must be able to connect or disconnect at any time without affecting the system negatively.

Network Structure Mobile communication devices carried by first responders could simply form a wireless network to communicate. Since the mobility pattern of first responders is hard to predict, this kind of network is not reliable [BL07]. Therefore, nomadic relay devices should be used, as they can be carried and deployed by first responders. They form a stationary ad hoc backbone network to link first responders to each other. Considering the nature and properties of incident scene networks, a flexible architecture is needed to establish a reliable communication infrastructure. A P2P network is the optimal solution. P2P networks are decentralized, distributed and server-less. They self-organize in a so-called overlay network and do not rely on a single entity's performance or availability. All peers in the system can connect to each other directly or via any path. The architecture is flexible enough to add features subsequently to address future needs [A.S03a]. Hence, using a P2P overlay network on top of an established connection of all participants will more than adequately meet first responders' needs. In order to support first responders, different types of information need to be exchanged [HHNL07]. Voice-oriented messages will always be important and require strict timeliness and high quality. Live video feeds from the incident scene improve situational awareness and make remote meetings and assistance possible. Databases need to be accessible, and the necessary information must be presented to first responders in a helpful way. Therefore the underlying network infrastructure needs to meet certain delay requirements, as discussed in [BL07]. Requirements of the 'Information Infrastructure' and the 'Network Structure' lead to:

> **R9 (Self-Sustaining)** - Incident scene networks may require to be established on demand without existing infrastructure.

Communication Schemes When examining the information exchange of first responders, two different types of information flow can be identified: vertical (upward and downward) and horizontal (parallel) communication [HHNL07] [CSRU05]. Vertical communication represents the information exchange between entities at different levels of the command hierarchy. It flows upward from first responders to commanders, or downward from the command center to first responders. Horizontal communication includes all information flows between first responders. Figure 2.1 shows the vertical and horizontal communication needs of first responders. Therefore, a communication infrastructure needs to implement different communication interfaces. These interfaces must be employed in such a way that they overcome interoperability issues between heterogeneous devices and agency policies. One-to-one and one-to-many communication schemes must be available to convey messages to certain first responders or groups according to their locations and roles. Requirements of the information flow lead to:

> **R10 (Information Flow)** - First response communication shall provide means for vertical and horizontal messaging.

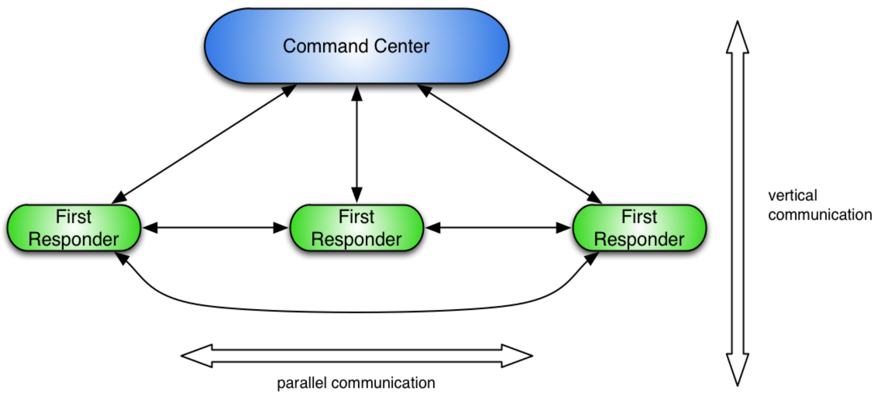


Figure 2.1: Information flow during crisis response

2.5 System of Systems

In the SAFECOM report [of06], the Department of Homeland Security introduces the so-called 'System of Systems'. It describes a network topology used to meet the requirements of first responder communication during an incident. It is designed to provide a connection between first responders and to command center. In [HHNL07], a similar network is described. The authors also survey implications regarding the architecture of first responder networks and discuss design criteria. These communication networks are mobile and temporary in nature because they must be deployed without the presence of an infrastructure. They are dismissed when the mission is accomplished. They must allow for the integration of other networks to connect incident scenes and to provide access to remote information sources. To support incidents of any size and allow first responders to connect or disconnect at any time, the architecture needs to be scalable and dynamic.

In present systems, communication often takes place between first responder and command center. First, responders cannot communicate among each other. Neither they nor the commanders are able to send messages to a certain subset of all helpers or address all personnel in a certain area. Multicast, publish subscribe systems and similar communication models are required to close this gap. These group communications are only possible if a network is established, that allows connections between every involved person. While the primary objective is to transport information between first responders, the infrastructure must also allow for the exchange of video feeds and miscellaneous data. This is necessary to raise the level of situational awareness.

System of Systems Hierarchy The System of System's network is hierarchically structured. It consists of PANs, IANs, JANs and an EAN all of which are logical concepts. The hierarchy with the proposed connection method is shown in Figure 2.2.

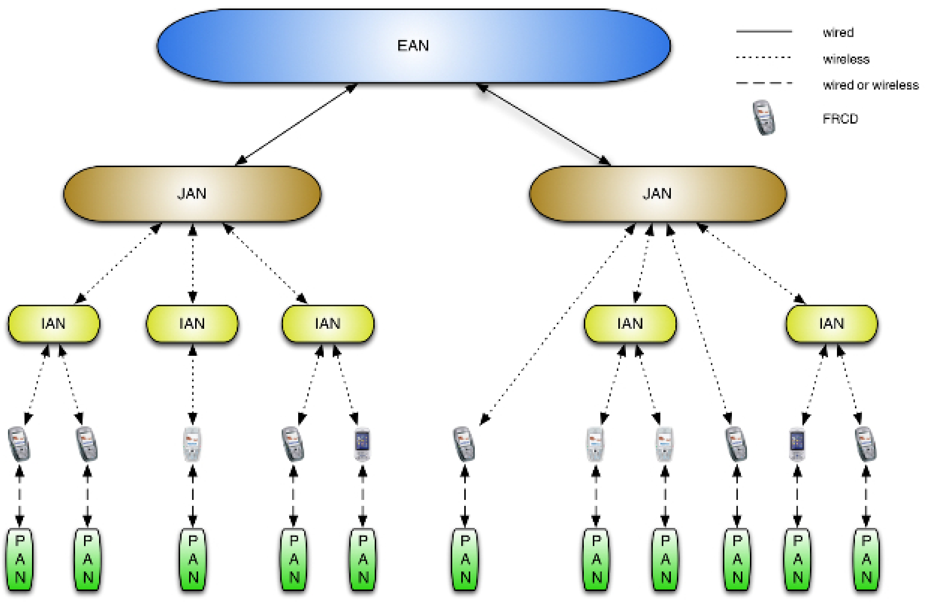


Figure 2.2: System of Systems

This thesis does provide novel approaches for network areas IAN and JAN. They are embedded in the the adjacent network areas PAN and EAN. Therefore, a basic understanding of EAN and PAN is required as well.

A *Personal Area Network* (PAN) represents the set of devices that are carried out by first responders or embedded in their clothing.
The *Extended Area Network* (EAN) is the backbone network for connecting IANs and JANs. It provides access to the Internet, as well as county, regional, state and national systems.
An *Incident Area Networks* (IAN) is centered on wireless access points, such as droppable relay devices or vehicle-mounted nodes. They are deployed on demand and create multi-hop, ad hoc wireless networks to connect PANs and JANs. They thereby act as a gateway for mobile first responders and connect them to the network.

These access points are deployed at the incident scene and can scale to the magnitude of the emergency.
Jurisdiction Area Networks (JANs) form the main communication network by handling the access of IANs to the network and the EAN. They also provide connection to a PSCD if the responsible IAN fails. These are of a more permanent nature and includes communication towers. If no JAN node is available, IANs will form an ad hoc network among themselves to maintain connectivity.
Partitioning of the communication areas lead to:

> **R11 (Communication Areas)** - First response communication shall be divided in the following categories: Personal Area Network, Incident Area Network, Jurisdiction Area Network and Extended Area Nework.

PAN: Personal Area Network The Personal Area Network is a small-scale wireless network for communication among devices and sensors, carried or embedded in clothing. The PSCD is the central communications hub of an involved individual and has three different networking interfaces: a wired field bus, a low-power wireless radio, and a WLAN radio. The first two allow communication with other devices in the PAN, while the latter one forms the uplink to the IAN. A wired field bus allows first response members to connect small devices, such as sensors embedded in clothing to the PSCD. It could be implemented using the Controller Area Network (CAN) standard. Since radio communication always costs additional power, a wired bus is very efficient in terms of energy. In addition, peripherals can be directly powered through the bus.

Energy Consumption In many cases, sensors sleep most of the time. If they are equipped with wireless network interfaces, it is important that listening for incoming data packets does not consume much energy. The ZigBee and IEEE 802.15.4 standards were specifically developed for sensor networks and address this issue. Similarly, the low power requirements of GSM, when solely listening for incoming calls, enable one to build cellphones with long standby times.

Size Bluetooth modules, together with the necessary antennae, are considerably smaller than WLAN or GPRS solutions. ZigBee products are not yet very mature. However, the industry is working on complete system-on- chip (SoC) solutions containing a microcontroller, baseband, and radio. In order to integrate the next generation wireless devices in the PAN, we propose using the ZigBee standard, which has been specifically designed for low-power sensor applications. Technologies like WLAN or Bluetooth do not meet these requirements. Bluetooth piconet is limited to one master and seven slaves. Furthermore, only a master can initiate the communication, and inquiry is slow and consumes a lot of power.

Consequently, body-worn sensors are mostly based on proprietary radio technologies today, e.g., the Nike+iPod shoe pedometer or Polar heart rate monitors. The WLAN interface of the PSCD provides the uplink into the IAN. Compared to the PAN, it requires a wider range and a higher bandwidth. Therefore, it will also require more power. To implement this radio interface, the well-established 802.11 standards can be used.

IAN: Incident Area Network The IAN is centered on wireless access points, such as droppable relay devices or vehicle-mounted nodes. They are deployed on demand and create multi-hop wireless networks to connect PANs and JANs. Thus, they act as a gateway for mobile first responders that connects them to the network. These access points are deployed at the incident scene and can scale to the magnitude of the emergency. It can be fully built based on 802.11 and Internet standards.

The basic 802.11 infrastructure can be provided by access points integrated into police cars, EMS vehicles, and vehicles of other emergency services. If the range of a network is insufficient, then it can be extended using WLAN repeaters. Such repeaters can be deployed ad-hoc on the ground or in the air using small balloons. According to tests, WLAN can be used up to 80 km/h and should therefore be usable in most scenarios. Nevertheless, although UMTS is usable for speeds above 120 km/h, for example, the transmission rate drops to 144 kbps.
Internet technologies such as Dynamic Host Configuration Protocol (DHCP) allow PSCDs to join the IAN in an ad-hoc manner. When PSCDs contact command centers or access databases on the Internet, the corresponding addresses can be obtained using the standard Domain Name Service (DNS). To find mobile resources provided by other peers in the IAN, discovery can be implemented based on a variety of technologies.

> **R12 (IAN)** - The IAN shall provide the link between the PSCD and the JAN; it must provide resource discovery and message routing. IAN nodes are of non-permanent nature, they shall be portable and offer a dynamic network topology for first responders on-site. The IAN network topology is expected to change during the course of the incident.

JAN: Jurisdiction Area Network JANs form the main communication network by handling the access of IANs to the network and the EAN. They also provide connection to a PSCD if the responsible IAN fails. JANs are of a more permanent nature and includes communication towers. A Jurisdiction Area Network (JAN) is a private network of an agency, e.g., police or an EMS. Through the JAN, the specific command centers can be reached for secure database access, certificate management, task dispatch, and resource mobilization. On the lower layer, these uplinks will utilize various radio technologies, while the higher layers can be fully based on Internet standards.

The vehicles of such agencies provide the base infrastructure for the IAN and radio uplinks into the specific JANs. Such uplinks can be implemented using Terrestrial Trunked Radio (TETRA). If it is necessary to support remote incident sites as well, such as satellite radios, the Iridium technology can be used as a fallback. In addition, it is advisable to utilize the civil cellphone networks, such as GSM, UMTS, or CDMA2000, because they add additional redundancy to the communication system, provide high mobile data rates, and the associated technology is cheap and small. However, because civil networks tend to be overloaded in large accidents, a separate network such as TETRA will always be the first choice.

On the higher layers, such uplinks will be realized using Internet tunnel protocols, such as the Point-to-Point Protocol (PPP), or the Point-to-Point Tunneling Protocol (PPTP).

> **R13 (JAN)** - The JAN shall provide the link between the IAN and the EAN. The devices should be portable, but do not necessarily need to be considered as mobile due to size constraints and power requirements. JAN nodes shall provide a more permanent network for disaster response; its connection topology shall hardly be influenced by movements of relief units. JAN nodes shall be able to mutually connect using the extended area network.

EAN: Extended Area Network The EAN is based on the infrastructure of the Internet. Different agencies can interconnect their individual JANs by using secure Internet tunnels. IANs may also directly connect to the Internet and then use secure Internet tunnels into specific JANs, thereby participating in the EAN.

PSCD: Public Safety Communication Device Each first responder carries a Public Safety Communications Device (PSCD) that is his primary tool for communication with other helpers and the command center. It links the first responder's PAN to IANs and thereby connects him to the network. It also connects the first responder's devices in the PAN, records data from sensors and sends them to the command center. In [HHNL07] these devices are called First Responder's Communication Devices (FRCDs). They record data from the Personal Area Network (PAN) and also provide the link to the higher-order Incident Area Network (IAN). Communication over the IAN allows first responders to convey information to the command center and to receive orders or notifications. Computing devices come in many different form factors and use different input and output methods. However, regarding their user interfaces, there are only two large distinct classes, namely ears & mouth and hands & eyes devices. The former draw their efficiency mainly from the semantic richness developed over millennia (speech, music, etc.) and from the orthogonal usability in many working contexts (hands-free operation); the latter draw their efficiency primarily from the output bandwidth (a picture is worth more than a thousand words) and input intuitivity. Controls may resemble well-trained activities, such as steering, drag-and-drop, or pointing.

The application domain requires the PSCD to be designed as a wearable computer. Because the user interface should only put a minimum of additional cognitive load on the user, neither computers with head-up displays, nor traditional palmtops qualify as good solutions. Using wearables with head-up displays and single-handed keyboards (e.g., Twiddler) puts a very high cognitive load on the user, because computer interfaces using unnatural modalities are employed while the wearer is performing some other, more primary, task in the physical world. An audio-based device with a speech user interface is more natural to use and does not require the level of attention that reading a display or operating a keypad would require. The most important functionality of the PSCD is speech communication with the command center or within peer groups. Consequently, the interaction with the computer should also take place in the same modality. The advances in speech recognition over the past few years also allow for its use in noisy environments, e.g., pilots of the interceptor plane Eurofighter can use voice commands to control the communication systems of their plane.

These considerations indicate that no currently existing wireless networking standard meets the requirements of all possible applications in the emergency response domain. We will have to cope with a heterogeneous landscape of different standards suited for certain applications only. In addition, multiple wireless networks using different standards will be available in the same region of space.

2.6 Summary

In this Chapter we presented the requirements and technology needed in order to build a distributed crisis response communication system.

Table 2.5 presents a summary of elicited requirements, they are divided into the categories ‘Use Cases’, ‘Technological Requirements’ and ‘Design Requirements’. While this list is far from complete, it does effectively limit the number of possible approaches in a reasonable way. These requirements are adjuvant tools for both the design of a first response communication framework and the deduction of verifiable metrics required for evaluation. Table 2.4 already provides a basic quantitative description approach for R1-R6, which allows for derivation of single-/multiscalar evaluation metrics. R8-R13 mainly affects design, and technological requirements. Design requirements are consequently followed in the proposed architecture and technological requirements are modeled in the proposed simulation framework (cf. 4). The coarse design is presented in the next chapter.

Type	**ID**	**Name**	**Implications**
Use Cases	R1	Multicast	Allow for multiple sources, prevent network congestion
	R3	Locality Awareness	Offer Store and Retrieval mechanisms for position informations
	R5	Active Search	Provide key-lookup and exhaustive search functionality
	R6	Hierarchy Maintenance	Provide dynamic command and control structure
Technological Requirements	R2	Shift Change	Resistance against churn
	R4	Resource Awareness	Hardware utilization according to device capabilities
	R7	Communication Channel	Multi purpose digital network
	R9	Self-Sustaining	Do not rely on remaining communication infrastructure
	R10	Information Flow	Allow horizontal and vertical message transmissions
Design Requirements	R8	Interoperability	Common device hardware and software required
	R11	Communication Areas	Expose interfaces for adjacent communication areas
	R12	IAN	Multihop Network Nodes
	R13	JAN	Portable Gateway

Table 2.5: Summary of elicited requirements

Chapter 3

Derivation of Coarse Concept

In this Chapter, the coarse concept for the distributed communication approach of this thesis is developed. We thereby concentrate on the research area of P2P networks, as this is the main focus of this thesis.
In Section 3.1, the design principles and advantages of distributed systems for disaster relief communication are presented. Then, in Section 3.2, principles of P2P systems are recapitulated and building blocks of this thesis are introduced. Finally, in Section 3.3 the proposed architecture is systematically derived from the elicited requirements and the P2P principles.

3.1 Background

In this thesis, we propose a distributed communication approach for first responders following the requirements delineated in chapter 2. In order to classify the applicability (cf. section 2.6) of a distributed P2P system for first response communication, the basic properties of distributed systems need to be revisited.

We use the following definition for a 'Distributed System' [CDK05]:

> A distributed system is one which components located at networked computers communicate and coordinate their actions only by passing messages.

Bal [BST89] sharpens the definition by an additional requirement for computers not sharing primary memory. This definition does not clarify the distinction between a computer network and a distributed system though. The difference to distributed systems can be found in the level of transparency established. Coulouris [CDK05] therefore defines transparency as:

> Transparency is the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components.

In other words the distributed nature of the underlying system is transparent to the user's and application developer's view for distributed systems, which is not the case for computer networks.

Another humorous and alerting quote by Leslie Lamport [Lam03] defines the term as follows:

> A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

This humorous and alerting aspect of the above definition already indicates the presence of additional barriers in distributed programming. The main problems arising in this context are lack of global state, lack of common clock and indeterministic behavior.

In the next section, an introduction to the subclass of distributed networks called P2P networks is presented. P2P networks are highly distributed and naturally face all challenges common to distributed systems. The P2P community developed several approaches and algorithms for designing and maintaining a highly distributed P2P network.

3.2 Introduction to Peer-to-Peer Networks

The term Peer-to-Peer (P2P) in general describes a communication approach of a distributed system. Many Peer-to-Peer definitions are proposed by different researchers. The IETF proposed a definition covering the common aspects of P2P [CI09]:

> We consider a system to be P2P if the elements that form the system share their resources in order to provide the service the system has been designed to provide. The elements in the system both provide services to other elements and request services from other elements.

P2P systems are designed to locate and share resources. The type of resources may vary from processor power, memory, storage to bandwidth or even a combination of all of them. Devices participating in a P2P system are called nodes or peers.

Each node neither acts as plain server nor as plain client; nodes act as the service provider and requester at the same time. There is no distinction between clients and servers [CI09]; this concept is often referred to as the servent concept [S$^+$01]. The first larger scale P2P implementations were file sharing applications like Napster [Inc02], one approach where storage information on data objects was managed centrally, whereas the actual objects were stored within nodes. This server-based management approach makes the concept vulnerable, shutting down the management server makes the network unusable. Kazaa [Smi03] was designed to distribute media data among users and completely decentralized. More P2P applications have followed, and more prominent fields are instant messaging, telephony, distributed computing and live video streaming. A successful application for instant messaging and telephony is Skype [Tho06]; a widely used, open source application for distributed computing is SETI@home [KWA+01], and the Huazhong University of Science and Technology developed the video streaming application PPLive.

Structured and Unstructured P2P Networks Two types of P2P overlay networks are the subject of active research and used in commercial products: structured and unstructured [SW05].

Unstructured P2P Networks One considers a P2P network *unstructured* when the links do not form a predefined topology, but may be chosen randomly. One advantage is that a new peer can easily join the network without establishing a set of mandatory links. The search process in an unstructured P2P overlay is performed by (partially) flooding the network, which we call exhaustive search in the remainder of this thesis. One obvious disadvantage is the often unsatisfying hit rate; the search process creates high network traffic but offers little or no guarantees about finding the object, and short of a byte-for-byte comparison, it is not possible to determine whether two objects are the same. If a node is searching for widely distributed informations it is very likely that the search process will return several successful hits; looking for data located at only a few peers may return an unsuccessful query.

However, many P2P overlay networks are unstructured and scale to several millions of users. The reason might be that unstructured networks are very suitable for human-friendly keyword searches. Note that the search is not limited to simple keywords; any query that can be evaluated locally on a peer is possible.

Structured P2P Networks A *structured* P2P overlay network does create a predefined pattern of node connections, which needs to be actively maintained. Compared to unstructured node arrivals, departures and failures cause considerable maintenance signaling traffic. Nevertheless, structured P2P overlay networks allow fast key-lookup mechanisms using distributed hash tables (DHTs). They hash peer and object identifiers and distribute the hash buckets among the peers. A DHT-specific routing algorithm defines how peers can route through the overlay when they want to retrieve a certain object. Typically, the number of messages needed to locate an object in a DHT grows logarithmically with the number of peers in the system. Thus, DHTs are very efficient for simple key-value lookups (for which they have been designed). Because objects are addressed with their unique names, searching in a DHT is difficult to make efficient [YDRC06, LLH+03, RV03a]. Furthermore, DHTs require the use of (globally) unique object identifiers, typically SHA-1 hashes, which are not very suitable for human users to type.
The hash uniquely determines the location of a peer in the overlay and its neighbors, as well as the placement of content on peers. Current DHTs assume that there is

some out-of-band mechanism for mapping more human-friendly names into object identifiers, but none of them goes into detail about such mappings.
The key-feature of DHTs is the fast lookup process and the high query hit rate; even rare information can be reliably addressed. The query success rate is independent from the number of objects available and therefore equally high for popular as well as rare objects. DHTs have strict rules about how the overlay is formed and where content should be placed in the network. The research world has seen several examples of DHTs [KK03, DZD+03, MNR02, RD01b, I. 01, ZKJ01, RFH+01a, Pla99, MBR03]. They build unique network topologies, which determine large parts of the runtime behavior. For example, a topology may be social network-inspired e.g., Symphony [MBR03]. However, there are many other overlay networks with unique topologies and characteristic runtime behavior.

Search and Lookup As discussed above, unstructured and structured networks have different strengths and weaknesses. The terms ‘Search’ and ‘Lookup’ can be easily misinterpreted, because they both deal with the process of locating information, but they describe a completely different approach.

Exhaustive Search In P2P networks ‘Search’ or often ‘exhaustive search’ describes a mechanism for searching fulltext information. It may be distributed in the P2P network among several peers. Exhaustive Search usually refers to a probabilistic search method, i.e., only a fraction of the available peers will receive and process the query, otherwise the immense message load caused by a search query would seriously decrease the performance of the overlay network. There have been several implementations of search algorithms. Gnutella used a decentralized flooding of queries [SW05]. Kazaa’s method has been found to be very efficient and robust in practice [LKR05]. The BubbleStorm network [TKLB07] is a fully decentralized network based on random graphs and is able to provide efficient exhaustive search with tunable success rates. Irrespective of the search mechanism, the actual content transfer happens directly between the two peers.

Key Lookup ’Key lookup’ or simply ‘lookup’ usually refers to the lookup mechanism used in DHTs. They are very efficient for simple key-value lookups (for which they have been designed), but fulltext search is still an ongoing challenge and usually imposes high signaling traffic on the network. Because the content is placed with hash functions, real search queries are hardly feasible in DHTs. For example, it is not feasible to query a DHT for all objects whose name begins with ‘Foo.’ This would usually require asking *every peer* whether it has any matching objects.
Structured networks, on the other hand, offer very efficient means of looking up known objects, but their ability to perform widely spanning searches is limited.

3.3 Coarse Architecture

Design criteria for incident scene communication networks are derived from the presented requirements (see chapter 2). The approach needs to be scalable, self-organizing and self-recovering to ensure the upkeep of communication flow in case of failures and malfunctions. Users and information flows need to be prioritized according to role and direction [HHNL07]. need a different planning and coordination approach than more severe incidents. Another important attribute is the phase of the incident, starting from the pre-incident phase followed by the incident and post-incident phase. authorities located off-site. especially effective in larger incidents, i.e. the regular communication infrastructure may be damaged or destroyed.

The proposed architecture is inspired by the desirable properties known from distributed systems and especially P2P networks. In particular, the following properties found in the better known P2P overlay networks provide great flexibility for the harsh environment found in the incident and post-incident phase:

- Communication without a central communication/coordination server
- Support of dynamic changes of the infrastructure (e.g., position, load and number of devices)
- Multiple applications on the same communication infrastructure

Consequently, the architecture reserves one distinct layer for P2P overlay networks, which acts as an abstraction layer for the deployed **applications**. Nevertheless, the desired properties known from existing P2P **overlay networks** do not automatically apply to any network environment. Communication devices may fail or might have malfunctions (cp. Section 2.1). Therefore, basic network access and transport, i.e., the underlying **connection topology**, need to be considered as well. Remaining existing communication infrastructure might be used by first responders, but they should not rely on it and should therefore bring their own communication devices to the incident. This hardware is **moved** to the required areas on demand, providing the pre-condition for **network** support. These considerations lead to the basic layered architecture.

Architecture Overview We propose a communication architecture consisting of four layers; the 'Movement and Network Model' layer, the 'Connection Topology' layer, the 'Overlay Network' layer and the 'Application' layer. Of course, a model with more than four layers is imaginable as well; in fact each layer may consist of a specific protocol stack, divided into functional layers of its own. The key benefit of the described four layered approach is that each layer provides a logical, self-contained structure with either an implicit or explicit interface to the adherent layers.
The transition from the 'Movement and Network Model' to the 'Connection Topology' is comparable to the transition between the Open System Interconnection

Reference Model (OSI Reference Model) [Zim80] layer 2 and layer 3.
The link between ‘Connection Topology’ and ‘Overlay Network’ divides the architecture in the transport-oriented and the application-oriented part, comparable to the transition from layer 4 to layers 5-7 in the OSI model.
One previously established approach for the interface between the ‘Overlay Network’ and the ‘Application’ layer is the Key-based Routing API (KBR) [DZD+03].
In the following, the tasks of the four layers are described more precisely:

Movement and Network Model The foundation of the architecture is the Movement and Network Model layer. On the one side, its task is to act as a replay and training environment for former incidents. This offers the possibility to alter the communication strategy under examination and to evaluate new concepts in a clean and safe environment. The higher the chosen level of detail, the more precise are the simulated test results. On the other side, simulated or replayed data of participating organizations are a perfect initial position for both, calculating a network coverage map and an initial connection topology consisting of the set of all valid connections.

Incident Case In the first case, this layer evolves and changes during a first response incident. Shortly after an incident, disaster relief organizations with professional first responders appear on-site and deploy their communication devices. Ideally , they strictly follow the trained course of action and adhere to elaborated guides and regulations. Nevertheless, communication devices tend to fail and larger scale disasters overwhelm the officers on duty. New temporary communication structures appear and disappear naturally and unpredictably. The geographical position, along with the state of all communication paths during the disaster relief process, is the natural occurrence of the Movement and Network Model layer.

Simulation Case The second case, i.e., the simulation environment, acts as a research, replay and training environment for former incidents. This offers the possibility to alter the communication strategy under examination and to evaluate new concepts in a clean and safe environment. Like in a real incident, the simulated network coverage map depends on the position and state of the simulated devices. An initial connection topology consisting of the set of all valid connections is generated for further processing; like in the real environment, position of devices may change and connections may break away.

Connection Topology In the connection topology layer, qualified neighbors from the large number of available nodes are selected. For example, the connection topology maintains a subgraph with dynamic edges of the Movement and Network Model layer. In this layer, neighbor selection protocols, optimization to minimize power consumption, detection of possible network fragmentation, collision handling and packet retransmission protocols are implemented. It is still not possible to address precisely a distinct node or search for a piece of information; these functions are implemented in the next layer.

Overlay Network It is a network of nodes and logical links that is built on top of the existing network topology with the purpose of implementing additional functionalities, the better known of which are key lookup and exhaustive search. Other mechanisms typically implemented in the overlay network are security, multicast and load distribution. Due to dynamic user behavior and network malfunctions, the overlay network must constantly adapt itself. Nodes constantly join and leave the network or simply fail without the possibility of following any protocol. An important task of the overlay is to maintain a safe state of the network. Network growth and shrinkage require maintenance mechanisms; malfunctioning nodes and network outages are challenging for every overlay network.

Application Layer The topmost layer is the application layer. Applications are allowed to access methods provided by the layers below. The application typically uses store, search and lookup functionalities provided by the overlay layer. It is important to build the applications independently from the chosen overlay network, which allows for the running of applications on a variety of overlay networks. While the functionality provided by most overlay networks is essentially the same, each overlay network expects subtly different input parameters. The common Key-based Routing API (KBR) [DZD+03] proposes an interface for DHT networks; it does delegate the routing, delete, update and lookup tasks to the overlay network layer and provides a common interface for all applications. In this thesis, we used KBR for the proposed mechanisms based on DHT overlays. Some tasks are not efficiently covered in the KBR-API: if one wants to search for an item instead of directly addressing it with its key, or if one wants to send a message to a group of people, the KBR-API does not offer an efficient method. We added two methods that are not covered by the KBR-API: exhaustive search and application level multicast (ALM). We are aware that full-text searching and ALM causes considerable signaling traffic on several P2P overlay networks, but instead of restricting the functionality, the consequence should be to provide mechanisms in the overlay network layer to efficiently support both.

Revisiting Requirements Figure 3.1 presents the determining requirement types leading to the proposed architecture. While design requirements pose specifications to all parts of the proposed architecture, the above-described technical requirements mainly affect the 'Movement and Network Model' and the Connection Topology. The identified use cases primarily impact the 'Application' layer and the 'Overlay Network' layer. Nevertheless, the elicited use cases do have indirect consequences for all layers below. The major requirements are described in greater detail in chapter 2.

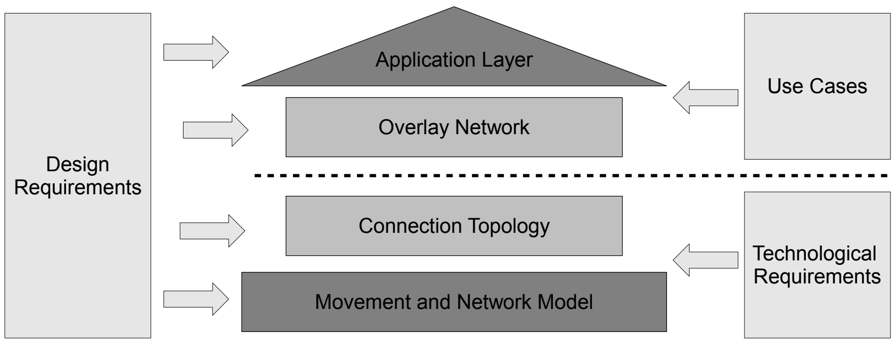


Figure 3.1: Design requirements and coarse software architecture

Especially important for designing the coarse concept are the design requirements and the technical requirements R8-R13 (cf. section 2.6) for the communication approach. By revisiting R8-R11, we conclude:

R8 (Common device hardware and software) This is one of the key elements of P2P technology, and it is designed to work on unreliable commodity hardware and is available for a wide set of operation systems and not restricted to programming languages.

R9 (Self-Sustaining) The ‘Network and Movement Model’, the ‘Connection Topology’ and the ‘Overlay Network’ layers are designed to provide medium access, routing, search, and lookup capabilities. An implementation following this coarse architecture provides all necessary properties for a self-sustaining system (cf. [Zim80]).

R10 (Allow horizontal and vertical message transmissions) The servent concept (cf. section 3.2) does not impose any restrictions on client or server functionalities on a network, and each peer may provide or request a service, even simultaneously.

R11 (Expose interfaces to adjacent communication areas) The adjacent communication areas are the PAN and the EAN network areas. In Paragraph 2.5 we argue that the PAN network is often custom built and therefore consists of highly heterogeneous devices. Consequently a PAN interface needs to be a custom built application of the application layer. This application does have access to all lower layers and might therefore process and relay the information gathered from the PAN.
The EAN network provides access to the Internet and is built on powerful nodes; it

is the task of the ‘Connection Topology’ and the ‘Overlay Network’ to optimally exploit the optional EAN access.

R12 (Multihop network approach) Multihop is one basic building block of P2P technology (one exception is the Napster P2P network); most overlay networks provide mechanisms for multihop searching, lookup and elaborated join and leave procedures. Our approach explicitly provides the ‘Connection Topology’ and the ‘Overlay Network’ layers for multihop functionality.

R13 (Portable Gateway) A JAN tower (cf. Paragraph 2.5) must act as a portable gateway to the EAN. It is the task of the overlay network to make gateways transparent to the available devices.

In addition, the communication approach needs to be able to work in a simulation/training mode. This is included in the architecture in order to allow for design adaptations and excessive tests, thereby minimizing the risk of failures in real incidents.

Chapter 4

Movement and Network Model

The foundation of the proposed communication architecture is the ‘Movement and Network Model,’ and the contribution of this Chapter is a consistent simulation framework capable of combining network and movement simulation. The discrete simulation framework supports feedback loops from the movement simulation to the network simulation and vice versa. It enables a detailed analysis of the reciprocal dependencies of wireless networks and moving peers. In this chapter, a universal description format for handling all relevant settings and actions typical for first response scenarios is presented. A user-friendly movement and environment simulator, which interacts with the network simulation are developed and used for evaluation of the mechanisms proposed in this thesis. The chosen data structure has proven to be well suited for describing settings and actions found in a first response scenario.

The remainder of this Chapter is structured as follows: Section 4.1 revisits the ‘Movement and Network Model’ layer and gives an overview of the proposed simulation approach. Section 4.2 discusses related work. In Section 4.3, the proposed data storage approach for the complete first response communication process is presented. Then, in Section 4.4, the prototype implementation covering the editor and the visualization component is presented. Section 4.5 presents the movement simulation approach and its integration in the storage process. The Section 4.6 addresses our approach of integrating network and movement simulation. A summary of the contributions in this Chapter is provided in section 4.7.

4.1 Conceptual Building Blocks

In the following, we present the proposed approach of the ‘Movement and Network Model’ and revisit the requirements relevant to this layer.

4.1.1 Movement and Network Model Approach

The foundation of the proposed communication architecture is the Movement and Network Model layer. It is used in two seemingly independent settings, during practical disaster relief efforts and in a laboratory simulation environment (cf. Section 3.3). In this thesis, we develop a flexible and extendible approach for the simulation environment.
Its task is to provide a connection topology consisting of the set of all valid connections for further processing, as the device positions may change and connections may break away.

The developed interaction work-flow is used to describe a scenario setup, including dynamic network traffic and peer movement data, as shown in Figure 4.1. Please note that further steps can be implemented at any time. Boxes with rounded edges represent the data structure, whereas the other boxes represent our simulator implementation on top of the data structure. The feedback loop between movement and network simulation is shown as the double-headed arrow between the scenario visualization module and the network simulator.
The simulation starts from the static world model, which describes the disaster settings. The world model is enriched by acquired movement data as well as by simulated network data, which can be added iteratively. Our contribution consists of a generic and extendible scenario description and trace format, which covers everything from the raw definition of terrain and dynamic movement of peers up to a detailed description of network traffic during simulation time.

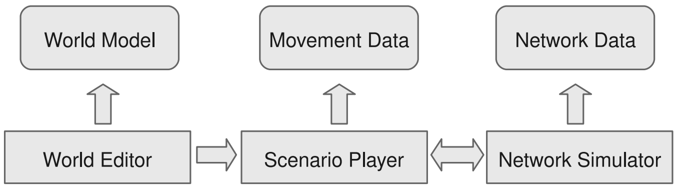


Figure 4.1: Simulation and Data Structure Interaction

A custom built visual editor for scenario layout, peer placement and for setting distinct behaviors for individual peers is used to create scenarios.

4.1.2 Requirements

The main task of the 'Movement and Network Model' is to provide a dynamic connection topology consisting of the set of all valid connections (cf. section 3).
Furthermore, elements requiring access to any information with direct geological and physical relations are considered in the 'Movement and Network Model'. These are especially the requirements R3, R4 and R10.

R3 (Locality Awareness) The simulation approach needs to offer location information for the adjacent layers. The upper layers may provide mechanisms requiring location information, and the topmost layer allows for location-based use cases. In Section 4.3.1, we describe the storage approach for location information, and in Section 4.5 we present how it can be accessed.

R4 (Resource Awareness) In order to allow the overlying layers for resource-aware mechanisms, we propose directly modeling device properties in the 'Movement and Network Model' layer. These properties are exposed in an API for the remaining layers (cf. Section 4.5).

R10 (Allow horizontal and vertical message transmissions) The proposed simulation approach needs to prevent any assumptions on the proposed network topology. The proposed approach is described in Section 4.3.2. The remaining layers must be able to freely adapt to the physical conditions provided by the 'Movement and Network Model' layer.

4.2 Related Work

There are several movement models, that can be plugged into our movement model simulator, and in general, the modeling approaches are usually based on behavior recognition combined with prediction [Ngu] [SG00] [GV06]. Our approach models simple behaviors observed in disaster scenarios, combined with object interaction.

Combination of Movement and Network Simulation Several movement models are implemented in the project BonnMotion [dWG03] and its successor, ANSim [Hel]. In general, these movement simulators do provide scenario files and offer several exchangeable movement models, but they both lack active interaction with environment and they are not able to adapt their behaviors according to the communication on top of the movement model. Nevertheless, by tightly integrating our simulation framework to PlanetSim, we are on the one hand able to set behaviors and active objects within the scenario and, on the other hand, to provide a feedback loop from the overlay simulation to the movement model. Our approach, in contrast to former movement simulators, is not to show the connection probability of two randomly selected nodes, but to show how complete P2P overlays perform. We integrated PlanetSim [Gea05] with our movement framework and, technically speaking, changed the network layer within PlanetSim; due to our interface, other P2P simulators can easily pick up the produced movement patterns. Additional simulators, especially ChunkSim [Kan07] and Peerfact.KOM [Kov07], would greatly benefit from a combination of real world movement and network simulation in order to obtain more realistic simulation results.

The mature simulation engine called GlomoSim [ZBG98] is developed for precisely simulating parts of wireless networks. It does support exchangeable movement models as well. The focus of GlomoSim is to calculate packet reception, radio models, radio propagation and different data link approaches; running complex applications like P2P overlay networks has never been a goal of GlomoSim and is therefore not supported. While GlomoSim does have desirable features, it is built on the programming language Parsec [BMT+98], which does not support current operating systems and is apparently no longer maintained.

Evaluation by Prototyping Especially for first response communication, there already exists a wealth of prototypes, e.g., University of Virginia [A.S03b] identified three main issues in current first response approaches and developed a prototype for a P2P-based first response solution. Further implementation of their P2P solution is done using hypercast, GPS capabilities, multicast streaming video and access control mechanism. Nevertheless, the goal was to develop a prototype implementation and evaluating their approach using our mobility simulation would clarify the suitability for larger scale installations.

4.3 Storage Design

Simulating a first response scenario leads to two distinct challenges.
First, the desired level of detail needs to be tunable and the focus of the simulations must be adaptable. Second, actors need to be deployed into the simulation on demand, and they act as data sources during the simulation and should have individual properties.

We propose an extensible data structure, which is on one hand reasonably fast, accurate and dynamic, and on the other hand, it is still human readable. This section describes the proposed data structure for storing the complete first response process.

4.3.1 World Model

First of all, information describing a first response scenario and information added by the simulation process must be identified. The world model must be able to be process movement and network data later, in order to correctly visualize and analyze the simulated environment.
The setup of a first response scenario needs to consider many different aspects of the environment, first response teams involved, type of catastrophe, terrain, and number and type of affected people. Since every disaster may have individual elements and characteristics, there exists no final set of building blocks for disaster simulation. Consequently. the chosen data structure must be generic enough to be extended for different types of environments and future catastrophes.

Generic Storage A generic storage scheme is developed, which is able to satisfy these requirements, and in addition, it can be extended with new types of peers and environments. The data structure is decoupled from the simulation engine in order to make simulations interchangeable and independent from the currently used simulation engine. Design choices were made in order to store the information very efficiently, as a scenario might grow rather big, containing hundreds or thousands of objects that need to communicate with each other.
The simulation starts with the static description of the environment, which is called the "worldmodel". Since all further incremental steps rely on the worldmodel, it is the root of the XML document. The data that need to be saved at this static stage of the scenario development are the description of the map and the objects placed onto that map, along with settings of their behaviors.

Three Dimensional Storage Model The map information is stored in a grid manner. It contains the size of the map and two layers of map information. The first layer is used to store the terrain of the map. Terrain information might be used later for changing movement behavior or network range and for describing the location and type of the disaster in the first response scenario. The second layer stores height information used to create a 3-dimensional map. The height might also influence speed and network range in the simulation.

Storing this data ineffectively would massively increase the file size, because a grid with a size of 100 to 100 already contains 10000 grid cells with two layers of information (one layer for the grid, one layer for height information). In order to store the scenario efficiently, areas with the same terrain or height are grouped together and saved only once.

The proposed scheme defines an ”object” tag that is used to store all actors. This tag contains all information needed to identify the object and to process it later. The mandatory information stored to identify an object are an ”ID”, object ”type”, and object ”name”.

Table 4.1 types are used within the simulations. Nevertheless, not all of the defined types need to be used in order to run a simulation. Custom types can be added easily. The first step of our simulation workflow may contain a small set of information, and as the simulation process continues, an increasing amount of data enriches the simulation embodiment.
At all stages of the simulation, data can be added to the proposed storage buckets: ‘World Model’, ‘Movement Data’ and ’Network Data’ (cf. section 4.3.1). This first static part produces a file containing object and map information.

Object	**Description**
CPU	Predefined CPU settings per object
Sender, node, receiver	Tags for logging network traffic
Duration	Simulation duration in hh:mm:ss
Point, size	Definition of map areas and positions
Message	Envelope for network messages
Terrain, height	Definition of terrain type and height
Disconnect, connect	Logging of connection state
Waypoints	Contains route information of nodes
Network	Network capabilities, bandwidth, range

Table 4.1: XML objects

4.3.2 Movement Data

Wireless networks are usually modeled using so-called unit disk graphs (UDG) [KMW04]. The construction principle is as follows: Given a number *N* of nodes and a distance threshold *d*, distribute all *N* nodes uniformly at random within an area of a fixed size. Then, all nodes among which the geometric distance is less than *d* are connected by edges. In other words, all nodes which lay close enough to each other are considered neighbors in the resulting wireless ad-hoc network.

This model is valid for wireless networks within a perfect environment. However, in practice, entities usually move in groups, have favorite spots and avoid other areas.

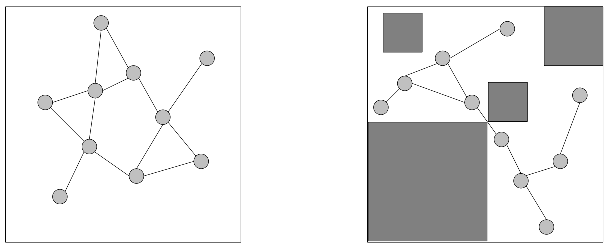


Figure 4.2: Unit disk graph (UDG) and our proposed obstacle augmented disk graph (OADG)

Therefore, we adapted the UDG model by inserting obstacles of different sizes within the area where we place the nodes. These obstacles represent unpopular or unaccessible areas. The generated graphs represent real world, ad-hoc wireless networks more precisely. The approach for more realistic network models is based on the findings of Jardosh [JBRAS03]. We call the network model OADG (obstacle augmented disk graph). Figure 4.2 shows a unit disk graph and its counterpart with obstacles of different sizes.

A similar model is obtained by extending UDGs with weighted edges [WZ04]. If the weight of an edge between two vertices exceeds a certain threshold, a connection cannot be established. In that sense, our generated obstacles are areas with infinite weighted edges, prohibiting any connections. More complex examples can easily be created with the provided user interface of FRCS, which is presented in Section 4.4.1.

After storing static scenario information (see Section 4.3.1), movement data will be stored directly into the objects created in step one. For that purpose, the scheme provides the ”waypoint” tag that contains position data, as well as a time stamp. This waypoint data is the only additional information needed to make the transition between a static first response scenario (before the movement simulation step) and a dynamic first response scenario afterwards. Saving waypoints decouples the storage format from the actual used movement pattern. Descriptions of complex movement patterns are stored without redundancy in the XML file. A scenario might contain hundreds and thousands of moving nodes so only changes of the actual movement destination are saved. Actual positions between two waypoints are interpolated by our simulation engine.

4.3.3 Network Data

The last step in the workflow enriches the XML scheme with network data. The network data are stored in two different locations in the XML file, which reduces redundancy and saves more space because a lot of network information may occur as well. Connects and disconnects will happen regularly in a network with moving nodes, and it is most efficient to save them into the objects themselves. Therefore, little additional information must be stored at this point, only a ”connect” or ”disconnect” tag. This tag contains the ”ID” of the object connecting to or disconnecting from and a timestamp to indicate when this event occurred.
Network messages on the other hand might travel between a lot of nodes before they arrive where they belong. It could potentially accumulate a lot of redundant information to save data into every object they might pass through on their way. Therefore network messages are stored as independent objects similar to movements of peers, though instead of waypoints, network messages contain a list of routing information.

A ”simulatormessage” tag is defined by the XML scheme for that purpose. It contains all the information about the message itself. This includes the ”ID” tag,

and ”content” information as well as all the nodes involved into forwarding this message. There is a ”sender” tag with the ”ID” of the sender object and a time stamp when the message was ”sent”. There is also an optional ”receiver” tag with the ”ID” of the receiver and a time stamp when the message was received. This tag is optional because it might happen, that a message cannot be forwarded to the intended receiver because the node is not connected to the network, and therefore the message might be lost.
Between ”sender” and ”receiver” might be a list of ”nodes”, which were involved into the forwarding process. Every node contains its ”ID” and a ”received” tag. Normally it also contains a ”sent” tag unless the message could not be forwarded in this step.
This separation of concerns enables message storing and tracing in the network without generating any redundant information and maintaining the ability to restore all types of important information, e.g., a message loss or routing delay.
The most important idea behind the XML design was that it standardizes the complete description of a first response process, which contains the world model, the movement data and the network data. Scenario description files are now readable by any simulation engine able to process the proposed XML scheme. This decoupling enables the use of different implementations for every step in the process and the ability to communicate through the XML document. As the structure of the XML file does not change (it can only be extended by additional information), visualization of the data can happen in virtually every step.
The XML scheme was designed from scratch in order to store first response scenarios used in movement and network simulation. Documents are easy and fast to parse and the steps are decoupled to make any implementation effort as simple as possible.

4.4 Simulation Approach

Our contribution is a generic scenario description and trace format and the proposed (cf. figure 4.1) XML workflow covering the whole first response communication process.

In order to show functionality of the XML design and facilitate the development of scenarios, a simulation framework that relies on the proposed XML data structure is developed.

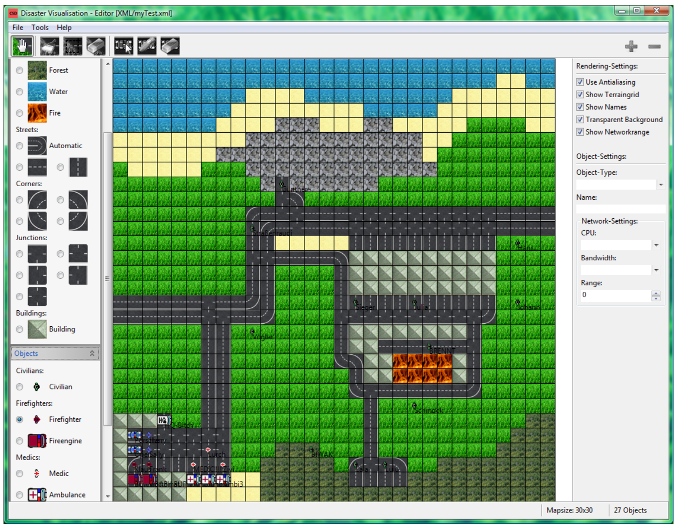
The first module is an editor for creating a disaster scenario. With the editor, the time interval for the scenario, as well as the size of the map, can be specified. Several editor tools are available to help users fill the map with terrain information and to place objects on the map.

From within the editor, the movement simulator can be executed. Its task is to fill all objects in the specified scenario with movement data. Diverse movement models, in order to show different movement possibilities of the objects, are implemented. The movement module is decoupled from the editor and can be used

separately. Different types of movement models can be plugged into the simulation framework. Two different prototypes of movement modules are implemented: ”Random Movement Model” and ”Behavior Based Movement Model”. These models are described in more detail in Section 4.5.
In order to run and visualize the scenario, a visualization module is developed. The visualization module can read complex XML files specified in Section 4.3 in order to visualize terrain information, object information, network data and movement data. The visualization module is therefore designed in the same way as common media players, which allows intuitive interaction with the user.
These modules are developed according to our defined workflow in the disaster scenario XML file. Additional steps for refining the simulation process can be added at any time.

4.4.1 Editor

The first step in the workflow described in Figure 4.1 is to define a world model. This is realized in the editor, where the user creates the environment and the basic layout of the disaster scenario. In this step, only static data are edited. At the end of this phase, the editor launches the movement module in order to fill the scenario with movement data. Thus, the user is able to create a new scenario with only a few clicks. Parameters like map size, start and end time-stamp of the scenario, as well as the default terrain filling can be specified in the editor as well. Figure 4.3 shows the GUI of the editor with a simple example of a disaster scenario.
Storing and loading of a scenario are implemented according to the XML design


Disaster Visualisation - Editor [XML/myTest.xml]

Rendering-Settings:
- [✓] Use Antialiasing
- [✓] Show Terraingrid
- [✓] Show Names
- [✓] Transparent Background
- [✓] Show Networkrange

Object-Settings:
Object-Type:
Name:
Network-Settings:
CPU:
Bandwidth:
Range: 0

Mapsize: 30x30 | 27 Objects

Figure 4.3: Screenshot of the editor

specified in Chapter 4.3. Different types of terrain information are implemented in order to analyze a variety of different disasters. Most of the common types of

Name	**Mobility**	**Special attributes**
Civilian	P	fear, helpfulness, health, knowledge
Civilian car	C	terrain specific behavior
Firefighter	P	helpfulness, health, knowledge
Fire engine	C	terrain specific behavior
Medic	P	helpfulness, health, knowledge
Ambulance	C	terrain specific behavior
Policeman	P	helpfulness, health, knowledge
Police car	C	terrain specific behavior
Mobile headquarter	I	CPU, network range,terrain specific behavior

Table 4.2: Peer Information, C=supports additional properties of a car engine, P=supports properties for pedestrians, I=immobile

terrain that occur in the real world are implemented. The user can fill the map with several implemented types of terrain information, such as grass, forest, sand, stones, streets, buildings, water and fire. In our example, the ”fire” represents the disaster location. However, it is also possible to describe other disaster scenarios by adding more terrain information to the program. It is possible to extend the terrain model with new types of terrain, like ”Collapsed Building” or ”Flooded Area” in order to simulate other types of disasters.

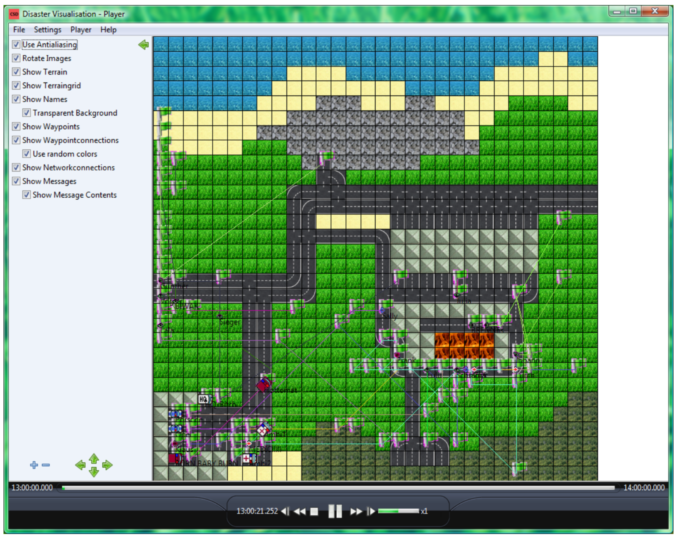
In order to modify terrain information, three different tools are implemented. The ”Terrain-Brush-Tool” lets the user fill single cells of the map with the selected terrain. With the ”Terrain-Rectangle-Tool” the user can draw complete rectangles on the map and, by releasing the mouse, the drawn rectangle is filled with the selected terrain. To delete the terrain information a ”Terrain-Eraser-Tool” is implemented. It sets the terrain information to ”No Terrain” for the selected cells.
Besides the terrain information, the user can place different objects onto the map. Table 4.2 shows a selection of available objects, with some selected attributes. Further attributes can be added easily and thus influence the behavior of the objects. These predefined objects represent a basic set relevant to a disaster scenario. Similar to the terrain information, the program can easily be extended to handle more and other types of objects because the objects are decoupled in code. Every object stores information about the terrrain where it is located, the speed it can move, etc.

Three different tools to modify the objects in the created disaster scenario are developed, analogical to those for the terrain. The ”Object-Placement-Tool” lets the user place the selected object onto the map. Sanity checks will prohibit placing several objects on one cell and placing objects on improper terrain. With the ”Object-Selection-Tool” the user is able to select and modify objects already placed. Their name, CPU, bandwidth, network range and even the object type can be changed. The ”Object-Eraser-Tool” lets the user remove objects from the scenario that are

not longer needed.
After creating and modifying a disaster scenario, the user can save this scenario to a XML file. Loading the disaster scenario is also possible. In addition, the user may start a movement calculation in order to fill the objects in the created disaster scenario with movement data. The current prototype allows selection of the ”Random Movement Model” and ”Behavior-Based Movement Model”. Other movement models can be added easily by extending the movement module. The editor executes the movement module with the selected movement model and calculates the movement of the objects in the scenario. The modified disaster scenario is then automatically saved to a XML file in order to store the adaptations, including the new movement data of every object in the scenario.

4.4.2 Dynamic Visualization

In order to visualize a complex disaster scenario, a dynamic visualization module is implemented. It can load disaster scenarios created by the editor. Figure 4.4 shows a screenshot of the module. It is designed to match the usability of any common media player, so it is easy to use.



[✓] Use Antialiasing
[✓] Rotate Images
[✓] Show Terrain
[✓] Show Terraingrid
[✓] Show Names
[✓] Transparent Background
[✓] Show Waypoints
[✓] Show Waypointconnections
[✓] Use random colors
[✓] Show Networkconnections
[✓] Show Messages
[✓] Show Message Contents

Figure 4.4: The dynamic visualization prototype

The dynamic visualization module consists of three areas with which the user can interact. The biggest part of the module is the ”Visualization Panel”. The disaster scenario is rendered on the main panel and played showing the simulated first response scenario from the chosen XML file. The user can interact with this panel by clicking and dragging. That way, the user can zoom in and out or pan the current sector. On the left is a sidebar with more options. Here, the user can select what should be rendered. It is possible to show the waypoints the objects are moving to, as well as connections between these waypoints. The network connections between the different objects and the messages they send to each other can be visualized as well. On the bottom of the proposed module are the basic control elements of the module. As in every common media player application, there is a time bar to jump directly to a specific time in the past or in the future. Normally the proposed module visualizes the scenario in real time, but the simulation speed can be modified by seeking forward or backward, or simply by slowing the animation down to see what happens in slow motion. The animation can be also paused or stopped. The visualization module supports an API in order to enable different kinds of network simulators. The network simulator fills the objects in the disaster scenario with network data. This allows objects to connect to each other, as well as send and receive messages. That API is implemented in such a way, allows the network simulator to indirectly influence the movement of the objects and vice versa. This interface implementation is described in Section 4.6.

4.5 Movement Simulator

The second part of the process after designing the first response scenario is the movement simulation.
The visualization module is able to process any scenario independently from the chosen movement model. The separation of movement creation and movement storage allows for plugin in new movement generators without the need to change the storage format. It is important for the simulator, so it can be used as a stand-alone product. Every implementation of a first response scenario using the XML scheme provided would be able to use the simulator right away or with minor adaptations.

Input API The input API works in a straightforward manner because it needs only one hook for the editor in order to load first response scenarios into the Simulator. However, because the Simulator is already able to handle scenario objects that are the result of parsing the XML input this is easy to achieve. No matter if the generic XML input is used or the API is called from the editor, it is handled by the same method of ”createXMLwithWayPoints” provided by the MobilityHelper class. This hook enables loading any movement model with any first response scenario, either from a file or directly by the API, and writing it to an output file that contains the scenario enriched with waypoint information. This movement simulation architecture is able to handle different types of movement models. The only requirement is to implement the abstract Mobility class in order to provide all methods needed for the output API. This output API is described in a separate section, Section 4.6, because of the crucial influence on the simulation process. There are two different movement models already implemented, which generate waypoints and make use of the simulator architecture and the information provided by the XML input. In order to show the generality of the simulator, two extremely different types of movement models were chosen.

Random Waypoint Model The first model implemented is a pure ”Random Waypoint Model”. At every step it will generate a new random waypoint. This random waypoint will be checked against all types of constrains, e.g., whether the object can stand on the terrain, whether there is another object already on the waypoint or moving towards this waypoint. Even with all the constraints, this movement model remains very simple. It generates waypoints for a network simulation step, and therefore, benchmarking of different network protocols against each other is possible.

Behavior Based Movement Model The second model implemented adopts a completely different path. Instead of using random behavior, it is a behavior-based movement model. This means it assigns some kind of behavior to each object that will influence the movement in such a way that the result is a human-like movement pattern. In every step, it will evaluate the current situation and recalculate the current target waypoint for every object and thereby making it able to react to changes in the environment.

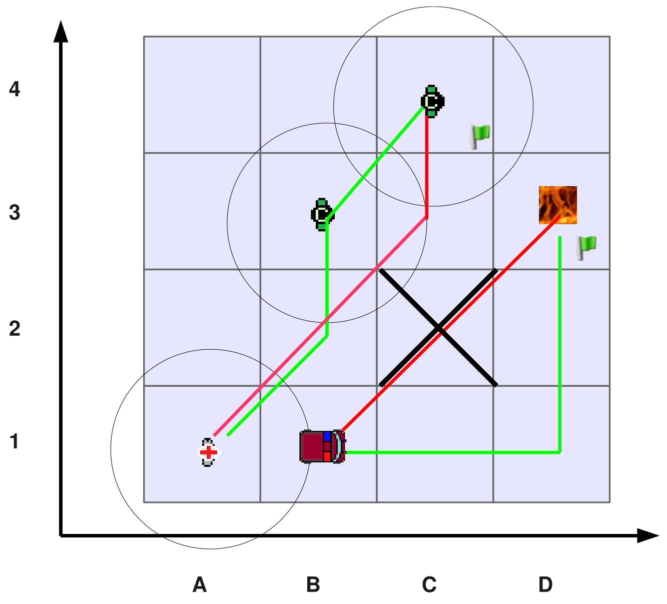


Figure 4.5: Behavior based movement calculation

The behavior-based movement model may be configured freely and supports the triggering of actions when using the properties shown in table 4.3.

While every object uses its behavior to find its target, a shortest path algorithm based on the actual map knowledge will provide the object with its next waypoint. The target itself is chosen in a human-like (behavior based) process, and the target trajectory is based on incomplete information, which simulates human-behavior as

Name	Description
Position	Current position
Waypoint	Next planned stop of object
Line of Sight (LoS)	Visual range of sight
Target	Larger goal, consisting of several waypoints
Helpfulness	Describes the possibility of detours for spontaneous targets in LoS
Fear	Probability of discarding waypoints and targets
Health	Determines movement speed
Shortest Path	Calculate optimal new route to a target
Message	Change target according to received message
Follow	Probability of following someone to his waypoint

Table 4.3: Properties of behavior-based movement model

well. More models can easily be developed using the properties shown in table 4.3 to trigger movement actions. Additional properties can of course be implemented as well.

The behavior-based movement model is designed to make the input for the network simulator as realistic as possible by approximating what a human would do in every step. Therefore one might be able to evaluate network protocols in a much more realistic way and find flaws or advantages that were not that obvious before.

Behavior-Based Movement Model Scenario Figure 4.5 shows an example of the behavior-based influence on the movement model. This example shows two distinct movement behaviors. An ambulance (B1) wants to directly drive to the disaster origin (D3) by passing C2. An obstacle prevents the ambulance from passing through C2, and therefore, it is driving a detour to reach its destination. The second example shows a medic (see A1) who has the task of caring for a casualty in C4; the line from A1 over C3 to C4 marks the chosen way for the task. The medic notices on his way that another casualty located in B3 needs medical treatment. He will stop at B3 and continues later on his way to C4. His tasks, as well as the chosen waypoints, are adapted at runtime depending on the actual environment. While the behavior itself provides every object with a goal, e.g., "rescue people" for an ambulance, a characteristic mix of attributes for every object and the situation it is facing, aside from factors like range of sight, will influence every object in a unique manner.

Both movement models show that the architecture itself can be used to implement different types of model approaches and that information provided by the world model is enough for any movement model to employ.

4.6 P2P Simulator Integration

As shown in Figure 4.1, the XML file is consecutively filled with network data. The XML scheme was designed in such a way, that the separated modules described in Section 4.4 can read the given information and add new information in every step of the simulation. A network simulator can read the XML file in order to obtain information regarding the disaster scenario, the objects and their movements. With this information, different P2P networks or other network infrastructures can be simulated and the information can be written back into the XML file. That way, the different network structures can be tested and compared to each other.

Object Access The network simulator accesses informations about the different objects and their movements. Every network capable object has the settings of ”CPU”, ”bandwidth” and ”range” to specify the network capabilities. Settings like ”CPU” and ”bandwidth” are held purposely in a relative way. There are three distinct settings for them: ”small”, ”medium”, and ”large”. That way, the network simulator can occupy every setting with an absolute value of its own. Thus different network devices can be tested and compared to each other. The setting ”range” is an absolute value. It describes the radius in grid cells on the map, taking the objects position as middle point. That means all objects in this circular range can be reached and receive messages.
The next section discusses the integration of the disaster scenario saved in a XML file into any network simulator.

Reciprocal Interaction Objects like rescue units react on the messages they receive. So the movement is linked directly to the network events. For instance, a firefighter might find a victim and calls an ambulance or a medic. After receiving the distress call, one or more idle medical units would move to the designated coordinates to help the victim. It is obvious that an interaction between movement module and network simulator is necessary.
In order to cover this case, an API for the movement module is developed to synchronize the network simulator with the movement module in such a way that every step in the movement model simulator influences every step in the network simulator, and vice versa. This creates more realistic movement patterns by using more information about the network and lets objects move in a more network-friendly or even less network-friendly way. Infintite feedback loops are theoretically possible, but did not occur so far. The last order is stored and prohibit iterating between two concurrent movement goals. In order to test P2P protocols handling worst case or best case scenarios, as well as distinct numbers of mobile nodes and

data, the twofold simulation process delivers both, a network environment as well as the according movement simulation.

Integration Approach As our input architecture is stepwise enhancing the data structure, this approach is retained for the output structure as well. First of all, we implemented a factory in order to switch between different mobility models. Therefore, any mobility model that is created by the factory has to implement the `mobility` interface. This interface can work with the disaster `scenario` object that is created using our editor. The network simulator can then interact with both objects, the `mobility` object that contains the desired movement model and the disaster `scenario,`to create more realistic behavior. Thus, the network simulator influences the movement of the nodes in every step and also reacts to the movement that is inflicted by this influence. Figure 4.6 shows how the API is built and the connection interfaces in order to remotely trigger the movement simulation.

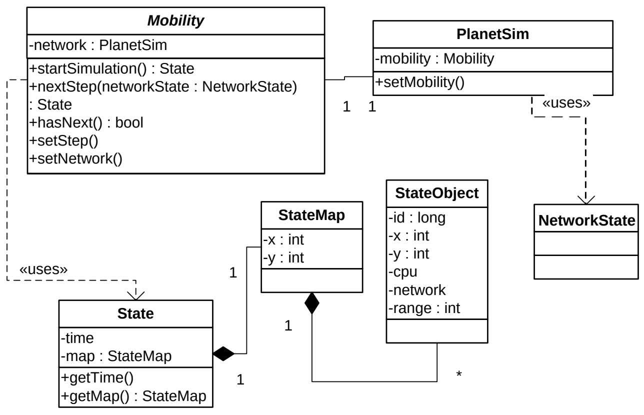


Figure 4.6: The Simulator integration

The new approach of the output API implemented in the movement simulator enables a combined movement and network simulation process, i.e., both simulation parts support feedback loops from one another. Since "PlanetSim" is used as a network simulator and is based on discrete steps, this API makes it possible to start the movement simulation and synchronize the step size. The step size is a time in milliseconds. Every time the method `nextStep(networkState)`is called, the movement simulator will add the step size to the current time. Nonetheless, it is also possible to jump to a certain timestamp by calling the method `setStep(long step)`. The movement simulator will then set the timer to the designated time given by the argument "step" ($time = step * stepSize$).
In every simulation step, the network simulator passes its network state to the movement simulator. That way, the movement simulator can react to network events that can occur between the network nodes. This is how the network simulator can

directly influence the movement of the nodes. It can pass commands to the nodes but the movement simulator has to decide whether to obey the order or to continue the current task, or even whether to dedicate a new task to nodes.
In every simulation step, the movement simulator will return a `state`object. This enables the movement simulator to influence the network simulation process. The `state` object consists of the current time and a `StateMap`, where all disaster objects are saved. The `stateMap`is constructed like the terrain map and is divided in grid cells. Every disaster object (`StateObject`) is saved in one cell of the map. The network simulator can use this information to calculate whether the nodes are in range of each other or not. It can connect the nodes with each other and let them send messages to one another. In order to identify every object the `state`object carries an ID that never changes throughout the simulation process. The position as well as the network information (”CPU”, ”bandwidth”, and ”range”) are saved in the `StateObject`as well. Otherwise, the network simulator would not be able to make decisions about the network behavior of the nodes.

This API was implemented in order to be able to extend any network simulator to interact with our movement simulator. It is possible to test different simulators and different network structures. Different movement models can be compared to each other using different network infrastructures and simulations as well.

4.7 Summary

Our contribution consists of a generic and extendible scenario description and trace format, which covers everything from the raw definition of terrain and dynamic movement of peers up to a detailed description of network traffic during simulation time.
We strictly designed the ‘Movement and Network Model’ to satisfy the requirements elicited in Chapter 2. In particular, the requirements R3, R4 and R10 (cf. Section 4.1.2) significantly determine the development of this simulation approach.

Common P2P network simulators, for example PeerfactSim.KOM [KKH$^+$06], Chunksim [Kan07], PlanetSim [Gea05], and Omnet++ [V$^+$01] are designed to conduct larger scale simulations and provide probabilistic models to simulate node churn. However, these network simulators assume a working Internet topology, movement simulations and mobile devices are not supported. Consequently these simulators are not considered in the direct comparison with FRCS. Nevertheless, we provide an interface which allows for integration of a P2P network simulator with FRCS, thus simulations can either be performed assuming an Internet topology or using the movement approach of FRCS (see Section 4.6).
Table 4.4 shows FRCS in comparison with GlomoSim and ANSim. By tightly integrating movement and network simulation, we are able to build very fine grained simulation models. This does allow for support of the reciprocal dependencies of wireless networks and moving peers. Our description model proved to be adequate

for the needs of simulating a first response process, from the beginning of the catastrophe until the completion of all tasks of the first response team. By combining movement and network simulation more detailed communication simulations are now feasible (cf. Figure 4.1). We are not aware of any other simulation framework supporting reciprocal dependencies between both types of simulations in one integrated framework.

Feature	**ANSim**	**GlomoSim**	**FRCS**
Scenario Support	●	●	●
Reciprocal Interaction	○	○	●
Behavior Support	○	○	●
Visualization of Scenario	○	●	●

Table 4.4: Comparison of FRCS with better known network simulators supporting mobility

In order to test the flexibility of our description format, we built various scenarios combined with two selected movement models (random waypoint and behavior triggered) and simulated active network traffic on top. The intuitive possibility of mix-and-match scenario, movement model and communication topology convinced us to further investigate the extension of the first response communication sandbox.

Using the movement patterns and the simulated communication devices, a graph containing all devices with their possible neighbors within range is calculated. This graph changes its shape during the conducted simulations, because devices move and new devices might be available. One challenge is to prevent the communication network from being partitioned. A mechanism for detection of articulation points, i.e., critical nodes, that keep the network connected, is presented in the next chapter.

Chapter 5

Connection Topology

The connection topology provides basic network access and is the precondition for a functional 'Overlay Network' layer. A robust connection topology is crucial to the whole communication approach. However, in the disaster's incident phase, communication devices carried by disaster relief personnel tend to be unevenly distributed over space (cf. Section 5.1). Several dense clusters of nodes are connected by a few important nodes in sparsely occupied areas. Removing these articulation points would partition the network and severely reduce or destroy the overall connectivity.

Our contribution in the 'Connection Topology' layer is the novel approach, called BridgeFinder. It identifies the critical articulation points, operates as a fully distributed algorithm and causes only very little messaging overhead. It is compliant to the Push-Sum gossiping protocol and is significantly faster and more precise than existing mechanisms.

The remainder of this Section is structured as follows:
In Section 5.1 we revisit necessary preliminary mechanisms and requirements. Section 5.2 discusses related work. In Section 5.3 we describe the application domain for BridgeFinder. The proposed approach is developed in Section 5.4. A theoretical model for critical peers is derived from classic graph properties in Section 5.5. Runtime observations of speed are conducted in Section 5.6. Section 5.7 proposes a practical approach for implementing BridgeFinder. In Section 5.8 we evaluate our approach. Section 5.9 includes a discussion of possible damage by malfunctioning devices or attacks. Section 5.10 concludes the Chapter.

5.1 Introduction

In this chapter, we present BridgeFinder, a distributed mechanism for calculating graph measures, which until now have only been feasible with centralized algorithms using global knowledge. BridgeFinder is fully decentralized, based on gossiping, and only requires local knowledge. Our evaluation shows that BridgeFinder is able to identify the critical nodes very efficiently. We also show how BridgeFinder

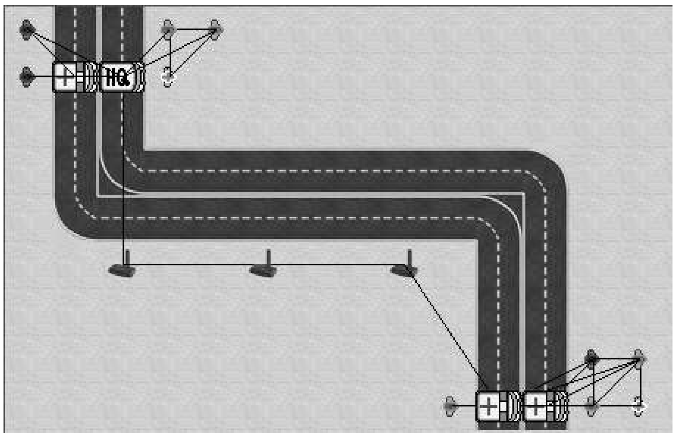


Figure 5.1: First response ad-hoc wireless networks.

can be protected against malicious nodes attempting to skew the process, and how BridgeFinder can easily be implemented with IPv6.

5.1.1 General Approach

Our simulations of disaster scenarios with FRCS (cf. Chapter 4) create sparse and irregular communication graphs, and similar topologies have been observed in real-world installations[MM07] [MM09]. Communication devices carried by disaster relief personnel tend to be unevenly distributed over space. Several dense clusters of nodes are connected by a few nodes in sparsely occupied areas. Removing vital nodes along such *bridges* would partition the network and severely reduce the overall connectivity. Consequently, detecting and protecting those few vital nodes is crucial for keeping the network operational.

Figure 5.1 shows an example of a wireless multihop network within a first response situation. Near the headquarter on the left side of the figure, and the recovery location on the right side, there are dense communication clusters. These clusters, however, are only sparsely interconnected. The challenge for establishing reliable communication is to identify exactly those critical paths among clusters and secure them by providing strong access points and rerouting messages in order to prevent congestion. Failing to protect these paths could lead to network partitioning and the inability of the headquarters to communicate with the personnel on the recovery location.

Standard approaches for identifying those vital links require global network knowledge, which is unavailable in mobile ad-hoc networks. Even if it were available, one would require $O(V^3)$ running time, where V is the number of nodes within the network (by using Floyd-Warshall, for example). Obviously, such approaches have extremely limited practical use.

In order to overcome these challenges, we developed BridgeFinder. It leverages the recently developed Push-Sum gossiping algorithm [KDG03]. BridgeFinder identifies critical paths between densely connected clusters without any global knowledge. Our results show that it has very modest running time and message complexity. Furthermore, BridgeFinder can be integrated in the regular maintenance and application traffic and therefore produces almost no message overhead.

5.1.2 Theoretical Foundations

In order to show how BridgeFinder efficiently identifies the critical nodes, we require several theoretical foundations:

Gossiping Gossiping was first proposed in the context of distributed database systems [DGH+87]. The developed mechanisms were called 'epidemic-style' or 'gossip' update algorithms. Databases did periodically check their state and used a randomized method for selecting their gossip target to synchronize their state. The basic epidemic protocol is called anti-entropy and requires two steps.
In step one, every node chooses another node at random for content exchange. In the second step, during the exchange, each node resolves the differences and both agree on a common state (cf. [DGH+87]).
In contrast to the classical network flooding, gossiping uses recurring bilateral communication with individual message content. Database systems employed these protocols using a low exchange frequency, while P2P systems use gossiping at higher speed.

Push-Sum Our proposed approach leverages the recently developed gossiping approach called Push-Sum [KKD04][TLB07]. The main purpose of the Push-Sum algorithm is to estimate the number of nodes within a network. Its basic idea is as follows: Imagine we want to measure the volume of a lake and we neither know the shape its bottom follows, nor how deep it is. What we do is to release a school of fish into the lake. Assuming that they spread out uniformly, we simply have to wait long enough for the fish to diffuse within the lake. Then, we remove a gallon of water from the lake and count how many fishes are in there. Knowing the original number of fishes and the number of fishes per gallon of water, we can calculate a very good estimate of the lake's volume.[1]

Push-Sum works as follows: initially, every peer manages some lake region. Then a peer starts the algorithm by picking up a number of fishes and distributing them uniformly among its neighbors and itself. Afterwards, when gossiping, each peer distributes fish uniformly among its own region and those of its neighbors. The algorithm is finished when all peers have the same amount of water and fish, according to some error tolerance. Knowing the current local amount and the fixed

[1]Analogy taken from [TKLB07]

original amount of fish distributed within the network, each node can compute the number of peers in the network.

Convergence Speed The convergence speed of the Push-Sum and therefore the BridgeFinder algorithm is determined by the diffusion speed of the underlying network. The diffusion speed characterizes how fast a value starting from an arbitrary node diffuses through the whole network. In graphs with high expansion, the convergence speed of Push-Sum has been proven to be $O(\log N + \log \frac{1}{\epsilon})$ [TLB07], where N is the number of nodes and ϵ, the relative convergence error. For geometric networks with "roughly" a uniform distribution of nodes in the Euclidian space the diffusion speed is shown to be $O(\log^{1+\epsilon} d)$ [KKD04], where d is the diameter of the network.

However, many complex networks, especially ad-hoc networks, have different topology than the networks described above. They consist of densely connected clusters that are only sparsely interconnected. An O-notation solution for the running time of Push-Sum in such networks is only available for special metric spaces. A general proof is not yet available [KKD04]. Both their irregular structure and varying number of clusters make it very unlikely that a general solution for ad-hoc networks could be derived. Therefore, we concentrated on numerical simulations to estimate how Push-Sum really performs on those networks. For this purpose, we used the OADG model (see Chapter 4).

Critical Peers To the best of our knowledge, two categories of critical peers are known: global and local [SLS06]. If a global critical peer fails, the network is broken into two or more components. If a local critical peer fails, this peer, along with its neighbors, are isolated from the network. The rest of the network remains intact.

There is a variety of specialized algorithms for detecting the second type of peers [LXKL06] [SLS06] [WZ04]. Depending on the accuracy of their results, they produce different computational overhead. Still, all of them require only the neighbor list of each peer and no further knowledge about the network.

Global critical peers are much more important. Their failure affects the function of the entire network. However, there are no local and therefore no efficient algorithms for detecting such peers.

Centrality Measures Before we explore the implementation details of BridgeFinder, we need to revisit two centrality measures, called betweenness centrality and closeness from graph theory. They describe the specific role that critical peers play in supporting communication flow.

Betweenness Centrality The first important measure is *betweenness centrality*, or simply betweenness [Fre77][Bra01]. Nodes that occur on many of the shortest paths between other nodes in the network have higher betweenness. Thus, the betweenness of a node is proportional to the number of shortest paths going through this node. Consider a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges connecting them. Then, the betweenness $C_B(v)$ of $v \in V$ is given by:

$$C_B(v) := \sum_{s \neq v \neq t, s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{5.1}$$

where s, $t \in V$ and $\sigma_{st}(v)$ is the number of all shortest paths from s to t going through v and σ_{st} the number of all shortest paths between s and t. The higher the betweenness of a given node, the more important this node is for communication within the network.

It is easy to see that betweenness is a very convenient measure for detecting critical nodes within a network. Unfortunately, it is very expensive to compute. One must to compute all shortest paths between all node pairs in the network and keep track of the nodes on those paths. Some global network knowledge, as the number of nodes in the network, is also required but in most cases unavailable in wireless ad-hoc networks.

Closeness Centrality Another important measure is *closeness centrality* [Sab66] [Dan06], here referred to simply as closeness. Using closeness, we can identify those nodes within a network, that are responsible for rapid communication flow. Again, consider a graph representation $G = (V, E)$. The closeness of a node v is defined as the mean geodesic distance (i.e., the shortest path) between a node v and all other nodes reachable from it:

$$C_c(v) := \frac{1}{n-1} \sum_{t \in N \setminus v} d_G(v, t) \tag{5.2}$$

where $d_G(v, t)$ is the geodesic distance between v, and t in G and n is the size of the connected component N reachable from v. Closeness is a measure of how long it will take for a particular piece of information to spread from a given node to all other reachable nodes. Unfortunately, to compute the closeness of each node, one again must to compute the shortest path between each two nodes in the network. Therefore, closeness is also both: very expensive to compute and requires global knowledge about the network.

5.1.3 Requirements

The main task of the ‘Connection Topology’ is to provide basic routing and network connections(cf. Section 3) for the ‘Overlay Network’ layer.

In our approach, we focus on identifying critical links, as their failure may destroy the connection topology and partition the network. A network consisting of several isolated parts is not usable in a reasonable way, and therefore, this mechanism can be seen as a key enabler for the elicited use cases (see Section 2.2).

However, unbalanced network topologies are created automatically by manually placing network devices in the field. Our approach, considers the device types listed in requirements **R12** and **R13** (cf. Section 2.2) and is especially developed for imbalanced network topologies (cp. Chapter 4).

The device sending range (IAN and JAN) and geological position may change during the incident phase. These dynamic properties need to be supported by the mechanism to identify 'critical nodes' as well.

5.2 Related Work

Our main target environments are wireless multihop networks. One example of an application developed for such environments is Groove [Gro04]. A wireless network environment is a challenge for such distributed applications. Due to many unpredictable factors, established connections or even nodes may suddenly disappear. As communication flow depends on network connectivity, it is crucial to identify and protect critical peers within the network. Those are the peers whose failure leads to malfunction or the complete breakdown of the network.
To the best of our knowledge, BridgeFinder is the first approach using properties of the gossiping mechanism to detect critical peers.

Gossiping Gossiping is used in many network applications. One of them is the Push-Sum algorithm [KDG03]. Its original epidemic protocol is called anti-entropy and is used for database replication [DGH+87]. Another application of the Push-Sum algorithm is counting peers in P2P overlay networks [TKLB07]. An overview of actual epidemic protocols is given by Eugster [EGKM04].

Our contribution, BridgeFinder leverages the Push-Sum-based algorithm proposed in [TLB07]. Our approach enables BridgeFinder to detect critical peers in a distributed manner; its original purpose remains fully functional. Critical peers have a large impact on the connectivity of a network. The absence of even only a small number of those nodes may successfully render the network unusable [SGG03].

Critical Peers There is a large body of recent research dedicated to identifying critical peers. The common approach is to send probe messages from a node, for example from C to other nodes in the network. When a node receives a probe message, it must send it back to C. C contacts iteratively a chosen set of nodes [QCC07] or simply floods the network [LXKL06]. After all probe messages have returned back to C, it computes how fast the single messages have returned. Based on its results, C decides if it is a critical node or not. Short return times mean C is critical; otherwise it is not critical. After the procedure is finished, the computed results are known only to C.

Another approach is the detection algorithm based on the Midpoint Coverage Circle (DMCC) [SLS06]. It also has a simple workflow. Assume that the node C wants to estimate whether it is a critical node. It chooses a random pair of nodes within the network and asks them to contact each other. If the message exchanged by the two nodes on its way goes through C, then C considers itself critical and not otherwise. The computational overhead for C is smaller than the one in the previous approach. However, in order to guarantee reliable results, C has to test a very large number of node pairs. This produces a large message overhead for testing just a single node. The acquired information is still known only by C.

5.3 Application Domain

There are two requirements the underlying network has to fulfill in order for BridgeFinder to function. First, a node must be able to communicate with some other nodes in the network, at least with all of its neighbors. Second, the links in the network must be undirected. In general, BridgeFinder can operate on any network that fulfills these two criteria. Both requirements are satisfied by the underlying 'Movement and Network Model' layer.

BridgeFinder detects critical peers, that is, nodes crucial for communication within the network. Therefore, the fewer the nodes on which communication depends, the higher the benefit of using our approach. Compared to algorithms with global knowledge in static networks, BridgeFinder is less accurate than standard approaches for detecting critical nodes. *However*, its most significant advantage is that it requires no global knowledge and can operate on continuously changing underlying networks. That makes it very attractive when working with wireless ad-hoc networks.

We have simulated wireless ad-hoc networks found especially in first response situations. These types of networks are challenging because global knowledge of the network is usually unavailable and moving peers are continuously altering the network. As in first response situations, in many other use cases of ad-hoc wireless networks, clusters with sparse interconnections arise naturally and need to be identified in order to protect the network connectivity.

Networks may become partitioned in case of a crash by the critical peers. Our approach identifies peers, that keep the network together. Once the critical peers are known, partitioning can be avoided by establishing additional links among susceptible clusters connected by those critical peers.

In any complex network, maintaining information flow is equal to the critical nodes remaining operational. The higher the importance of a node, the higher the impact of removing that node.

5.4 The BridgeFinder Algorithm

BridgeFinder relies on gossiping to identify the critical nodes (cf. Section 5.1.2). The core algorithm of BridgeFinder operates just as Push-Sum with a small extension: each peer, aside from gossiping the currently possessed amount of water and

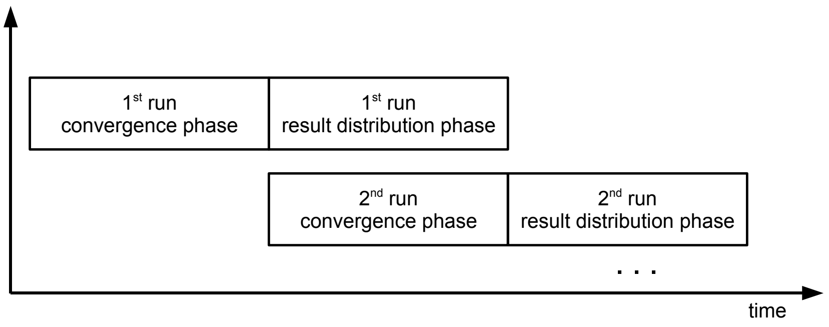


Figure 5.2: Phases of the BridgeFinder algorithm and interleaving among multiple runs

fish, also sends along how many times it has already exchanged values with its neighbors to achieve that state. The following is a pseudocode of the core procedures of BridgeFinder:

```
currentEstimate = my_water / my_fish;

Periodically:
    my_fish = my_fish / (my_degree + 1);
    my_water = my_water / (my_degree + 1);
    iterations = iterations + 1;
    sendToAllNeighbors(my_fish, my_water);

Receive(fish, water):
    my_fish = my_fish + fish;
    my_water = my_water + water;
```

Each node keeps track of the number of exchange operations with its neighbors until it has converged. Overall, in our simulation, we set $\epsilon = 0.02$. Thus, if the current exchanged fish and water values do not change the previous values by more than 2%, a node sets its state to converged.

After this *convergence* phase, the result is distributed in the *result distribution* phase. Figure 5.2 shows how these phases are interleaved among multiple runs of the algorithm. The results of the first run are distributed in parallel to the convergence phase of the second run, and so on. The BridgeFinder algorithm exchanges data between peers solely by piggybacking data on the normal network traffic.

Each node keeps a list of the 10 fastest converging nodes and the number of exchange steps they required to converge. When a node v has converged, it checks whether it has not been faster than some of the nodes in its top list. If this is the case, v sends its next exchange messages to its neighbors, as well as the information that it has converged and that it qualifies for the top 10 nodes list. Assume that a node w receives a message that v has qualified for the top 10 list. Then, if not already done so, w adapts its top list. Note that w can receive the update message along different paths. Then, if not already completed, w in turn builds the update request in its next exchange messages to its neighbors, and so on. When BridgeFinder is started for the first, time every node has only one entry in its list: the node itself. Distributing an update message to every node in the network requires at worst as many exchange steps as the network diameter.

In many real networks, the diameter is extremely small: the network of human acquaintances that has over 6 billion nodes has a diameter close to 6 [Gua92]; and the Internet with hundreds of millions of nodes, and it has a diameter around 20 [AJB99]. Wireless networks have larger diameters (no exact estimate available, as it depends on the network size) but possess similar community structures and high clustering features as the two networks mentioned above. Therefore, distributing the fastest converging nodes list within such networks also produces very modest overhead.

After each run of the BridgeFinder algorithm, every node knows the fastest converging nodes. As we show in Section 7.6, the fastest converging nodes are also the nodes with high betweenness and centrality, i.e., they are the critical nodes. Once they have been identified, measures can be taken to protect the network around them.

5.5 Properties of Critical Peers

The crucial observation on which our approach relies is that critical peers also play a significant role in distributing information within the underlying network (cf. Section 5.1). Our results show that with a very high probability, a critical peer either lies on many communication paths among other nodes or has very short communication paths to almost all other nodes, or even both. Detecting peers with such properties is with a very high probability, equivalent to detecting critical peers.

A straightforward question arises: How do we describe and detect peers playing such an important role in supporting communication flow? To answer this, we can use centrality measures from graph theory. However, these measures require global network knowledge and are very expensive to compute. Overcoming these two problems is the main contribution of BridgeFinder. It leverages the recently developed Push-Sum protocol and uses the convergence speed of the nodes to detect their role for supporting communication flow. Consequently, it requires no global network knowledge. Furthermore, as almost all nodes execute the protocol simultaneously, it has modest running time and produces almost no additional state per peer.
A detailed description of how BridgeFinder detects the nodes with the most significant centrality measures is given in the next section. Before that, we want to extend the betweenness and closeness measures, because they do not necessarily cover all cases of critical nodes.

Recall that the betweenness of a node v in graph G is equal to the number of shortest paths between any two other nodes in the network going through v (see Section 5.1). However, if v is even one single edge away from a node with high betweenness, its betweenness could be zero. That is because no shortest path goes through v, as this automatically increases the path length by one. Figure 5.3 displays the situation. It shows two groups of nodes. The numbers within the nodes are their corresponding betweenness coefficients. In the group on the left side, there is one central node with high betweenness. Its neighbors partly share its high betweenness, as the paths going through the central node are equally divided among all its neighbors. On the right side, however, betweenness is not that accurate in detecting central nodes. The node on the left side has a zero betweenness coefficient because it is one hop away from both high betweenness nodes. Therefore, no shortest path goes through it, as the path length will increase automatically.

Average Betweenness To overcome this drawback, we introduce the following measure:

Definition: The *average betweenness* of a node v is equal to the average betweenness of its own and the betweenness of its neighbors:

$$C_{AB}(v) := \frac{C_B(v) + \sum_{w \in M} C_B(w)}{|M| + 1} \tag{5.3}$$

where M is the set of neighbors of v.

In other words, not only nodes with high betweenness but also their neighbors have high average betweenness. This more accurately reflects their central position in the network for two reasons. First, these are the nodes that must overtake the information flow if the leader node fails. Second, they can serve as a backup for the leader node before it fails by interconnecting among each other and thus reducing its load. This also significantly reduces the impact of suddenly losing the leader node due to some unexpected reasons.

Average Closeness We also extended the definition of closeness to allow for a better identification of the critical nodes. Above, we mentioned that it points out central nodes. Nodes with small closeness are fast in distributing information through the network. However, closeness is still a linear measure. In large networks, there are whole regions of nodes with very similar closeness coefficients. In order to sharpen the precision of closeness, we define *square closeness*. It also computes all shortest distances among all nodes in the network, but squares them before the average is computed:

Definition: The *square closeness* of a node v is equal to the sum of square distances from v to all other reachable nodes:

$$C_{sqc}(v) := \frac{1}{n-1} \sum_{t \in N \setminus v} d_G(v,t)^2 \tag{5.4}$$

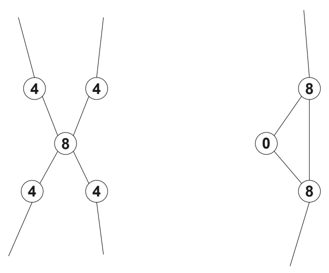


Figure 5.3: Betweenness does not always reflect the central role of a node.

where d_G, N and n are defined as in (2).

Nodes with best closeness have significantly higher squared closeness than the rest of the nodes. This makes it easier to detect outstanding nodes within clusters of nodes with relatively equal closeness.

The centrality measures we just introduced overcome the drawbacks of the standard centrality measures. However, they also describe the extent to which each node plays a role in supporting communication flow. Our empirical results show that the main load carriers in a network are also, in most cases, the critical peers. Their absence will break the network apart.

Although all these measures are useful in identifying the critical nodes, they are very expensive (if at all possible) to compute. Overcoming this problem is BridgeFinder's primary task, which we describe in the next section.

5.6 Gossiping Convergence

We generated four types of networks, and 500 networks of each type. The first type is the standard unit disc graph. In the second type, we place two large obstacles within the area where the nodes are distributed. The obstacles occupy the diagonal of the Euclidian space and basically divide it in two halves, leaving just a narrow bridge between the two halves. Then, we place the nodes uniformly at random within the free area and connect by an edge all nodes lying within a distance of 12 units from each other. The size of the Euclidean space is fixed to 100 x 100 units and all networks contain 250 nodes.

The third type of test networks is similar to the second one. The only difference is that this time, we place three equally large obstacles, instead of two. This results in two narrow bridges between areas accessible for nodes. Analogically, we place four obstacles in the fourth model, resulting in three bridges.

Note that the existence of a bridge between the obstacles does not necessarily mean that there are nodes occupying it, nor that it is used to connect different parts of the network. Thus, there is also no guarantee that a network generated with

Network Type	Push-Sum	BridgeFinder
3 Bridges	469	214
2 Bridges	597	282
1 Bridge	823	367
Unit Disk	94	72

Table 5.1: Convergence comparison between PushSum and BridgeFinder on different network models measured in average number of exchange steps per node.

models 1 to 4 will even be connected. Therefore, we generated as many networks from each type as necessary to acquire 500 connected instances from each type.

We ran the Push-Sum algorithm on the 500 instances of each network type. The averaged results over each 500 runs are displayed in Table 5.1. For a definition of convergence speed, please refer to Section 5.1. The result for a single network is measured as the required average number of exchange operations per node. We observe a large discrepancy in the required exchange operations between the ideal case, the unit disc graph, and the most demanding model of just one single bridge.

Convergence Speed Observation In our tests, we saw that there is one very important condition on which the convergence speed of the algorithm depends, namely, which node starts the algorithm. The faster a node can distribute information within the network, the larger the benefit of starting the algorithm from that node. If we know the nodes with best topological positions, then we can improve the running time of the algorithm by starting it from such nodes.

An intuitive solution arises. After running the algorithm once, the fastest converging node from the last run is responsible for starting the next run. If it is not able to do that, the second fastest converging node becomes responsible for starting the algorithm, and so on.

This mechanism is integrated in BridgeFinder. Recall from Section 5.4 that the list of the fastest converging nodes is constantly exchanged among nodes, and that at the end of the algorithm, each node possesses the list of the 10 fastest converging nodes. Thus, all nodes within the network are aware which node should start the next run.

Adaptation of Protocol Initiation We also need to overcome the problem of starting the algorithm for the first time, as well as avoiding different nodes from starting and running BridgeFinder simultaneously. To achieve that, each node, which decides to start the algorithm and has not already participated in it, picks a random number and sends it over the network together with its values. If a node receives values with a different random number in it, this means that two simultaneous instances of BridgeFinder are running. The node ignores the gossiping messages with the smaller random number and processes only the larger values.

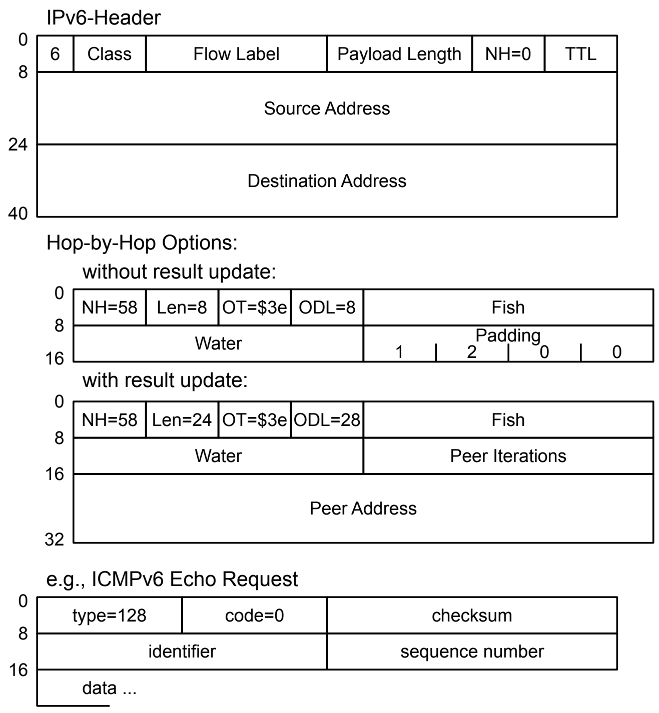


Figure 5.4: Implementation of BridgeFinder with IPv6 optional headers

Thus, BridgeFinder instances started with larger random numbers take priority over those started with smaller numbers. This produces a slight overhead during the first run of the algorithm, as initially computed values are being thrown away, but still gives each node the possibility of starting the algorithm. This resolves the problem of having to select a starting node within a distributed environment. Any node can start the algorithm.

The benefit of starting BridgeFinder from the best converging nodes from previous runs of the algorithm can be seen in Table 5.1. We ran the algorithm over all 500 instances of each network model and averaged the results. The values of BridgeFinder are always measured at the end of the second run, as the first one is used to determine the top converging nodes within the network. Using that approach, BridgeFinder performs better than the original Push-Sum algorithm in all four network models. The acquired speed-up ranges from 20% to 220%.

Ultimately, even in a hostile environment like one-bridge network models, our algorithm requires only a few more exchange operations per node than the overall number of nodes in the network.

5.7 Implementation

When a gossiping protocol like Push-Sum is used on the application level, then no additional data needs to be transmitted during the convergence phase, because the original gossiping protocol already transmits the *water* and *fish* values. To implement BridgeFinder, it is only necessary to distribute the information about the fastest converging nodes throughout the network after the convergence phase.

However, BridgeFinder can also be used in conjunction with any other routing algorithm. In the following, we consider an implementation based on IPv6 optional headers (RFC-2460), where the information required by BridgeFinder is piggy-backed on the normal IPv6 traffic. Figure 5.4 shows the layout of an IPv6 packet. In the IPv6 header, the *Next Header* (NH) field is set to 0 to indicate that a Hop-by-Hop Options header follows.
In the Hop-by-Hop Options header, the *Next Header* value specifies the type of the successive header. For example, a ICMPv6 packet would use type 58. Following RFC-4727, the option type field is set to $3e. The type is encoded such that the highest-order two bits specify the action that must be taken, if the processing IPv6 node does not recognize this type. A value of 00 instructs the receiver to skip this option and to continue processing the remaining packet. Hence, it is possible to use BridgeFinder even in networks where some peers do not support our custom header. The third-highest-order bit of the type specifies whether or not the Option Data of that option can change en-route to the packet's final destination. This bit is set, because we replace the header information at each intermediate peer. Setting this value excludes the header data from the calculation of message digests, as used by IPv6 authentication. Consequently, authentication will still operate correctly end-to-end.
The options header exists in two variants. The fields *fish* and *water* are sent with each message. As long as a peer has result updates to distribute, it also adds the next item of its update set to the header.

5.8 Evaluation

We evaluated BridgeFinder using the OADG (cf. Chapter 4) network model. As described in Section 5.6, we place obstacles in an Euclidean space, and the nodes are divided in densely connected clusters that are only sparsely interconnected. The number, size, and location of the obstacles determine the number of possible *bridges*, i.e., narrow paths among occupied areas. Thus, different clusters can be connected only sparsely, when at all. We evaluated four different network types with none, one, two and three *bridges*.

Simulation Note that nodes are distributed in an unpredictable way within the areas around the obstacles. Therefore, there is no guarantee how many of the

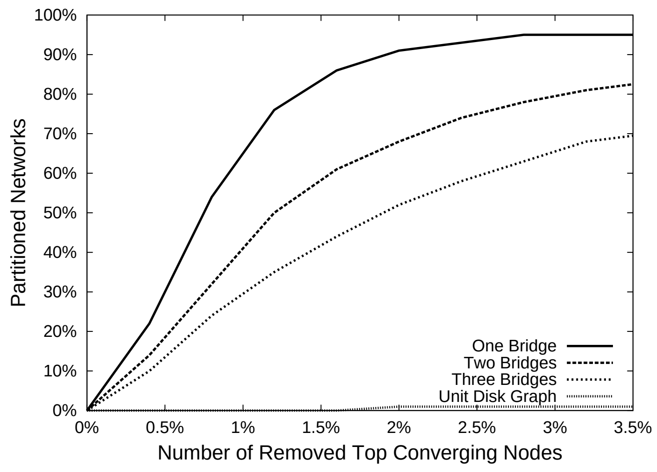


Figure 5.5: Destroying networks by removing the fastest converging nodes

existing bridges are actually used to connect the different clusters, nor how many different paths run through each bridge.

However, networks generated in this manner are very vulnerable. The few nodes lying on the paths among clusters are exactly the few nodes keeping the network together. We address two different questions. First, what topological properties do these few nodes possess? Second, and more importantly, how effective is BridgeFinder in detecting those very nodes?

We created 500 connected networks of each type. Each network consists of 250 nodes placed in a physical area of 100 x 100 units. The maximum edge length is set to 12 units, i.e., we place edges among all nodes within distance of 12 units from each other. We created multiple obstacles with dimensions of 25 x 25, 50 x 50 and 30 x 30 units (see figure 4.2 for an example setup). Due to the obstacles and the random placement of nodes, the resulting networks are not always connected. Therefore, during the generating process, we discarded non-connected networks, and new ones were generated until 500 connected instances of each type were acquired.

Evaluation Results Figure 5.5 shows the important role the nodes identified by BridgeFinder play in keeping the network connected. One by one, we removed the fastest 3.5% converging nodes. The x-axis shows the percent of removed nodes. The y-axis shows the fraction of partitioned networks from the 500 instances of each type. We consider a network partitioned if more than 25% of the remaining nodes are not able to communicate with the rest of the network. Hence, isolating only a small number of nodes is not a globally critical event, and we do not consider it as a problem.

Figure 5.5 shows averaged results for all four different network types. Removing

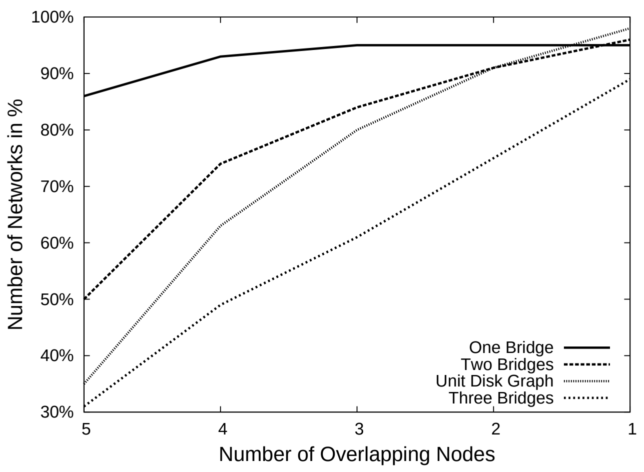


Figure 5.6: Intersecting the nodes with best centrality measures with 2% of the fastest converging nodes

only 2% (equivalent to just 5) of the fastest converging nodes breaks 90%, 70% and 50% of the one, two and three bridge networks, respectively. BridgeFinder finds with very high probability exactly the few nodes that keep the networks together.

Note that the standard unit disc graph does not partition; this is not surprising. In such graphs, there are no critical nodes at all. Depending on the density of the nodes, there are two possibilities. Either each node is very strongly interconnected with the rest of the network, or all links are so sparse that each node is critical. Thus, in unit disc graphs, there are no "special" nodes. Therefore, they all have very similar convergence speeds. Removing the fastest converging nodes is equivalent to removing randomly chosen nodes. Figure 5.5 just confirms the well-known fact that unit disc graphs are resilient against attacks.

Analysis of Results Recall the two global measures we defined in Section 4: *average betweenness* and *square closeness*. They are tools from graph theory for describing the importance of nodes for distributing information within a network. To answer the above question, we evaluate to what extent the top nodes identified by BridgeFinder overlap with the best nodes identified by the two global measures.

Betweenness and Closeness Sets For each of our test networks, we compute two sets: A and B. Set A consists of the highest 5% (i.e., 13 of all 250 nodes) average betweenness nodes, and set B contains the highest 5% square closeness nodes. The nodes in the union of A and B, $A \cup B$, have either the highest average betweenness or the highest square distance coefficients in the network, or in most cases, both. Note that A and B are not necessarily disjoint. On the contrary, averaged over all generated networks, A and B overlapped to over 80%. The overlapping factor is almost identical for the four different network types.

Simulation Results The set $A \cup B$ contains the nodes with the "best" topological properties in the network. The question is how many of those nodes are within the nodes identified by BridgeFinder. Figure 5.6 shows the results from the interSection of 2% (equivalent to 5) of the fastest converging nodes with $A \cup B$. In 85% of the one-bridge networks, all five nodes have either high closeness or high betweenness, i.e., are in $A \cup B$. Three of those five nodes have the same features in about 80% of the two-bridge and unit disc networks, and in about 60% of the three-bridge networks. Thus, the top nodes identified by BridgeFinder lay exactly on the key positions in their networks. It is not surprising (as shown in Figure 5.5) that removing even a few such nodes can damage a significant portion of the underlying networks.
Next, we intersect $A \cup B$ with 5% (13 nodes) of the fastest converging nodes. Figure 5.7 displays the results. At least half of the fastest converging nodes lay on key topological positions in their networks. This is the case in more than 80% of the two-bridge networks and in over 90% of the one-bridge networks.

Our empirical results show that with very high probability, BridgeFinder identifies the same nodes one would obtain by using global network measures. In both cases, those are the few nodes keeping the network together.

Still, there are two very important differences between the two approaches. First, our algorithm is faster in several orders of magnitude than computing global network measures and furthermore produces very modest computational costs per node. Second, and more importantly, BridgeFinder is a distributed approach. That makes it a useful tool that can be applied to a vast variety of real world networks.

5.9 Security Issues

All gossiping-based systems are unfortunately vulnerable to attacks and malfunctioning nodes. This is because gossiping is a distributed approach, and an application based on it cannot rely on a central trusted authority. This excludes any standard approaches from the cryptography for verifying any exchanged information or identifying its sources.

BridgeFinder is also based on gossiping and is thus similarly vulnerable. However, we are not interested in the values exchanged among nodes, but rather on the speed with which this process converges. Convergence, on the other hand, is not a local decision, but a mutual decision of the whole network of nodes. Relying on the convergence speed instead of the values is enough to develop mechanisms that make BridgeFinder resistant to attacks.

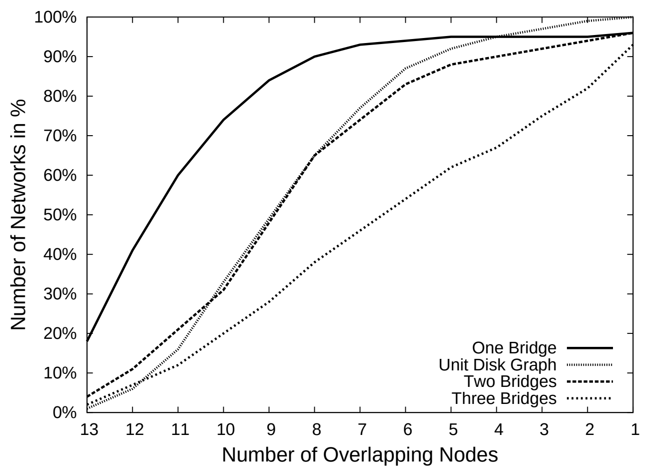


Figure 5.7: Intersecting the nodes with best centrality measures with 5% of the fastest converging nodes

Basic Approach The main idea behind our solution is that no node converges on its own. A node can only claim it has successfully converged when the fluctuations of the values possessed by the nodes surrounding it become relatively small. On the other hand, a node can only claim it has still not converged when the values maintained by its neighbors still vary significantly.

Therefore, each node needs just a scheme to estimate the fluctuations in the neighbors values either as acceptable or not acceptable. Then, it can protect both, itself and the whole network, from the following two types of possible attacks. The first attack happens when a compromised node falsely claims it has converged (lies to its neighbors about its real values) in order to qualify for the list of fastest converging nodes. The second type of attack is when a node neglects the values sent to it by its neighbors and continues flooding the network with higher/smaller values, and thus preventing the algorithm from converging. If a node receives suspiciously small or high values from one of its neighbors, let us say X, it can warn its other neighbors. If its suspicions are confirmed by any other nodes, they send a message through the network that X has been compromised. Then, all nodes having X in their neighbor list exclude X from their communication flow.

To estimate whether the values sent by its neighbors are legitimate, a node can calculate the mean value of the values received from the neighbors. Then, it can use the average in a local decision process: each single value can differ only with a given factor from the mean value.

The idea of using the mean of the neighbors values within a gossiping-based application to estimate the impact of compromised nodes has already been studied in [KDG03]. There, the authors show how one can use the median to compute an acceptable quantile for the exchanged values under erroneous inputs from compromised nodes. As mentioned above, it is not the values we are interested in, but rather the convergence speed.

Progression Speed We know that the fluctuations in the values are significant at the beginning of the algorithm and decrease progressively with each iteration. The speed of that progression for a node v at time i can be expressed through the difference of its new and its old values:

$$s_v(i) := \frac{x_i - x_{i-1}}{x_i} \tag{5.5}$$

where x_i and x_{i-1} are the current and the last values of v respectively. We call s_v the speed coefficient of v. As the algorithm converges, s_v goes to zero for all nodes in the network.

Furthermore, each node knows the number of exchange operations it has already carried out (cf. Section 5.4). Based on that number and the speed coefficients of the values it receives, each node qualifies them as suspicious or not suspicious.

The local decision making process for a node v works in three steps. When at iteration k, v receives a new value x from its neighbor w: i) v computes the speed coefficient of w at k, $s_w(k)$ (the one corresponding to x); ii) it computes the mean value of the speed coefficients of the rest of its neighbors MS_v; and iii) it checks whether the following inequality holds:

$$|s_w(k) - MS_v| \leq \frac{1}{k} \tag{5.6}$$

In case it holds, then v accepts the new value x of w. Otherwise, it marks x as suspicious and reports it to its other neighbors. If enough nodes (one can vary the number of required votes from one to all neighbors) confirm the suspicions regarding v, then w can be excluded from the network.

Figure 5.8 shows the progression of the averaged speed coefficient over all nodes in the corresponding network and the function $1/x$. All plots are averaged results over 100 instances of each network type. The speed coefficients in all network types have similar gradients. They all are dominated by $1/x$ for $x > 70$.

One can easily conclude that the decision rule proposed in 5.9 is inadequate within the first 70 iterations, but is very tight afterwords. That is, we allow compromised nodes to lie about their values at the beginning of the algorithm. The huge fluctuations in the beginning of the algorithm make it highly unlikely that a meaningful mechanism for that phase of the algorithm exists. At the beginning, large values are exchanged with very small values in every exchange operation.

Our security mechanism remains successful if it overcomes the following three challenges: i) detect nodes falsely claiming they have converged; ii) detect nodes that manipulate their values after the network has stabilized; iii) detect malicious nodes changing their values always within the tolerance interval, which should not influence the convergence speed of the whole network.

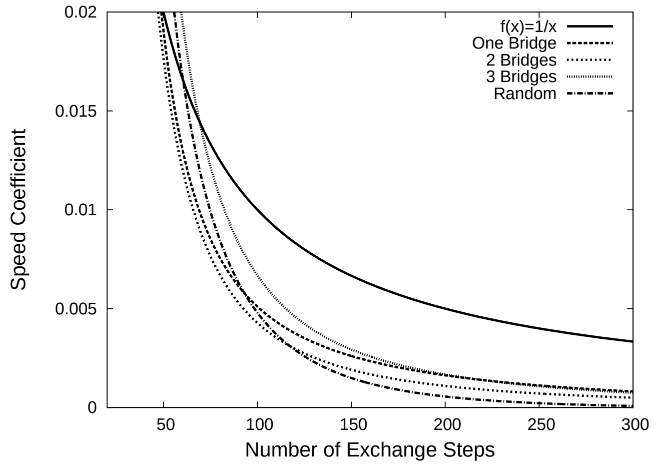


Figure 5.8: Average speed coefficient within one run of BridgeFinder on all four network types

We address the first issue directly. A node cannot possibly lie it has converged because once the network has stabilized, its neighbor will detect that its speed coefficient is extremely close to zero in comparison to its surrounding nodes.

From Section 5.9, one can also read the magnitude to which a given value can be manipulated. One can easily calculate that after only 100 iterations, even discrepancies within 5% of the real value will be detected as suspicious. This buffer decreases with each next iteration of the algorithm.

To address the final issue, we simulated the following scenario. Ten percent of the nodes are malicious. They all have an oracle that provides them with the mean value of the speed coefficients of their neighbors. These mean values are not known under realistic circumstances. Then, all malicious node lie within the maximum buffer at each fifth iteration. The results are displayed in figure 5.9. Even under those circumstances, the overall convergence times of all four network types remain intact.

We point out that this security mechanism does not prevent nodes from inserting false values as long as they stay within the current tolerance interval of the receiving node. That means that there is no guarantee that the output values have not been compromised. Therefore, any information that can be extracted from those values, e.g., number of nodes in the network, has to be used very carefully.

However, compromised nodes change convergence speeds only marginally. This guarantees the proper function of the BridgeFinder algorithm, even in the presence of compromised nodes.

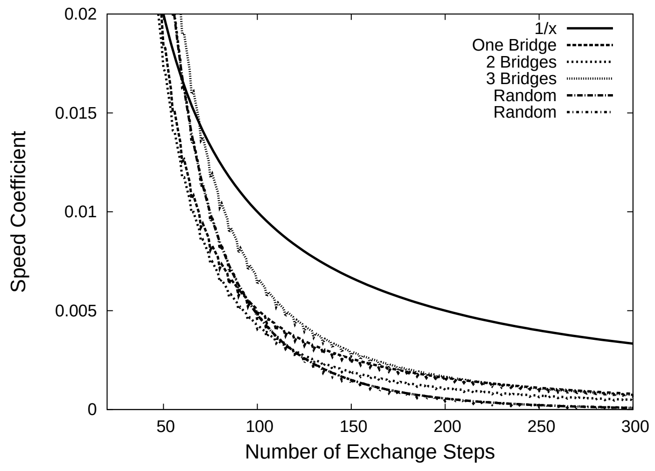


Figure 5.9: Optimal attack strategy every fifth iteration of BridgeFinder with 10% of malicious peers

5.10 Summary

In this chapter, we presented BridgeFinder, a distributed algorithm for identifying critical nodes in complex networks. Critical nodes are those whose disappearance would partition the network, which would affect almost all elicited use cases 2.2. Identifying and protecting those nodes is vital to keeping the network operational. The benefit of using BridgeFinder is significant in fast-changing environments typically seen in first response situations. The approach may be applied to wireless multihop networks, sensor networks and P2P networks. To the best of our knowledge, there is no equivalent distributed approach. BridgeFinder is fast, reliable, produces almost no additional message overhead and only small state per peer.

Approach	Detect Global Criticality	Construction Steps	Error Rate	Global Error Rate
CAM	○	1	Low	high
DMCC	●	2	No error	tunable
XTC	○	2	No error	n.a.
BridgeFinder	●	1	n.a.	low

Table 5.2: Contributions and comparison with related work

Table 5.2 shows BridgeFinder in comparison to its directly related approaches. The main contribution is the reliable detection of global critical nodes. It does not test only some nodes, but rather evaluates, in parallel, all nodes in the network.

When the algorithm is finished, the results are known to all nodes. Even with a small state per peer and based only on local computations, the results are highly accurate.

Furthermore, unlike other distributed approaches, it is resilient to attacks by malicious nodes (cf. Section 5.9). BridgeFinder can easily be implemented on any application or routing protocol, and we have shown as an example how to implement it using IPv6 header options. With BridgeFinder a communication network may be prevented from partitioning. This is essential for a P2P overlay network, which heavily relies on a connected communication graph. In the next chapter, better known P2P overlay approaches are evaluated for their applicability to the volatile environment found after larger scale disasters.

Finally, although BridgeFinder is a fully distributed algorithm, it still has a reliable guarding mechanism against malicious nodes attempting to harm the system. This is because BridgeFinder focuses on the convergence speed and not on the values collected by gossiping.

Chapter 6

P2P Overlay Network Layer: Search and Lookup

In this Chapter existing types of P2P-networks are evaluated under the harsh environment found in first response situations. The findings of the conducted simulations are valuable building blocks, which finally lead to the new P2P overlay network PathFinder. It is designed to provide both, key lookup and exhaustive search, while being able to recover quickly from network outages.

Section 6.1 revisits the challenges for overlay networks in first response situations. In Section 6.2 state of the art robustness analysis for P2P overlay networks, as well as the related lookup and exhaustive search mechanisms, are surveyed. In Section 6.3 the better known types of P2P networks are evaluated under the occurrence of high churn, which leads to the design recommendations presented in Section 6.4. Then, in Section 6.5 we propose a new P2P overlay network, which works under hostile first response environments and combines flexible exhaustive search with quick key lookup capabilities. Section 6.6 summarizes the most important aspects of the proposed PathFinder overlay design and runtime metrics.

6.1 Introduction

First response communication needs to deal with heterogeneous devices, small PDAs with limited bandwidth and limited CPU power, as well as highly available servers offered by operation control (see Section 2.2). Current first response communications and planned future systems are based on wireless LAN or WIMAX [int05, www06] (cf. Section 2.4).

6.1.1 General Approach

We cannot assume a wireless access point infrastructure right after a catastrophic event. Therefore, the network must also support a wireless ad-hoc mode, at least until a more stable communication infrastructure is available. In a typical first response case, the personnel carries small, handheld devices. Even sensor technology might be used. In addition, more powerful equipment is also present, e.g., at the command post. These nodes generally have much more power and bandwidth available. In addition, these fixed spots are able to provide reliable wireless LAN infrastructure, which greatly improves transfer rates.

In this thesis, we analyse four different overlay architectures: a fully unstructured network, a network based on a distributed hash table, and two kinds of superpeer networks, one with random superpeer selection and another with superpeer selection based on actual peer capabilities. Our goal in this context, is to identify the strengths and weaknesses of each of these overlays for the particular needs of a first response scenario. The expected results are to find a reasonable range in which the different P2P overlays perform as intended and to find the borders at which an overlay becomes unusable.

The unstructured network is a randomly connected network where each peer has three neighbors. We evaluated two superpeer network configurations. In the first, all peers had the same resources, and some were selected as superpeers. In the second, we gave some peers higher uptimes and selected them as superpeers. For the DHT, we selected Tapestry [Zea04] (the results apply for other DHTs as well). We initially assume that the actual network is able to provide fully reliable ad hoc routing from any node to any other node. This enables us to evaluate the resistance to churn.

6.1.2 Requirements

The main task of the 'Overlay Network' layer is to provide peer discovery and search functionality (cf. Section 3). Our contribution in this context, a novel P2P overlay network, may be used for a wide set of applications. However, it was designed to better support the following typical requirements described in Section 2.2.

R4 (Resource Awareness) In order to utilize the heterogeneous resources effectively, the P2P overlay network needs to be designed to allow for dynamic load distribution. This property is one of the key design choices of the proposed overlay network PathFinder.

R10 (Allow horizontal and vertical message transmissions) The developed communication strategy must not restrict the message flow by design. It is the task of the 'Application Layer' to implement the desired communication workflow and eventually to restrict distinct communication channels.

R5 (Active Search) Searching is one of the key features of any overlay network. Nevertheless, DHT overlay networks provide quick key lookup mechanisms, but do not support effective full-text search mechanisms. However, unstructured networks provide flexible exhaustive search mechanisms, but are not capable of efficient key lookups (cf. Section 3.2). Being able to combine searches and lookups in the same overlay is very desirable, since it allows efficient and overhead-free implementation of natural usage patterns. Our design requirement for a novel P2P overlay network is to combine searches and lookups in an efficient manner.

6.2 Related Work

The first P2P overlay network for first response communication was developed by the University of Virginia [A.S03a]. They identified main challenges in current first response approaches and developed a prototype for a P2P-inspired first response solution. Further implementation of their P2P solution [A.S03a] is done using hypercast, GPS capabilities, multicast streaming video and access control mechanism. Nevertheless, the goal was to develop a prototype implementation, large scale simulations were not conducted.

DHT Overlay Networks in dynamic environments The basic DHT functionality is very well studied [ZKJ01, I. 01, MNR02, RFH^{+}01a, RD01b]. DHTs have been studied in volatile environments, in particular, at higher churn rates, DHTs tend to become unstable[S. 04]. Churn rate behavior is both theoretically examined [S. 05] and simulated.

Unstructured Overlay Networks in dynamic environments BubbleStorm [TKLB07] is one novel unstructured P2P overlay network. Evaluations have shown its great robustness in highly volatile environments. As discussed in [TKLB07], BubbleStorm is currently the most efficient overlay for exhaustive searches. However, like all other unstructured P2P overlay networks, it does not support any mechanisms for key lookup.

Hybrid Approach Solar [CK03] is a middleware layer suitable for transportation context information in emergency response scenarios. Solar is based on a superpeer architecture, in which the superpeers are connected over the Pastry [RD01a] DHT. Although the superpeers have more transmission power, their interconnection network is still subject to churn and network congestion effects, and it is not clear that a DHT is the correct choice for the superpeer interconnection.

Combination of structured and unstructured P2P overlay networks These approaches and our findings in this Chapter lead to our main contribution, the efficient combination of an unstructured and a structured P2P network. To the best of our knowledge, PathFinder is the first such overlay. This combination allows for the support of both efficient exhaustive searches over the overlay, as well as DHT-like key-value lookups.

Besides our approach of implementing DHT functionality on a random graph, some different kinds of overlay graphs are proposed to improve their performance. These include Viceroy [MNR02], which is based on a butterfly graph, and Koorde [KK03], whose underlying topology is a DeBruijn graph. Even though butterfly graphs have nearly optimal diameters, practical implementations of Viceroy leads to lookup delay of $3log_2(N)$ [LKRG03], on average. The Koorde had to abandon the property of constant degree in order to handle node failures. This leads to degree growth of $O(log(N))$.

6.3 Evaluation

We now present the results from our evaluation, which are averaged over 20 simulation cycles. All four evaluated connection topologies were simulated using Peersim (see http://peersim.sourceforge.net) and perform perfectly in the case of no churn. In each case, we simulated the disconnection of some fraction of peers, and varied this fraction from 0 to 1. The failed nodes were randomly picked. Analyzing the resulting network graph, we determined the number of clusters and the size of the largest cluster. These are vital metrics for ensuring reliable delivery of critical messages in a first response scenario. If there is no path between two nodes, there is no possibility to send a message. If a path exists, it should be possible to forward all messages to their correct recipients under the assumed reliable ad hoc routing (see Section 4.5). The goal of our first evaluation is to compare the different overlay networks and to determine which of them performs best at different node failure ratios.

6.3.1 Unstructured Network

Figure 6.1 shows the results for the unstructured network case. The network consisted of 2000 peers. The x-axis shows the percentage of failed nodes, the left y-axis the size of the largest connected component (cluster), and the right y-axis shows the number of clusters. The random topology is created with an average node degree of 3 and non-directed connections. The advantage of a random topology is the low clustering factor. Even with a 30% rate of failure, only 2.5% of all remaining peers are unreachable. Further increasing churn rates result in a complete collapse of the topology, occurs at 70%–80% failure rates.

6.3.2 Superpeer Network

We consider two cases for the superpeer network. In the first case (random select), all nodes are homogeneous and some of them are designated as superpeers. The second case, sensitive select, simulates a typical first response scenario after the first larger wave of responders have arrived with more communications equipment. In this case, we select a small subset of peers, make them more robust against failure, and designate them as superpeers. In both cases we assume that the superpeers form

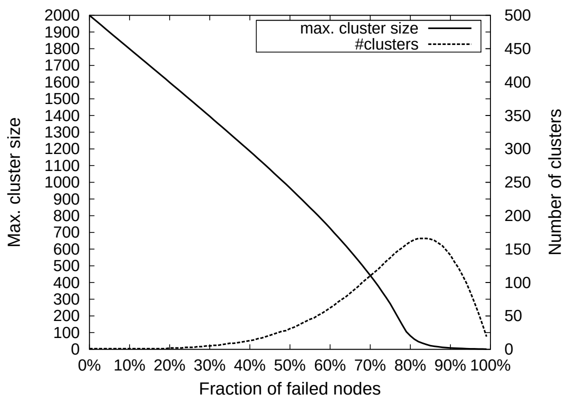


Figure 6.1: Clustering in unstructured network

a full mesh between them, but other network topologies are also possible [CK03]. Selecting good peers as superpeers can be performed on-line by the peers themselves. However, these details are outside the scope of this thesis.

From the 2000 peers, we designated 200 peers as superpeers, and the rest of the peers were evenly distributed between the superpeers, with each peer connected to only 1 superpeer. (Connecting to several superpeers in range might be desirable in a real-world deployment.) As mentioned above, the superpeers form a full mesh between themselves. In the first case, all nodes had equal failure probabilities. In the second case, with more powerful peers as superpeers, the superpeers had a failure probability that was 1/30 of the failure probability of the normal peers. Note that it is still possible for superpeers to fail; it is just much less likely for this to occur than for normal peers. As we evalute a worst case scenario, we did not provide backup links for client peers, i.e., if a superpeer fails, all connected clients are considered as disconnected.

Figure 6.2 shows the size of the largest cluster and number of clusters for the two superpeer cases. The thin lines show the random select case, and the thick lines show the sensitive select case where superpeers had longer uptimes.

The sensitive select strategy, i.e., selecting superpeers with better uptimes, improves performance significantly. The size of the largest cluster is almost the same as the number of remaining peers, meaning that in most cases, the complete network remains connected. The number of clusters is also minimal. In contrast to the randomly selected superpeers, the number of clusters is cut by a factor of 10.

6.3.3 DHT

We also investigated the Tapestry DHT in the same scenario. It is known to maintain many connections to its neighbors, which should make it robust against node failures.

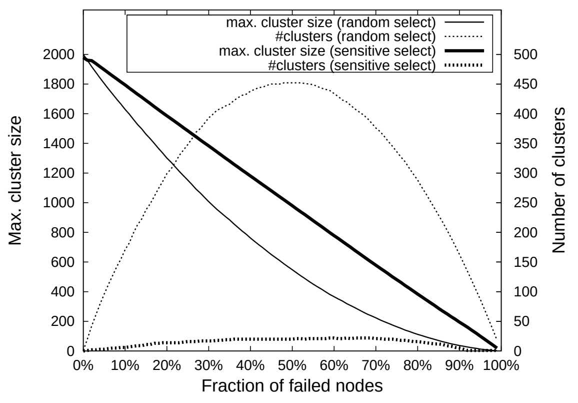


Figure 6.2: Clustering in random and context-sensitive superpeer selection

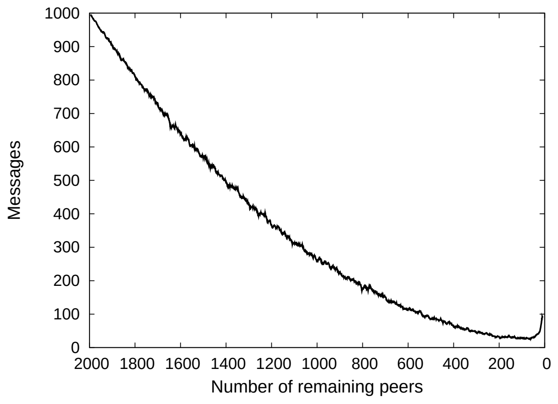


Figure 6.3: Message reachability in Tapestry

We observed that in all of the cases, the Tapestry network remains connected; thus, in principle, every message should always reach its intended recipient. We evaluated the successful messages by choosing a random pair of peers 1000 times and letting one of them perform a lookup of the other. The results are shown in Figure 6.3. The x-axis again plots the fraction of failed nodes, and y-axis shows the fraction of messages that reached their intended recipient. Note that the sender and receiver were chosen from the set of nodes that were still alive after the failure. In other words, 100% reachability was theoretically always possible.

Our results show a message success rate of less than 30%, if approximately 50% of peers fail. In other words, even though a graph of Tapestry is not clustered, the delivery rate of messages dramatically decreases even at medium churn rates.

The main reason for the low success rate is that Tapestry relies on certain routing table entries for forwarding messages. Note that we did not run any topology maintenance in any of the overlays after the failure. Each of the overlays is able to recover full connectivity after the usual periodic maintenance, but the costs are different. Unstructured and superpeer networks have relatively low costs, but Tapestry needs to rebuild its routing tables, which presents a considerable overhead.

Although our evaluation was made with Tapestry, similar results could be expected from any other DHT, with small differences in terms of connectivity and reachability. However, all DHTs are based on the assumption that any peer is able to send a message to any other peer. Under the reliable ad hoc routing assumption from above, this is feasible, but under more realistic conditions, this assumption is no longer valid.

6.3.4 Summary of Results

Both the unstructured network and Tapestry are very good at keeping the network connected, i.e., minimizing the number of clusters. The superpeer network with correct superpeer selection is able to achieve similar performance. However, both the unstructured network and Tapestry require a fully reliable network to deliver the service; failures will severely impact their performance.

In Table 9.1 we summarize the behaviors of the overlay networks as a function of the fraction of failed nodes. We consider two cases: *operational*, where 70–100% of the messages are routed to the correct recipients, and *failed* where less than 50% of the messages arrive correctly. For each overlay network, we show how many nodes in the overlay are allowed to fail for the network to meet the performance targets. The reason for failure is that either the network has clustered or is unable to find a correct routing path (DHT).

The unstructured topology performs better than randomly elected superpeers (SP/Random); the failure occurs at 75% node failure, while in the superpeer case, a 50% rate is sufficient to push the network to a failed state. This is because when a superpeer fails, all its normal peers become individual clusters. Tapestry, as discussed in Section 6.3.3, performs very poorly but the network remains connected. All values are derived from the measured clustering which is shown in figures 6.1, 6.2 and 6.3. For example in the unstructured, case 72% of the nodes (1440 peers) must fail in order to keep 70% of the remaining nodes (392 peers) operational, i.e., 30% (168 peers) of the network are not part of the giant component.
If superpeers are selected based on their uptime, the network remains operational. This is because, as shown in Figure 6.2, there is only a minimal amount of clustering.

In summary, good selection of superpeers yields significant performance gains over normal unstructured networks. Our results also show that simply selecting superpeers is not sufficient to obtain good performance.

Topology	Operational	Failed
Superpeer	Always	Never
SP/Random	27% failed	50% failed
Unstructured	72% failed	76% failed
DHT	15% failed	30% failed

Table 6.1: Effect of node failures

6.4 Design Recommendation

We have evaluated different overlay networks in terms of their suitability for first response scenarios. Our main findings and design recommendations are:

Exploit stable peers We have found that an overlay that may adapt its topology to carefully selected stable peers (superpeers) is promising and greatly increases the operational state (see Section 6.3.4).

Unstructured topology An unstructured network is able to handle larger amounts of churn as well. Node failures hardly partition the network, and a giant component remains functional until the failure exceeds about 75% of the nodes. The high robustness against churn is predicted in a theoretic analysis of unstructed networks [GHW07].

DHT overlay network A DHT-based network, although it remains connected, is sensitive to churn. It is able to transmit messages to correct recipients only if the network is relatively stable. Nevertheless, key lookup is one important functionality provided by structured P2P overlay networks. Keeping this sensitiveness in mind, DHT functionality should be offered if the network reaches a stable state. As soon as more volatile user behavior or environment changes occur, the overlay network should return to the unstructured mode of operation.

6.5 PathFinder

Given the derived requirements from Section 6.1.2 and the findings from Section 6.4, the proposed overlay network

1. shall provide flexible search mechanisms.
2. should exploit peer heterogeneity i.e., be resource-aware.
3. might use an unstructured topology.
4. should support DHT functionality.

Our approach, PathFinder, satisfies all of these requirements; it offers efficient lookups and efficient exhaustive search in the same overlay. PathFinder is based on a random graph overlay, which allows us to use existing mechanisms for efficient search. In addition, we have designed an efficient key lookup mechanism that enables DHT functionality. By introducing the concept of virtual peers, resources may be freely distributed among the available peers. In addition, our evaluation shows that PathFinder scales up to hundreds of millions of nodes and is comparable or better in terms of lookup and search performance, when compared to existing overlays.
To reiterate, the main contribution of this Section is the *efficient combination of searching and lookups in a single overlay*.
We have evaluated PathFinder analytically and with simulations. We investigated its resistance to churn and its robustness. Our results clearly show that PathFinder is highly scalable, fast, robust, and requires only a small per-peer state. In terms of exhaustive search performance, PathFinder is similar to BubbleStorm [TKLB07]. In terms of DHT-like lookup performance, our results show that PathFinder is at least as good as current DHTs and, in most cases, is able to retrieve objects with less overlay hops than other DHTs.

6.5.1 Introduction

Given the attractive properties of human-friendly keyword searches in unstructured networks and computer-friendly and efficient lookups in DHTs, it is natural to ask the following question:
Is it possible to combine the desirable properties in a single overlay network?
Of course, it would be possible to run two overlays in parallel, but a single overlay is much more desirable, as it has considerably lower overhead, both in terms of overlay maintenance and replication effort. Parallel overlays require twice the maintenance traffic and state maintenance, as well as wasting space when objects are replicated.

General Idea In this thesis we present PathFinder, a P2P overlay that combines an unstructured and a structured network in a single overlay. PathFinder is based on a random graph, which gives it low average path length, a large number of alternate paths for fault tolerance, and highly robust and reliable overlay topology. Furthermore, the number of neighbors in PathFinder does not depend on the network size. Therefore, the load of single peers in PathFinder remains constant, even if the network grows up to 100 million or more peers.

Practical Example Being able to combine both efficient lookups and efficient search is extremely useful for distributed applications and human users (cf. Section 3.1). An efficient search (i) allows users to discover what content is available. An efficient lookup (ii) enables them to retrieve the content later or forward it to their friends, knowing that the friends will see the same content.

The equivalent of this on the Web is (i) making a Google search and then (ii) bookmarking the resulting Web page. The bookmark can then be retrieved later or sent to friends. The *key difference* of PathFinder is that there is no need for a centralized indexing service. The searches and lookups are *built into the system architecture*.

Challenge In our design of PathFinder, we are fully compliant with the concepts of BubbleStorm [TKLB07], namely the overlay structure based on random graphs. We augment the basic random graph with a deterministic lookup mechanism (see Section 6.5.2.3) to add efficient lookups into the exhaustive search provided by BubbleStorm. As shown in [TKLB07] and discussed in [GHW07], overlays based on random graphs are highly resilient even to large simultaneous crashes of peers. At the same time, they provide short average path lengths between two peers.
The challenge, and one of the key contributions of this Chapter, is developing a deterministic mechanism for exploiting these short paths to implement DHT-lookups.

6.5.2 PathFinder Design

In this Section, we present how the PathFinder overlay is constructed and managed. We will present in detail how key lookup, search, joining and leaving the network, and handling of crashed nodes are performed in PathFinder. We also show how the PathFinder network can be built in a practical scenario.

6.5.2.1 PathFinder Overlay Basics

PathFinder is based on a random graph[1] G with a fixed number of nodes M. For a network of N peers, M is typically between N and $2N$. Section 6.5.2.7 shows how to adapt M to the actual number of peers in the system. We call the nodes of G *virtual nodes*. The degree of a virtual node is Poisson distributed. Each physical node (i.e., peer) is responsible for one or more virtual nodes. In the remainder of the Chapter, we will use the terms "peer" and "virtual node" to denote the different actors.

Topology Creation Our approach is leveraging off a well-known characteristic of pseudo random number generators (PRNGs); given the same seed every peer will obtain the same sequence of pseudo random numbers. In PathFinder every node must keep a PRNG and the preset average degree of virtual nodes c. The graph construction proceeds as follows:

1. A peer initializes its pseudo random number generator with its own ID. (We propose generating peer IDs from their IP addresses (see Section 6.5.2.5)).

[1]More precisely, we use an Erdös-Rényi random graph. In the remainder of the thesis we use the term random graph.

```
seed_generator(myID);
foreach (virtualNode) {
  numNeighbors = nextPoisson();
  for i = 0, i < numNeighbors, i++ {
      neighbor = nextRandom();
      add(neighbor);
  }
}
```

Figure 6.4: Code for calculating neighbors

2. The peer determines how many virtual nodes it should handle. See Section 6.5.2.5 for details.

3. For every virtual node handled by the peer:

   (a) The peer determines how many neighbors it should have by drawing a pseudo random number and considering it to be from a Poisson distribution.

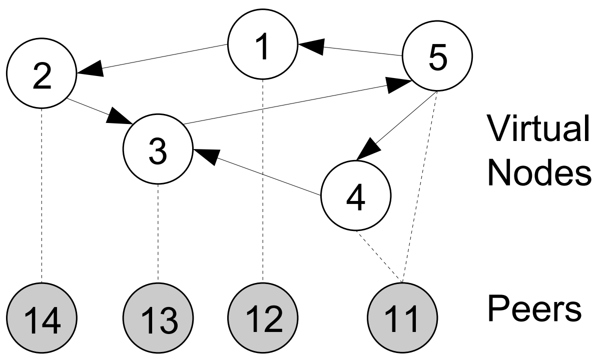
   (b) The peer then draws as many pseudo random numbers as it has neighbors and selects those virtual nodes as its neighbors.

Figure 6.4 illustrates this. Function `nextPoisson` returns a random number from a Poisson distribution to determine the number of edges. Given an average degree of c, the probability of having k neighbors is well-approximated by $P(k) = \frac{c^k}{k!}e^{-c}$. Function `nextRandom` returns a random number uniformly distributed between 0 and M.

A peer only needs a pseudo random number generator to perform this. There is no need for network communication. Similarly, *any peer* can determine the neighbors of *any virtual node*, by simply seeding the pseudo random number generator with the ID of the virtual node. Note that neighbor links in the random graph are *directed*.

Routing Table The routing table of a peer is determined by the neighbors of its virtual nodes. It contains all the direct neighbors of all of its virtual nodes in the random graph. These tables are easy to maintain, because all peers only hold between one and two virtual nodes on average (i.e., c to $2c$ neighbors). As our results show, the value of $c = 20$ is sufficient for good performance, and better performance can be obtained for higher values of c. Because one entry in the routing table contains the virtual node ID and its IP address, the value of c could possibly be set to very high, on the order of hundreds or thousands, without undue burden on a peer's storage.

Figure 6.5 shows a small sample of PathFinder with a routing table for peer 11. The random graph has 5 nodes (1 through 5), and there are 4 peers (11 through 14). Peer 11 handles two virtual nodes (4 and 5), and all the remaining peers have 1



Routing Table Peer 11			
	Outgoing		**Incoming**
ID	3	1	3
Peer	13	12	13

Figure 6.5: A small example of PathFinder with peers and virtual nodes

virtual node each. The arrows between the virtual nodes show the directed neighbor links.
Each peer keeps track of its own outgoing links, as well as incoming links from other virtual nodes. A peer learns the incoming links as the other peers attempt to connect to it. Keeping track of the incoming links is, strictly speaking unnecessary, but makes key lookups much more efficient (see Section 6.5.2.3). The routing table of peer 11 therefore consists of all outgoing links from virtual nodes 4 and 5, and the incoming link from virtual node 3. In general, every peer is responsible for keeping only its outgoing links alive. In contrast to established DHTs, the maintenance costs of PathFinder does not depend on the network size, as the average number of neighbors in the random graph is fixed.

6.5.2.2 Storing Objects

Each object stored in PathFinder has a unique identifier. We derive this identifier by hashing the contents of the object with some hash function (e.g., SHA-1). Objects, such as regularly updated news, lists of injured people, status reports, etc., that have changing content but would prefer to remain under a single identifier can create the hashes in any other manner.
The object is stored on the virtual node (peer responsible for the virtual node), which matches the object's identifier. If the hash space is larger than the number of virtual nodes, as with SHA-1, then we map the object to the virtual node whose identifier matches the prefix of the object hash.
There is no need for an additional lookup service, because PathFinder provides exhaustive search over all the objects (see Section 6.5.2.4). When a user wants to find an object, he performs a search that returns the object and its identifier.

Subsequent retrievals of the same object can use the identifier to perform a lookup (Section 6.5.2.3). If a user wants to send a link to the object to her friends, he can send the identifier (which can include human-friendly metadata for display).

6.5.2.3 Key Lookup

Key lookup is the main function of a DHT. In order to perform quick lookups, the average number of hops between peers needs to be small. We now present how PathFinder achieves efficient lookups.

Basic Approach Suppose peer I wants to retrieve object O. Peer I determines that virtual node J is responsible for object O. Denote the set of virtual nodes managed by peer I as V. For each virtual node in V, peer I calculates the neighbors of those nodes. (Note that this calculation is already done, as these neighbors are the entries in peer I's routing table.) Peer I checks whether any of those neighbors is virtual node J. If yes, peer I will contact that peer to retrieve O.
If none of peer I's virtual node neighbors is responsible for O, peer I calculates the neighbors of all of its neighbors. Because the neighbors are well-defined (see Figure 6.4), this is a simple local computation. Again, peer I checks whether any of the new neighbors is responsible for O. If yes, peer I sends its request to the neighbor whose neighbor is responsible for O.
Again, if no match is found, peer I expands its search by calculating the neighbors of the nodes from the previous step, and checks again. This process continues until a match is found. Eventually, peer I will have searched through the whole random graph, so a match is guaranteed.

Forward Backward Chaining For an average degree of c per virtual node, the above process requires us to compute c^i nodes for each step i. This becomes unwieldy for larger networks, which may require a larger number of steps. For example, with $c = 20$ and 100 million nodes we need approximately 8 steps, i.e., $20^8 = 2.5 \cdot 10^{10}$ nodes. We mitigate this by *expanding the search rings from both I and O*, as shown in Figure 6.6.

Because peer I is able to compute O's neighboring virtual nodes, I can expand the search rings *locally* from both the source and target sides. In every step, the search depth of the source and target search ring is increased by one. For efficiency reasons, peer I keeps its rings on its own side pre-computed in memory.[2] This way, peer I most only needs to start expanding the ring around the target and checking if the newly computed virtual node is in its own list. As peers keep track of their incoming links (Section 6.5.2.1), the above procedure functions effectively.

When a match is found, we have a path from the source to the target. We pass the discovered path along in the lookup message. Thus, every peer on the

[2]A peer can pre-compute rings for *all* of its virtual nodes and use all of them when determining the shortest path.

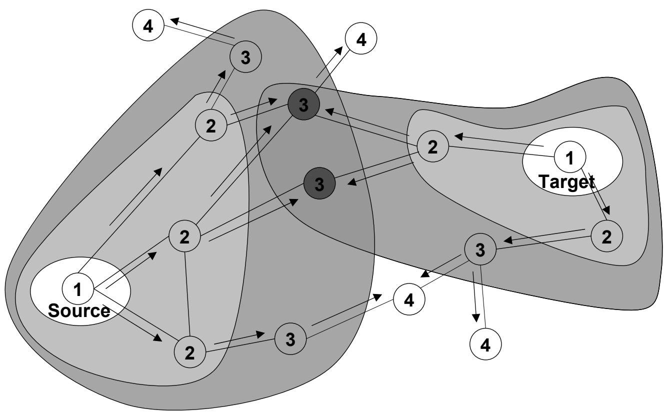


Figure 6.6: Key lookup with local expanding ring search from source and target

path knows immediately to which of its neighbors it should forward the query.[3] In essence, PathFinder uses source routing for key lookups.

Performance Analysis Note that the whole computation of the path happens locally on the source peer. The added cost comes in the form of additional memory usage and computation time. We now show that they are both completely negligible in practice.
We generated various PathFinder networks from 10^3 up to 10^8 nodes, with an average degree 20. In all of them, we performed 5000 arbitrary key lookups. It turned out that expanding rings of depth 3 or 4 (i.e., path length between 6 and 8) is sufficient for a successful key lookup, as shown in Figure 6.7. In the figure, the x-axis shows the path length and the y-axis shows the cumulative fraction of paths observed. For example, for 1 million nodes, the average path length is concentrated around 6. The theoretical average shortest path length for Erdös Renyi random graph with 1 million nodes and average degree 20 is 4.6, so the difference is only slight.

Figure 6.7 also shows that increasing the network size by a factor of 100 leads only to two additional hops for key lookups. The key lookup performance depends mainly on the average number of neighbors c and only slightly on the number of virtual nodes N. It has been shown that the average path length scales with $O(\ln(N)/\ln(c))$ [Bol01]. Increasing c leads to better key lookup performance, but higher average loads for the peers. Therefore, c can be adjusted to meet the needs

[3]An intermediate peer could also re-compute the path for the messages it receives, since the original source is not aware of the incoming links of the intermediate peer. However, we do not expect this to shorten the lookup path much, so in the interest of efficiency, we do not expect peers to do such re-computations.

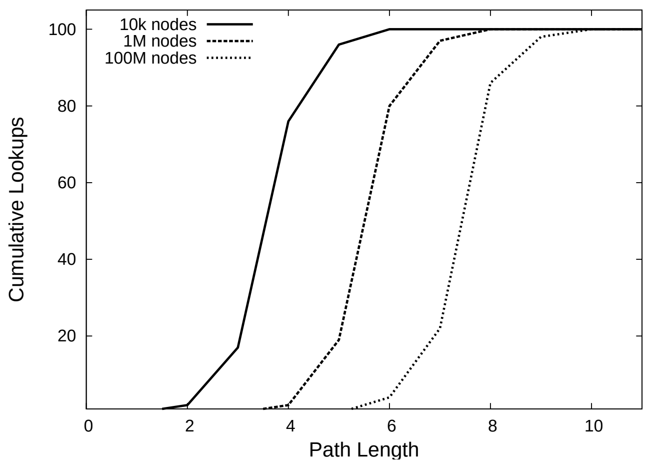


Figure 6.7: Distribution of path length for 5000 key lookups

of the concrete application scenario.
Assuming an average degree of 20 and ring depth of 3 (on both sides), peer *I* needs to compute $2 \cdot 20^3 = 16,000$ neighbors. Assuming neighbors are stored as 4-byte integers, this would require 64 kilobytes of memory. With ring depth 4 on both sides, this rises to 320,000 neighbors and about 1.3 megabytes of memory. The computation time for finding the match is typically on the order of 1 ms, even on laptops. In summary, the additional cost of calculating neighbors in memory does not represent any kind of a strain on memory nor does it take a long time. Furthermore, a peer performs these computations only for its own requests; requests from other peers can be simply forwarded using the path given in the message.

6.5.2.4 Searching with Complex Queries

PathFinder supports searching with complex queries with tunable success rates like BubbleStorm [TKLB07]. In fact, because both PathFinder and BubbleStorm are based on random graphs, PathFinder is fully compliant with the search mechanism of BubbleStorm. In BubbleStorm, both the data and queries are sent to some number of nodes, where the exact number of messages depends on how high the probability of finding the match is set. We can use the exact same algorithm for PathFinder for searching, and the reader is referred to [TKLB07] for details. The key advantage of PathFinder is that it allows for both key lookups and exhaustive search on the same network topology.

6.5.2.5 Node Join and Leave

We now describe the join and leave process for nodes in PathFinder. Its task is to:

- assign virtual nodes to peers.

- maintain outgoing links by the peer responsible for the corresponding virtual nodes.
- keep incoming links in the peer's state.

The purpose of the node join process is to integrate a new peer into the PathFinder overlay. The join process can also be used to automatically rebalance the network load, because virtual nodes may be unevenly distributed among the peers.

Node Join When a new peer A joins the network, it must contact a peer I that is already part of the network. The first virtual node assigned to A is calculated as a hash of its IP address. Peer I routes the join request using the key lookup procedure (Section 6.5.2.3) to the virtual node, which matches A's identifier. This node is handled by peer B. Peer B will hand 1 or more of its virtual nodes over to A and inform the neighbors about the new peer. In Section 6.5.2.7, we consider the case where a peer has to contact several other peers to find a free virtual node and how the network can adapt to such cases.

A successful join means that (i) a peer releases some of its assigned virtual nodes to the new peer, but keeps at least one virtual node for itself, and (ii) the new peer has successfully established connections to its neighbors. After the join process is completed, the new peer has some number of virtual nodes with an up-to-date neighbor table. It can be easily seen that the three invariants hold throughout the join procedure in the absence of node failures. We consider them in Section 6.5.2.6.

Node Leave When a peer leaves the network, it relinquishes its virtual nodes to its neighbors, who are responsible for establishing the connections to the right peers. Note that the peer can divide its virtual nodes among its neighbors evenly, as opposed to most existing DHTs, where the responsibilities are given to a single node.
Observation: A peer joining or leaving the network will cause on average $c + \ln(N)/\ln(c)$ messages.

The peer joining the network is routed to an arbitrary position determined by the hash function of its IP address. This costs one key lookup, which is on average $c + \ln(N)/\ln(c)$. The outgoing neighbors are directly transfered from the issuing node. Then, on average c incoming links transferred from the issuing node need to be updated, which causes c update messages.

6.5.2.6 Node Crash

A node crash is the sudden departure of a peer from the network without correctly following the departure protocol from above. A crash violates the invariants from Section 6.5.2.5, and neighbor tables are no longer correct.

Crash Detection and Peer Responsibility The absence of the failed node is recognized by its neighbors when they stop receiving keep-alive messages from it. The time for detecting a failed node depends on the interval used for keep-alive messages. When a peer detects a failed node, it locally calculates all neighbors of the failed node. The first peer in the neighbor list has to take over the missing node. Therefore, the peer, that has detected the failed node sends the first peer its own IP address and a message that it should start a recovery process. If the first peer in the neighbor list does not respond, it is replaced by the next peer in the list. Each peer in the neighbor list, which detects the missing node and still has not received a recovery message, also follows the above protocol. In case of concurrent attempts taking over the failed node, the node with the smallest position in the list continues the recovery process.

Topology Maintenance After a responsible peer has been determined, a new routing table for the failed node has to be generated. For this purpose, c key lookups have to be performed. After this step, which needs $c\frac{\ln(N)}{\ln(c)}$ hops on average, a routing table containing all outgoing links of the failed node is established.

The remaining incoming links are recovered automatically by the nodes pointing to the failed node. They notice timeouts of keep-alive pings sent to the failed node. Then, they update their routing tables by performing a regular key lookup to the failed node. This reveals the new responsible peer in $c\frac{\ln(N)}{\ln(c)}$ steps, on average. To reduce the load of the new responsible peer, we can also route to its neighbors, because they already know the IP address of the new responsible peer.

Observation: A failed peer causes $2c(\ln(N)/\ln(c))$ messages, on average, to repair the network overlay.

Robustness Analysis Figure 6.8 shows the message volume for a simulated PathFinder network with 5000 peers and different fractions of crashed peers. The x-axis shows the time from the crash in steps, where one step is at least one roundtrip time between two peers in real time. The y-axis shows the total amount of maintenance messages for each step in the whole system. The crash occurs at step 0, when all the failed nodes, 10–50% of the nodes, disappear immediately.

Moderate crashes (up to 30% crashed nodes) are healed in about 100 simulation steps, and even a crash of half the nodes is practically healed in 300 steps. The message load also remains reasonable, with about 36 messages per peer on average (half of 5000 peers crashed in worst case scenarios and a total number of messages in the system is under 18000). Considering the real-world time required to heal the system, if one step is 120 ms, then the system would heal itself in 12 s for the smaller crashes and in 36 s for the 50% crash. The main determining factor is the ability of peers to send all the required messages in a step, so the recovery could potentially take longer. However, recovery times are still very short, on the order of a few minutes at maximum. A similar performance was observed in BubbleStorm [TKLB07], which is also based on a random graph.

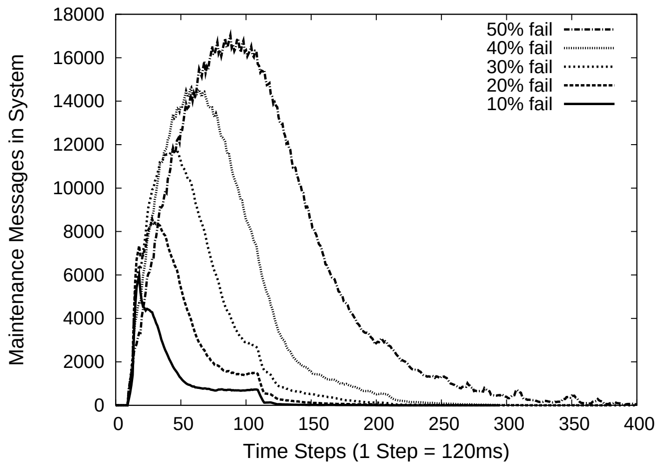


Figure 6.8: Repair costs for network with 5000 peers

6.5.2.7 Network Size Adaptation

In order to keep a reasonable ratio between peers and virtual nodes we must keep track of the number of peers in the network. We estimate the number of peers in the PathFinder overlay with the push-sum gossiping protocol [KDG03]. In [TLB07], this procedure was extended to make it applicable to peer-to-peer networks. Push-sum needs $\log(N) + \log(1/\varepsilon)$ steps to converge.

as virtual nodes can easily be transferred between peers, PathFinder is able to adapt itself to the current workload. Weaker peers may give up virtual nodes to stronger peers.

Recall the join process from Section 6.5.2.5. When a new peer joins the network, it takes some number of virtual nodes. However, when most of the peers in the network have only one virtual node, the joining node has to make several requests before finding a peer that still has available virtual nodes.

Adaptation Threshold The probability of finding a peer with more virtual nodes can be described through the hypergeometric distribution $f(1; N, m, c)$. A peer can ask any of its c neighbors for virtual nodes, and success depends on the number of peers with more than 1 virtual nodes within the network (m). As the ratio of peers in the network to average number of neighbors (i.e., N/c) is higher than 0.95, the process can also be approximated through the binomial distribution. A ratio of virtual nodes to peers of 1.15 establishes a successful join for 80% of requests within 10 hops ($B(10, 0.15)$) and over 96% after 20 hops.
We aim to keep the cost for a join reasonable. Therefore, when the threshold of 1.15 virtual nodes per peer is reached, the network starts a transition phase in which the amount of virtual nodes is doubled.

Transition Phase Assume that a virtual node w notices that the above threshold is reached. Then w starts the transition phase by generating two new virtual nodes, w_1 and w_2, adding a new bit to the left of its ID. One of w_1 and w_2 is given bit 0, and the other bit, 1.
Now, w_1 and w_2 have to calculate their neighbors. They proceed as in Section 6.5.2.1, but use the new ID space. The IDs of the calculated neighbors also have one extra bit. Let x_1 be one of the neighbors of w_1. At this moment w_1 still does not have a routing table and cannot route to x_1. Therefore, w_1 disregards the left-most bit of x_1's ID and uses the routing table of w to determine the node x corresponding to this ID.

The peer responsible for x will be responsible for x_1 when the transition phase is over. This is because the ID of x_1 is the ID of x, with one bit added to the left. Therefore, w_1 now adds this peer to its new routing table. If x has not yet started the transition phase, then it does so now. When the above procedure is carried for all new neighbors of w_1 and w_2, the transition phase for w is completed. When the pushsum protocol confirms that the number of virtual nodes in the network has been doubled, i.e., all virtual nodes have completed the transition, the old routing tables are abandoned.

We expect the transition phase to be a rare event in an operational network. Furthermore, it is not a time-sensitive process, so in a large network, it could possibly be artificially stretched out to hours or days, since both the old and new networks are operational during the transition.

Network Shrinking Shrinking the ID space is also possible and works similarly. If the virtual node to peer ratio is higher than, e.g., $4/1$, the amount of virtual nodes can be reduced in order to improve lookup performance. The procedure is the reverse of expanding the network, whereby peers will strip one bit from left of their IDs. In case two different peers map to the same shorter ID (quite likely), the one who had bit 0 as the first bit takes over. The peer who had bit 1 must find another virtual node to take over. If we set the threshold of shrinking high enough (e.g., $4/1$), then there are enough virtual nodes for all peers, but a peer might have to search for a free one.

Note that routing performance does not substantially change if the network grows or shrinks by a factor of 2, which is a property of Erdös-Rényi random graphs.

6.5.3 DHT Comparison and Analysis

Most DHT overlays provide the same functionality, because they all support the common interface for key-based routing (cf. Section 3.2). The main differences between the various DHT implementations are average lookup path length, resilience to failures, and load balancing. In this section, we compare PathFinder to other DHTs presented in the literature. We perform the comparisons both with simulations and analytically for networks that are too large to simulate (up to 100 million virtual nodes). All simulations were performed with the P2P simulator PlanetSim [Gea05] in combination with FRCS (cf. Chapter 4), the P2P overlays were either already available within PlanetSim or were built and tested by the authors. All overlays perform as expected from the evaluations of the related publications.

Chord The lookup path length of Chord is well studied [ZZ03].
It is asymptotically:
$L_{avg}(N) = \frac{1}{(1+d)\log(1+d)-d\log(d)} \log N$, where N is the number of peers in the network. The parameter d tunes the finger density. Usually, Chord has finger density $d = 1$, and therefore, $L_{avg} = \frac{\log(N)}{2}$. The maximum path length of Chord is $\frac{\log(N)}{\log(1+d)}$. The average path length of PathFinder is $\frac{\log(N)}{\log(c)}$, where c is the average number of neighbors. In other words, even for relatively small c, PathFinder has a much shorter path length than Chord.

Pastry The path length of the Pastry model can be estimated by $\lceil \log_{2^b}(N) \rceil$ [RD01b], where b is a tunable parameter. The authors recommend $b = 4$. In this model, there are $\log_{2^b}(N)$ levels and $2^b - 1$ neighbors per level. This results in 96 neighbors for a network of 50 million peers. PathFinder achieves comparable results with only $c = 20$ neighbors, on average. For $c = 50$, the average path length of PathFinder drops to 2/3 the path length of Pastry. Theoretically, PathFinder should achieve Pastry's performance for $c = 16$ (for $b = 4$). Given that our results show that PathFinder matches Pastry for $c = 20$, we suspect that Pastry's real-world performance for large networks would not be quite as good as the theoretical model leads us to expect.

Symphony The Symphony overlay is based on a small world graph. This leads to key lookups in $O(\frac{\log^2(N)}{k})$ hops [MBR03]. The variable k refers only to long distance links. The actual amount of neighbors is indeed much higher [MBR03].

CAN The diameter of CAN is $\frac{1}{2}dN^{\frac{1}{d}}$ with a degree for each node $2d$, with a fixed d. With large d, the distribution of path length becomes Gaussian, like Chord [LKRG03].

Butterfly The butterfly network has a nearly optimal diameter and average path length. The average distance in a butterfly network is given by [HK91]: $\mu_d \approx \frac{3\log_k(N)}{2}$. An implementation of the butterfly network, Viceroy [MNR02], has an average path length of $3\log_2(N)$.

PathFinder The theoretical average path length of PathFinder is $L = \frac{\log(N)}{\log(c)}$, where c is the average number of neighbors per node. This is a property of its underlying random graph.

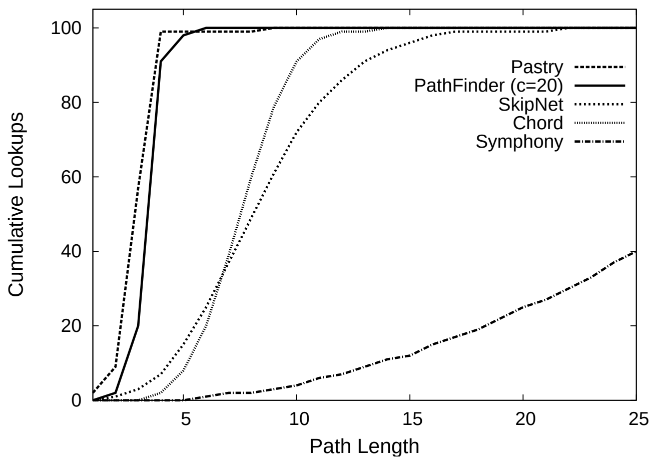


Figure 6.9: Average number of hops for 5000 key lookups in different DHTs

In summary, all DHTs have a path length proportional to $O(\log(N))$, with some individual multipliers or other factors. In this sense, there is not much difference between them.

Practical Evaluation We used simulations to evaluate the practical effects of the individual factors. We compared PathFinder with Pastry, Chord, Symphony, and SkipNet [HJS+03]. Figure 6.9 shows the results for a 20000 node network. We performed 5000 lookups from random nodes to random nodes and measured the number of hops it took for each of the DHTs to find the object. We plot the number of hops on the x-axis, and the y-axis shows the cumulative fraction of requests that were successful with the given number of hops.

Pastry and PathFinder perform similarly, with the maximum number of hops being around 4. Chord and SkipNet perform more poorly, needing on average 7 additional hops. Symphony's performance is extremely poor, with some lookups requiring up to 40 hops (not shown in the figure). CAN and Viceroy had even poorer performance and were thus dropped from further comparisons.

We also performed an analytical comparison, using the equations from literature summarized above. Our goal is to gain some idea about how well the different networks would scale to large networks with hundreds of millions of peers. We compared PathFinder with Pastry and Chord. We ignored Symphony because of its poor performance in the earlier experiment, and SkipNet because of the lack of a well-understood analytical model for its performance. We also used a DeBruijn-graph, because they are known to have optimal diameter.

Figure 6.10 shows the results. Note that PathFinder numbers *are actual numbers, not analytical*. For the other case, we had to resort to analytical modeling due to lack of suitable simulation environments. In Figure 6.10, x-axis shows the size of

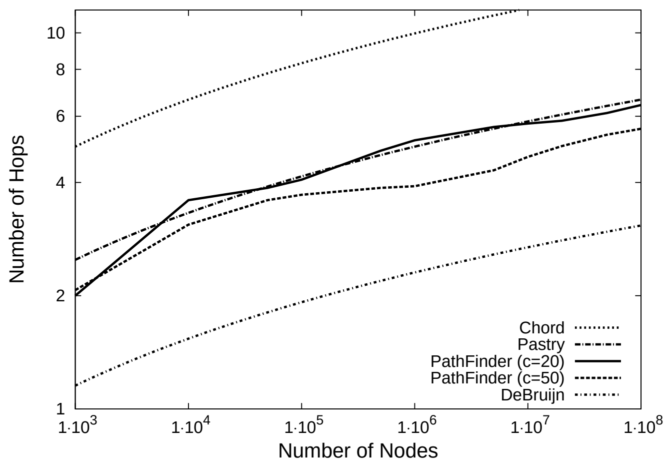


Figure 6.10: Average number of hops for different DHTs measured analytically. Numbers for PathFinder are simulated.

the system, and the y-axis shows the average path length in the system. As was to be expected based on Figure 6.9, Chord's performance is clearly lower than that of Pastry and PathFinder. Pastry and PathFinder are very similar in performance for $c = 20$. Making $c = 50$ gives PathFinder a similarly sized neighbor tables as Pastry, and yields about 1 hop less in systems of 100 million nodes. The line for the DeBruijn graph shows the absolute shortest path for a PathFinder network with $c = 20$, and PathFinder is only slightly over 1 hop longer.

6.5.4 Resilience Against Failures

High churn rates [SGG02] are common in peer-to-peer networks. Therefore, alternative paths may be needed to find a particular node. This is where PathFinder benefits from its randomized structure. There are always as many alternative independent routes between any two nodes as the minimum of their degrees [GHW07].

The question then becomes, how difficult is it to find a valid alternative path? Note that a peer A is not aware if there is a failed node on its path to peer B. It is only when A tries to reach B, that A notices that it has to search for an alternative path. However, due to the high number of alternative independent paths (i.e., paths that have only common start and end nodes) between each two nodes, the number of retries required is very small.

We evaluated this by taking a network of 50000 virtual nodes and allowed some percentage of them fail. We then checked all pairs of remaining nodes and performed a key lookup using the procedure from Section 6.5.2.3. We recorded how many times we had to re-try the path because of the failures. The results are shown in Figure 6.11. One observes that 5 tries are sufficient to connect over 90% of the node pairs, even when 25% of the nodes in the network have failed. In a case

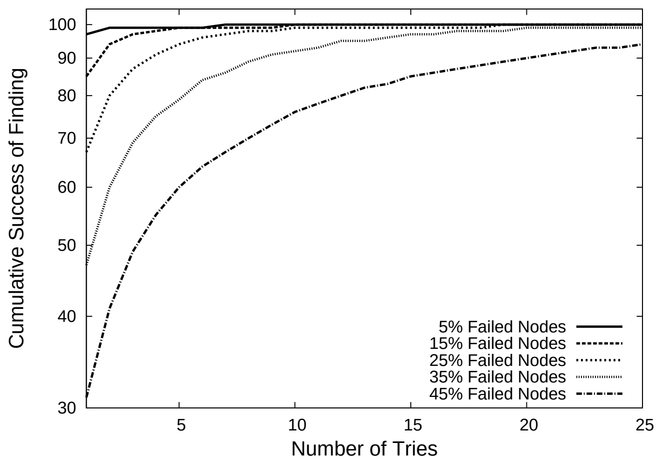


Figure 6.11: Required tries in order to find an alternative path between each couple of nodes in a network with 50000 virtual nodes.

when almost half of the nodes have failed, then 12 tries lead to a successful lookup in 80% of the cases.

In the test, we performed no maintenance in the overlay. After maintenance, the number of retries drops to zero.

We have seen that between each pair of nodes in our network, there are enough alternative independent paths, so that only a few attempts are needed, even in the case of massive node crashes. However, there is still one crucial question remaining: How does the average path length change with the number of failed nodes?

Recall that the underlying structure of our network is an Erdös Rényi random graph. That is, for a large number of nodes and fixed average node degree c, the degree distribution is well approximated by a Poisson distribution:

$$P(k) = \frac{c^k}{k!}e^{-c}$$

where $P(k)$ is the probability that a node has degree k. From here it follows that the diameter of the network is around $D = \ln(N)/\ln(c)$, where N is the number of nodes in the network [Bol01]. Furthermore, and more interestingly for us, the average shortest path length L is almost equal to the diameter and is given by $\ln(N)/\ln(c)$ [Bol01].

Thus, given the number of nodes in the network N and c, we can always compute L. A property of Poisson-distributed networks is that L will change only slightly, even if half of the nodes in the network fail. In Figure 6.12 we plot the average shortest path length for $N = 50000$ and different values of c. The x-axis shows the fraction of failed nodes, and the y-axis shows the average shortest path length. As we can see, the increases are minimal, even if half the nodes suddenly fail.

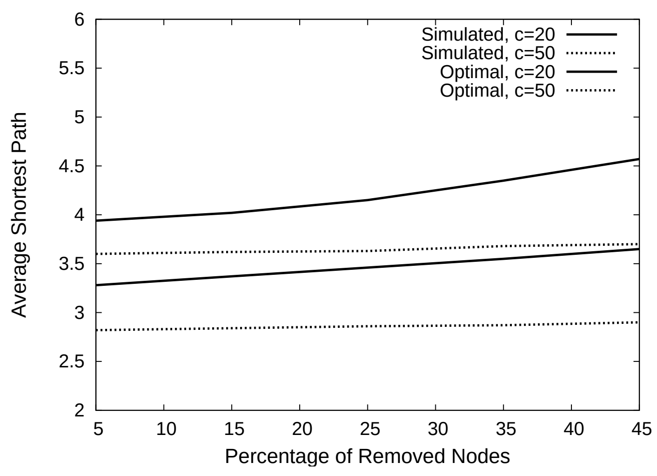


Figure 6.12: Actual path length and optimal path length with node crashes

Even though the average shortest path length changes only minimally, it is not guaranteed that our key lookup procedure will find a good alternative path. We evaluated the probability densitiy for key lookups under node failures, and find that the maximum number of hops only increases from 6 to 7 for $N = 50000$ and $c = 20$.

6.5.5 Security and Other Issues

In terms of security, PathFinder is comparable to other DHTs presented in the literature. Peers receive their virtual node IDs as a hash of their IP addresses, which is the same as in other DHTs. Note that an attacker with access to a large pool of IP addresses can "choose" where to place herself. This is a common weakness of all DHTs.

If a malicious peer drops messages that are routed through it, the sending peer will eventually notice this because it does not receive a reply. Recall from Section 6.5.4 that there are as many node-disjoint paths between the sender and receiver as the minimum of their degrees. Thus, the sender can simply try again using a different path. The attacker is unlikely to be present on all the alternative paths. A simple approach to detecting malicious peers is to send a message always using two different paths.[4] A comparison of the results will immediately reveal whether the message or reply has been modified.

One approach for handling malicious peers deleting information stored on them is to provide replicas, whereby an object is always handled by several independent peers. In PathFinder we can use an approach similar to Tapestry [ZKJ01] and use a constant sequence of salt when hashing the object names.

Because PathFinder builds an overlay network with randomly selected neighbors,

[4]Given the overhead caused by this, this strategy would only make sense if the number of malicious peers is large.

it is very likely that initially, it does not match well with the underlying IP network. Other DHTs suffer from the same problem to a degree as well because their routing algorithms also require contacting arbitrary peers in the network. DHT routing is typically optimized by selecting neighbors with small RTTs, with the goal of reducing the overall path latency. PathFinder supports dynamic transfer of virtual nodes, which does alter the neighborset of the nodes. Although, it might not start with an optimal adaptation, the network adapts to the underlying topology over time. When a peer needs to route a message in PathFinder it computes the shortest path as described in Section 6.5.2.3. If the peer notices that another of its neighbors has *considerably* shorter RTT than the "correct" peer, it can send the message to that neighbor. This may increase the path length in terms of hops, but might reduce the latency. An evaluation of this scheme is part of our future work.

6.6 Summary

In this Chapter, we have presented PathFinder, an overlay that combines efficient exhaustive search and efficient key lookups in the same overlay network. Being able to combine search and lookups in the same overlay is very desirable, because it allows efficient and overhead-free implementation of natural usage patterns. PathFinder is the first overlay to combine search and lookups in an efficient manner.

Approach	**Probabilistic Search**	**Key Lookup**	**Access Delay**	**Churn Resistance**
BubbleStorm	●	○	Probabilistic	High
Chord	○	●	High	Low
Pastry	○	●	Low	Low
Viceroy	○	●	Lowest	Low
PathFinder	●	●	Low	Low/High

Table 6.2: Comparison of PathFinder with state-of-the-art approaches

Table 6.2 presents a direct comparison with state-of-the-art approaches. PathFinder offers desirable features for the volatile environments found in first response situations. Being able to combine search and lookups in the same overlay greatly satisfies requirement R5 (active search) and furthermore creates a flexible approach for all kinds of use cases. In addition, we have shown in Section 6.5.5 that PathFinder provides a means for R4 (resource awareness) and R10 (allow horizontal and vertical message transmissions) by not assuming any predefined communication structure.

Our results show that PathFinder has performance comparable or better to existing overlays (average path length of an object lookup grows with $ln(N)/ln(c)$, where

N is the number of peers and c is the average number of neighbors per peer). It scales easily to hundreds of millions of nodes, and key lookup performance is in large networks better than in existing DHTs. Because PathFinder is based on a random graph, we are able to use existing search mechanisms (e.g., BubbleStorm) to enable efficient exhaustive search. We have shown the performance of PathFinder both through simulations and analytical means. DHTs, although just as scalable on paper, have not been demonstrated in dynamic networks of that size, so their true scalability is still an open question.
Furthermore, PathFinder has a *constant degree*, so the state maintained by peers is constant. In most DHTs, the state increases logarithmically with the network size.

In situations of high churn, PathFinder may therefore not be able to provide key lookups. For example, in Chapter 6 we evaluated the Tapestry overlay behavior in volatile environments and found that despite its robustness against network partitioning, its lookup success rate is sensitive to churn. PathFinder still provides probabilistic search in volatile environments, as long as the network is not partitioned. In stable environments, both the lookup and exhaustive search functionalities are provided.
If the overlay network is in a stable state and therefore able to provide key lookup, we propose adding a two-dimensional CAN network in order to offer locality preserving search capabilities. The costs per node are reasonable, because each node just needs to maintain approximately four additional links.

Chapter 7

P2P Overlay Network Layer: Application-Level Multicast

Our Distributed Tree Construction (DTC) approach enables structured P2P networks to perform prefix search, range queries, and multicast optimally. It achieves this by creating a spanning tree over the peers in the search area, using only information available locally on each peer. We evaluate the performance of DTC by comparing its performance to existing application-level multicast solutions; we show that DTC sends at least 30% fewer messages than common solutions.

The remainder of this Section is structured as follows: Section 7.1 presents the general approach and requirements. In Section 7.4, we reveal how we construct the distributed spanning tree and prove its properties. Section 7.5 shows an example of how to build a prefix search using hash functions and our tree construction algorithm. Section 7.6 evaluates our algorithm on different DHTs and compares its performance against existing algorithms. In Section 7.7, we discuss the robustness of our algorithm and present mechanisms for improving its resilience against malicious peers. Section 5.2 discusses related work. Finally, Section 7.8 concludes the Chapter.

7.1 Introduction

We tackle the problem of implementing a prefix search on a structured P2P network. Our algorithm selects a subset of the overlay network and creates a spanning tree for that subset rooted at any of the peers in that set. Typical applications that benefit from prefix search are all systems that need to handle structured data. Structured data is very commonly used in many applications. For example storing data based on geographical location (e.g., tourist information) or any kind of classification systems lend themselves readily to our algorithm.
The more attributes are available in the user provided search term (i.e., the more precise the user's query), the less nodes will be queried. In the case of a user

performing a very precise search, the query will reduce itself to a standard DHT key lookup. Because we create a spanning tree, our solution is not limited to implementing prefix search; other applications, such as multicast and broadcast, can also be implemented with our algorithm. Multicast (and also broadcast) is extremely useful in cases where the DHT is built according to some specific criteria (as opposed to a standard hash function; see Section 7.5), as it enables us to reach a given set of nodes with minimal overhead.

The key feature of our algorithm is that it creates a spanning tree *without any communication between the peers*, using only local information available to every participating peer. As it is a tree, we are guaranteed that only the minimum number of messages must be sent. These key features set our algorithm apart from previous work, such as the application-level multicast proposed by Ratnasamy et al. [RHKS01] or SplitStream [Cea03b]. Previous work typically either has a high number of duplicate messages, requires additional coordination traffic, or requires a second overlay network. Additionally, range and origin of the spanning tree can freely be set to any value and any peer.

Our distributed tree construction algorithm (DTC) presented in Section 7.4 has none of the shortcomings of the previous algorithms. It can be used on top of most existing DHTs, and it works using information available at each peer about the peer's neighbors. As we show in Section 7.4, this standard information maintained by the DHT is sufficient to span a tree.
As our evaluation shows, our DTC algorithm yields optimal performance in terms of messages sent. We also show that the overhead of duplicate messages in existing solutions is at least 30%, but can in several cases be up to 250%.

7.2 Requirements

The main task of the 'Overlay Network' layer is to provide peer discovery and search functionality (cf. Section 3). Our contribution, the Application Layer Multicast, may be used for a wide set of applications.
The approach should provide strong mechanisms for R3 (locality awareness) and R5 (active search) as well.

R1 (Multicast) First Responders need to send messages to a distinct group of people or to a specific location (cf. Section 2.2).

R3 (Locality Awareness) The overlay needs to pass location information from the 'Connection Topology' layer to the 'Application' layer. The multicast mechanism may provide a means for accessing groups of objects with proximate locations, which allows for a wider set of location-based services in the 'Application' layer.

R5 (Active Search) The efficiency of active search, especially in the case of key lookup, can be significantly enhanced using multicast mechanisms. Storing objects with similar properties (e.g., objects with a common prefix, similar location information or adjacent numeric identifiers) close to each other (see Section 7.5.1) enables efficient searching for objects with common properties.

7.3 Related Work

Flooding approaches for P2P networks in general have been extensively investigated [JGZ03, CRB+03, TKLB07]. While simple flooding approaches may apply to unstructured networks, DHT networks can take advantage of the structured neighbor lists and reduce the number of duplicate messages.

Multicast in CAN In the case of CAN, work by Ratnasamy et al. [RHKS01] implemented an application-level multicast on top of CAN. As our evaluation shows, this approach can have a significant overhead, even in the best case, will have an overhead of about 32%. As we have demonstrated, the overhead is a function of the network size and CAN dimensions, and in smaller search areas, the overhead can grow considerably higher. An improved version of ALM is described in [Cea03a]. They reduce duplicate messages, but duplicates may still occur, especially in the case of uneven zone sizes within the CAN overlay.

Range Queries and Prefix Search One approach for Range Queries within DHTs [RHRS04] is to form a tree similar to our quad tree approach. The difference is that they explicitly use a DHT in order to store a generated tree for accessing objects. Our approach, in contrast, does not use the DHT interface to generate a tree; it directly restructures the mapping from object to nodes. The advantage is that our solution enables prefix searching without any overhead.

Another approach for prefix search is presented in [JY06] by Joung et al. The idea is that the prefixes build a sub-hypercube within a hypercube, which spans a binomial tree [JFY05]. In general, a binomial tree can be traversed without generating duplicate messages. On the downside, the hypercube needs to have as many dimensions as bits in the ID space. Given a common 160 bit ID space, this would lead to 160 dimensions, which in turn leads to 160 neighbors per node for a binomial tree. Further overhead is generated, because each of the 2^{160} bits needs to be addressable. Therefore, a prohibitively large number of logical nodes is necessary in smaller networks. In contrast, our DTC algorithm achieves the same optimal number of messages with *no overhead* over a standard DHT.

A different approach for prefix searching [AS03c] uses a combination of different overlay networks, one for file management and another for site (participants) management. The combination of a DHT-based overlay (e.g., CAN) with another independent overlay enables prefix search as well, but at much higher costs. Our solution does not require any additional maintenance costs or additional links; all required knowledge can be calculated locally. While our work concentrates on raw address coding and message distribution, additional performance optimizations applied to routing for keyword search in DHT-based P2P networks [RV03b, Gna02] can be used with DTC, as they are complementary to our solution. We do not alter the underlying routing scheme, and therefore all optimizations used for the DHT apply to DTC as well.

Applying multi-attribute range queries to current DHTs has been done by choosing hubs [BAS04]. The idea of hubs, which are responsible for one attribute each, is independent from the underlying DHT. A hub can be seen as a separate overlay for each attribute. Our prefix and multicast approach can also be applied to hubs, which would lead to an optimized ID space for each attribute. On the one hand fewer nodes would need to be queried; on the other hand, a separate overlay network must be maintained for each attribute.

7.4 Distributed Tree Construction

In unstructured networks, more than 70% of the messages are redundant, even with a moderate TTL [JGZ03]. This high overhead is one of the main reasons for the poor scalability of unstructured networks. With our *distributed tree construction* algorithm (DTC), we are able to eliminate this overhead. Note that DTC builds on top of a standard, structured overlay network. We first discuss the requirements, on the structured overlay, and then present the DTC algorithm with optimality proofs. Finally, in Section 7.4.3, we discuss different applications and how to build them using DTC.

The idea behind DTC is to build a spanning tree to connect all the nodes we want to search. When a query is sent from the root, every node in the tree receives it exactly once. The challenge lies in constructing the tree without any overhead and using only local information available in each node.

7.4.1 Structured Overlay

Our DTC algorithm works on any structured overlay (a.k.a. distributed hash table), which fulfills the property that every node knows *all* of its immediate neighbors in the overlay hash space. Networks like Chord [I. 01], CAN [RFH^{+}01b] and VoroNet [BKMR07] obviously fulfill this property. In case of Chord, the critical information is knowing the successor and, for CAN, knowing all neighbors in all coordinate directions. In the case of Pastry [RD01a], the condition is fulfilled since the leaf sets of all nodes always contain the closest neighbors in the hash space. Although Tapestry [Zea04] is very similar to Pastry, it does not have the equivalent of the leaf set and thus might not be suitable without modifications.

Kademlia [MM02] also fulfills the required condition, as the buckets for the shorter distances contain the closest nodes in the hash space, and they should be complete. In the remainder of this chapter, we consider only Chord and CAN as overlay networks. Although the principle of DTC in both networks is the same, differences in the overlay structures lead to performance differences (see Section 7.6).

7.4.1.1 DTC Algorithm

In the distributed tree construction algorithm, we start spanning the tree from a point in the overlay, and expand from that point through the overlay according to the overlay routing. The area of the overlay which the spanning tree is supposed to cover is explicitly defined. The information about the root of the tree and the area are sufficient to construct the spanning tree in a purely distributed manner. We will now show how this can be done in Chord and CAN.

Note that even though the algorithm constructs a spanning tree, no peer has a complete view of the tree. The tree is always constructed on-demand by having the root send a message, which constructs the tree as it is passed through the peers. Thus, when we say below that a peer adds some other peers to the tree as its children, it means in practice that the peer in question forwards a message to the other peers.

Example: Chord

In the case of Chord, the area is an arc on the Chord ring and the root of the tree is the first node on the arc. The simplest solution is adding nodes to the tree along the chain of successors until the end of the arc has been reached. However, this is not very efficient for large areas. Instead, we should use the fingers to create shortcuts and to broaden the spanning tree. The root of the tree selects all of its fingers that are in the area as its children. Each of them will recursively perform the same operation until all peers in the same area (the arc) have been included in the tree. Any of the peers can easily determine which of its fingers it should include, as it knows the root and the length of the arc. The successor of the last point of the arc[1] *is* part of the tree, because the last points in the arc might contain objects that are stored on their successor.

Example: CAN

In the case of CAN, the area is a convex area of the d-dimensional coordinate space, with the root somewhere in this area. The restriction to convex search areas is imposed by our algorithm. Non-convex areas can be searched by splitting the search into non-overlapping convex searches that cover the desired area. The root first adds its immediate neighbors ($2d$ neighbors in a d-dimensional CAN), which then continue adding their neighbors, according to the rules defined below. As in the Chord-case above, the information about the root of the tree and the area it is supposed to cover are available to the peers. The area can be defined either with simply the radius of the area, or by specifying for each dimension separately how far the area reaches in that dimension. As mentioned above, the only restriction on

[1]Often the first node after the arc.

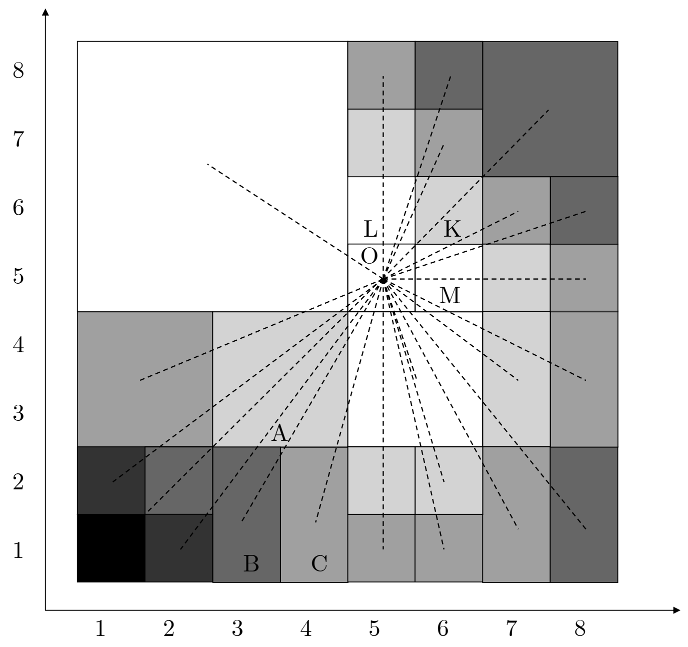


Figure 7.1: Example spanning tree of a CAN

the search area is that it must be convex.

We assume every node knows the following:

- Size of the zone of each neighbor (maintained by standard CAN routines)
- Root of the tree and the area it is supposed to cover (available in the message that is used to create the tree)

Figure 7.1 shows an example of a spanning tree of a two-dimensional CAN. The tree is rooted at the white zone marked *O* at coordinates (5, 5). The other white zones are the children of the root, and the levels of the tree are shown in increasingly darker shades of gray. The result is a spanning tree consisting of all nodes within the CAN overlay.

The tree is constructed as follows. When a peer *X* receives the query, it computes for every one of its neighbors the vector from the center of the root's zone to the center of the neighbor's zone. If that vector intersects the common border surface between *X* and the neighbor, then *X* should add that neighbor as its child. The vector must intersect the common border surface between the two nodes; it is not sufficient for the vector to pass through *X*'s zone. Consider the third zone from the left on the bottom row in Figure 7.1, marked *B*. This zone has one light gray neighbor *A* on the top and one medium gray *C* neighbor to the right. The vector from the root passes through both of these neighbors, but the one on the right (marked *C*) is the parent of node *B*.

It is important to note that every node is able to compute the vectors and determine whether it should add any of its neighbors as children (and thus forward the message) by using only information available locally through normal overlay communications. No coordination between nodes is needed, nor is any additional traffic generated.

7.4.2 Proof of Optimality

We prove the following properties of our DTC algorithm:

1. The DTC algorithm creates a spanning tree over the area.
2. The depth of the tree is proportional to the message complexity of the underlying DHT.

For the simple version of a Chord-based DTC (i.e., every node passes the query to its successor), the first property is obvious. The proof of the first property with fingers is also straight-forward and is omitted from this discussion.

In the following, we will demonstrate the properties for a CAN-based DTC. We make the simplifying assumption that the overlay network is able to heal itself under churn without loss of messages. Note that this assumption explicitly *allows* churn, as long as changes to the overlay structure are performed in a locally atomic manner and no node departs between receiving and forwarding a query. We also assume that the area over which the tree is to be spanned is convex.

Theorem 1 *All nodes are added to the tree at least once.* □

Proof For every zone Z in the convex area, there is a single vector that connects the center points of that zone with the zone of the root of the tree. Starting from zone Z, the vector determines Y, a neighbor of Z that will add Z as its child. Considering zone Y, we can draw the vector between the center of Y and the starting zone, which determines a zone X and neighbor of Y which adds Y as a child. Continuing in a similar manner, we arrive at the zone of the root of the tree. Thus, we are able to find a chain of zones that leads us from the root zone to zone Z. Thus, all nodes have at least one path from the root, i.e., are part of the tree.

In some cases, it is possible that the vector between the root and a zone Z does not pass through any direct CAN neighbor of Z. For example, in Figure 7.1, the vector between the root and the zone marked K passes directly through the corner point of the two zones. Depending on how the ownership of edges is defined, it is possible that there is no neighbor through whose zone the vector passes on its way from the root to K. (Note that regardless of how the ownership of edges is defined, it is always possible to construct the zones such that this problem persists.) In general, this issue arises when two zones share up to $(d-2)$ dimensions in a d-dimensional CAN (e.g., a point in 2-dimensional CAN and a point or a line in a 3-dimensional CAN). In this case, the forwarding algorithm does not reach all nodes. We have defined the following tie breaker for these cases.

The Tie Breaker We use the following rule to determine how to construct the tree in the above case. The two problematic zones differ in at least 2 and up to *d* dimensions. We order the dimensions beforehand. The forwarding path should be such that the smallest dimensions with differences are used first. The length of the tie breaker path will be the same as the number of dimensions in which the two problem zones differ (i.e., between 2 and *d*). Note that none of the nodes on the path would normally forward the query, but *all of them* are able to compute locally that they are part of the tie breaker procedure and are able to perform their duties correctly. In the example of Figure 7.1, the tie breaker would mean that *M* is the node responsible for adding *K* as its child, because the x-coordinate is considered before the y-coordinate. ■

Theorem 2 *All nodes are added to the tree at most once.* □

Proof We prove this by contradiction. If a node *A* were to be added to the tree twice, this would imply that two of its neighbors would think that the vector between *A* and the root passes through their zones. This is clearly impossible, because the responsibility is defined by the vector and the vector between *A*'s center point and the center of the root's zone intersects only one of the borders between *A* and its neighbors. Thus, *A* can be added to the tree by at most one of its neighbors. ■

Theorems 1 and 2 prove that every node in the area is added to the tree exactly once, and thus the DTC algorithm creates a spanning tree rooted at the root zone and covering all the nodes in the area. Note that the spanning tree is only *a* spanning tree; it might not be the minimal spanning tree (but this property is not a requirement of the applications we are considering).

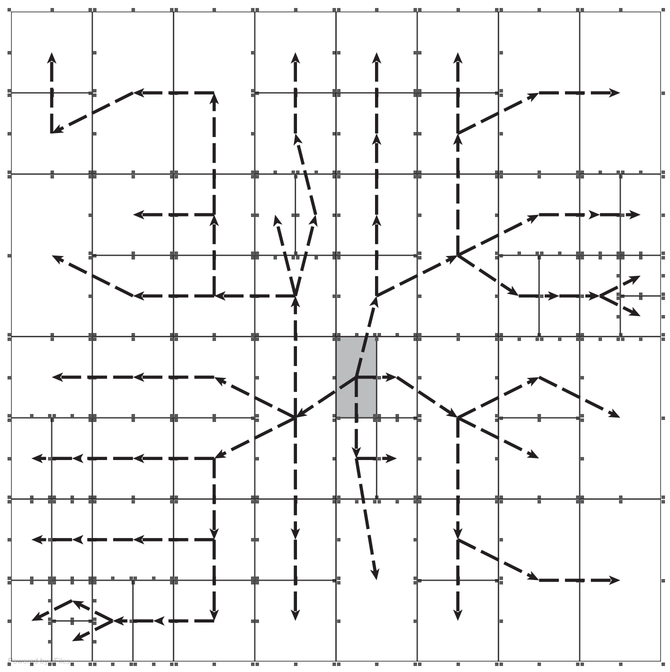
Figure 7.2 shows a larger example of how the DTC algorithm constructs the spanning tree. In Figure 7.2(a), we show the spanning tree overlaid on the underlying CAN topology and show which peers add which other peers as their children. Figure 7.2(b) shows only the resulting spanning tree.

Theorem 3 *The depth of the spanning tree is proportional to message complexity of the underlying DHT.* □

We prove this for both Chord- and CAN-based DTCs. In the case of a Chord-based DTC, we can construct the spanning tree using fingers, as mentioned above. The links in the Chord-DTC spanning tree are determined by the fingers and successor pointers of the nodes.

The claim of the theorem refers to the depth of the spanning tree. Note that every path in the spanning tree is a legal DHT-routing path between the root and the chosen node.

Proof (Proof for a Chord-based system) In a Chord-based system with fingers, the spanning tree is simply a mapping from the fingers and successor pointers to the nodes, and every path in the tree exactly corresponds to the routing path that the standard Chord routing would take to reach that node. Hence, the depth of the tree is $O(\log(N))$. ■


(a) Spanning tree overlaid on CAN

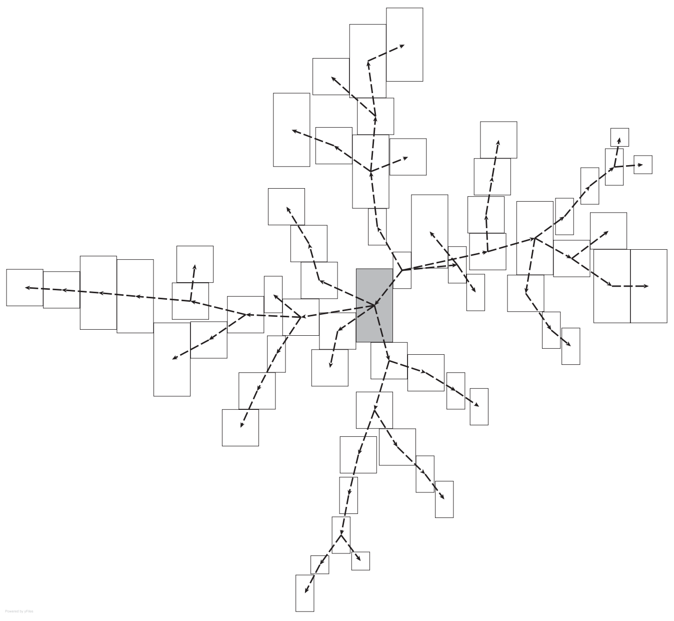

(b) Spanning tree

Figure 7.2: Example of DTC-constructed tree

PROOF (PROOF FOR A CAN-BASED SYSTEM) In a CAN-based system, every hop is also a legal CAN routing hop, but not necessarily a hop that the standard greedy CAN routing would take in a given situation. The DTC algorithm always follows the vectors, but the greedy CAN routing might take shortcuts over large zones. Nevertheless, the length of the path in the spanning tree is still $O(\sqrt[d]{n})$. ■

7.4.3 Sample Applications

The ability to construct a spanning tree from any point in the DHT is very powerful, and allows us to develop many different kinds of applications. We now discuss some of the applications that can be built with DTC.

Prefix Search in DHTs

As mentioned in the introduction, searching in DHTs is extremely inefficient. With DTC and the hashing scheme from Section 7.5, we are able to implement a prefix search over a freely selectable prefix, with only the minimum number of messages needed. We achieve this through a slight modification of how content is mapped on the nodes (see below) and by spanning a tree over a pre-determined area.

As discussed in the introduction, many applications can benefit from prefix searches. In particular, applications that use any type of structured data lend themselves readily to prefix searches. Structured data are very common and easily map to hierarchical concepts that are used in many different applications.

Group Communication Primitives

If the root sends a message along the tree (as is done during the tree construction), then every node in the area will receive the message exactly once. By tuning the area that the tree spans, we can easily define different multicast groups and reach them with the minimum number of messages. (As the comparison in Section 7.6 shows, the application-level multicast on CAN [RHKS01] has significant overhead compared to our DTC-based approach.) It is even possible to let the area be the complete hash space of the DHT, in which case we have an optimal broadcast mechanism.

An important point to keep in mind when designing applications running with DTC is whether there will be feedback to the root of the tree. In other words, a search requires an answer, i.e., all the nodes in the tree with matching content should answer. In contrast, a multicast or a broadcast might not require any acknowledgement from the receivers. The presence or absence of feedback is thus application-dependent, and we will return to this issue in Section 7.7.

7.5 Prefix Search in a DHT

We now present a way of mapping objects to peers in a DHT, such that the DTC algorithm allows us to perform prefix searches on the DHT. We achieve this by placing all objects matching a prefix in a certain area of the DHT, and the prefix search corresponds simply to spanning a tree with the DTC algorithm over this area. As an example of how to implement a prefix search, we consider the case of a d-dimensional CAN. (Note that other DHTs mentioned as candidates in Section 7.4.1

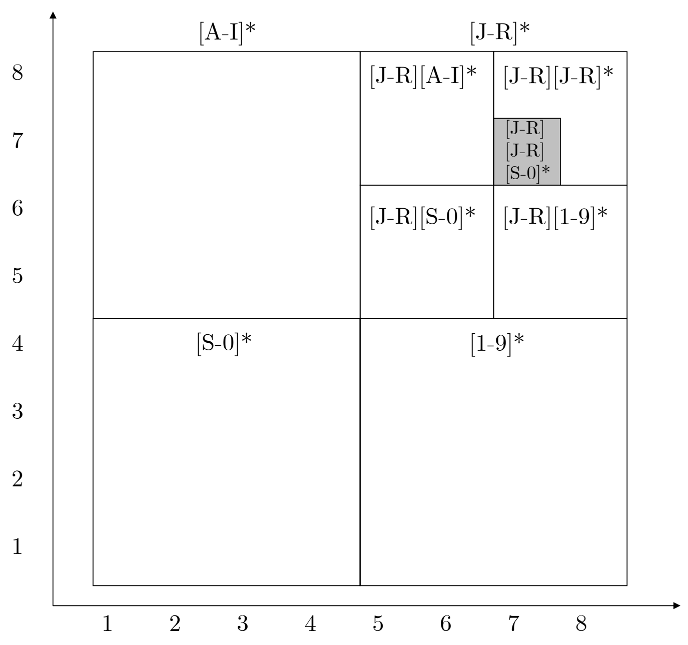


Figure 7.3: Example of quad tree coding

are 1-dimensional, thus they are simply special cases of the example.)

7.5.1 Quad Trees

The standard hashing algorithm of the DHT needs to be replaced by a function that maps equal names to equal places. We use a region quad tree [Sam84] to preserve the lexical order of all keys. The result is that one is able to spot in advance the area where a set of keys with the same prefix is located. One example of such a mapping is shown in Figure 7.3. The hash space is divided into four quadrants, according to the first character of the object name; (in the example, object names can only contain capital letters A–Z and numbers 0–9; the approach easily generalizes to any character set). Each of the quadrants is further divided into four quadrants according to the same mapping, and so on. As a result, any object whose name starts with the prefix "JOS", would be mapped to the shaded area near the top right corner of the area.

Combined with the DTC algorithm, this mapping enables an efficient prefix search in a DHT. We simply pick any node at random from the area of the desired prefix and span a tree over the area covered by the prefix.

We can also adjust the granularity of the mapping in order to control how many peers are present in the areas for prefixes of different lengths. We call this the *split factor*, which works as follows. For example, consider the mapping shown in Figure 7.3. The whole hash space is divided into four quadrants and each quadrant is assigned a set of letters. When objects are stored, we look at the first character of

Length of Prefix	**Nodes in Area**	**Share of Keyspace Split Factor 1**	**Share of Keyspace Split Factor 3**
0	1,000,000	100%	100%
1	250,000	25%	1.6%
2	62500	6.25%	0.02%
3	15625	1.6%	0.0004%
4	3906	0.4%	0.0001%
5	976	0.1%	$< 6 * 10^{-6}\%$
6	244	0.02%	
7	61	0.006%	
8	15	0.001%	
9	4	0.0004%	
10	1	0.0001%	

Table 7.1: Split factor one and one million nodes

the object name and select the quadrant that matches. Then we split that quadrant again into four according to the same rules and map the second character of the object name. As the space is split once per each character, we call this split factor one.

Table 7.1 shows how many nodes need to be searched for a given prefix. (Note that we consider prefixes as characters of object names, because this would be how users would employ a prefix search.) The numbers have been calculated for a CAN of one million nodes and uniform distribution of peers. As we can see, already relatively short prefixes of four to five characters map to a relatively small number of peers, on the order of a few thousands. Already with a prefix of ten characters, we would typically end up with only one peer being responsible.

Increasing the split factor results in a more aggressive splitting. For example, consider a mapping of letters and digits to six bits (similar to Base64 encoding [FB96]). For each split, we pick two bits and map them to one quadrant of the space. One character is three such two-bit groups, and we have to split three times before the area of a single character prefix can be determined. We call this split factor three, an example of which is shown in Table 7.1 .

As we can see in Table 7.1, increasing the split factor results in a much more aggressive splitting. For example, a prefix of two characters maps to an area of 244 peers.

We also define split factors smaller than one, which are less aggressive than those in Table 7.1. This means that each splitting requires a prefix of multiple characters. For example, splitting factor 0.5 (i.e., prefix of two characters for each split) is implemented as follows. We arrange all the characters on the x- and y-axes and then the points in the coordinate space define two-character prefixes. We simply divide this space into four equal areas and use those areas for splitting. This approach easily generalizes, thus we can freely select the granularity of the splitting according to the needs of the application. We do not change the general overlay structure only the hash function is replaced. Thus, already known methods for performance optimization [DLS$^+$] and congestion control [KLBA] will apply as well.

7.5.2 Range Queries

Note that regardless of the splitting algorithm and split factor, the DTC algorithm can span the tree in any convex area of the hash space. When spanning the tree, the DTC algorithm can also specify individually how far along each coordinate axis the tree should span. By selecting the appropriate coordinates, it is possible to span the tree in such a manner that it corresponds to a range query over the objects in the DHT. Further evaluation of the practical performance of range queries is part of our future research.

7.5.3 Discussion

A correct split factor may be critical for application performance. For the case of a prefix search of objects in a DHT, we obviously want to limit the search areas to be as small as possible without unduly stressing peers. For a high split factor, even short prefixes map to a very small number of nodes and may thus result in severe load imbalances because some prefixes are more popular than others. On the other hand, a high split factor means that only a very small number of nodes must be included in the DTC-constructed spanning tree.

Usually, the requirements for distributed hash tables demand a more or less equal distribution of all keys in the hashtable. On the one hand, the equal distribution balances the average load per peer which reduces local hotspots; alternatively, the equality destroys the order of all keys.

One way to reduce hotspots is to choose a smaller split factor for multidimensional quad trees at the cost of larger search areas. In addition, there are several approaches for caching or load balancing queries within the CAN overlay [SMB01, RLS$^+$03]. Techniques such as virtual nodes[I. 01] can also help to alleviate the load imbalance. Evaluating the effectiveness of different load balancing techniques is part of our future work.

7.6 Evaluation

We now evaluate the performance of our DTC algorithm and compare it against two other solutions. We tested DTC over two DHTs, Chord and CAN, to show how its performance in some cases depends on the underlying DHT. For comparison, we have selected the application-level multicast on CAN by Ratnasamy et al. [RHKS01] and a simple flooding scheme, where each node just forwards a message to all of its neighbors, except the neighbor from which it obtained the message.

As already reported by Jones et al. [JTWW02], implementing CAN is not always a straight forward process, but we did not encounter any further problems than those mentioned in [JTWW02]. We verified our implementation of the CAN network by comparing it to the results in [JTWW02] and thoroughly testing our own, and the results were as expected. Our Chord implementation is directly based on the original work in [I. 01]. Our implementation of the ALM algorithm of [RHKS01] did not face the race-condition mentioned in [Cea03a, JTWW02]; because we used the cycle-based approach of PlanetSim in our simulation approach, it eliminated the mentioned race-condition.

We simulated the four selected systems. The two DTC-based approaches built the spanning tree, as described in Section 7.4. ALM from [RHKS01] and the simple flooding were permitted to run as specified. We varied the size of the network (values selected according to Section 7.5 for reasonable search areas) and, for CAN-based systems, also the number of dimensions in the CAN. All simulations were repeated 30 times, and the reported numbers are averages over the 30 simulation runs.
We measured the number of messages received by each node and the depth of the spanning tree. The first metric determines how (in)efficient the mechanism is, and the second determines how quickly search results are available.

Table 7.2 shows the distribution of how many messages a given node received in a 2000 node CAN with 10 dimensions. (DTC-Chord was run on a standard Chord of 2000 nodes.) The table shows for each of the systems how many nodes on average received the message from the root. We cut the table at 14 messages per node and summed up all the nodes that received the message 14 times or more (applies only to simple flooding, in this case). The last row shows the total number of messages sent within the system. As there are 2000 nodes, 2000 messages are sufficient in the optimal case.

As expected, the two DTC-based solutions do not generate any duplicate messages. ALM performs relatively well, generating about 50% too many messages. We return to the evaluation of the overhead of ALM below. As Table 7.2 shows, simple flooding has an extremely high overhead in terms of messages sent. The factor-13 overhead shown is typical of the performance of simple flooding.

The performance of the DTC-based algorithms was as expected in all investigated parameter combinations (e.g., network size, dimensions). Both of them were able to perform their task consistently with the *minimum* number of messages, i.e., as many messages as nodes.

We also evaluated the depth of the spanning tree, as this directly affects the time it takes to complete the operation (e.g., search or broadcast). We compared different network sizes from 200 to 20000 nodes and different dimensions in CAN (ranging from 2 to 20).

Figure 7.4 shows 5- and 10-dimensional CANs with 20000 nodes. In Figure 7.4(b), we also plot the DTC-Chord. The x-axis shows the number of hops and

Messages Received	**DTC CAN**	**Simple Flooding**	**ALM**	**DTC Chord**
0-1	2000	1	1143	2000
2-3	0	2	684	0
4-5	0	6	78	0
6-7	0	22	12	0
8-9	0	67	2	0
10-11	0	247	0	0
12-13	0	925	0	0
≥ 14	0	7851	0	0
Sum	2000	26106	3087	2000

Table 7.2: Average number of messages received per node

the y-axis shows how many nodes were reached with that many hops (i.e., how many nodes are at that depth in the spanning tree).

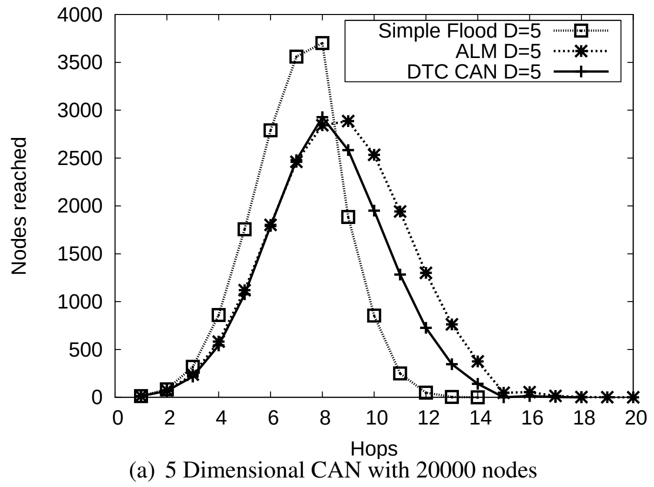
As we can see, simple flooding has the smallest depth. It always takes the shortest path to each node, because it forwards a message to all neighbors except the sender. Therefore all nodes are reached with the minimum number of hops.

DTC-CAN and ALM have slightly poorer performances than simple flooding and are very close to each other. In the case of the five-dimensional network and 20000 nodes, the optimum would be about seven hops, while ALM needs nine and DTC-CAN eight hops in the average. The greater the number of dimensions used for the CAN network, the smaller is the difference between the approaches. Differences in the ten-dimensional CAN network (see figure 7.4(b)) with 20000 nodes are almost non-existent; both ALM and DTC-CAN need about seven to eight hops on an average, while the optimal case would be approximately six hops.

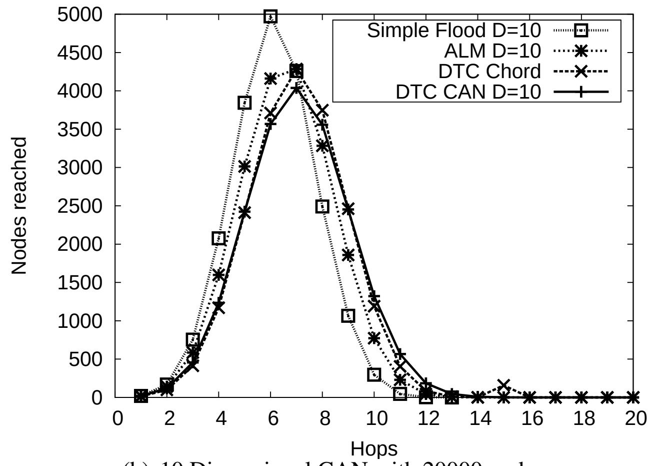
DTC-Chord (shown only in Figure 7.4(b)) performs similarly to DTC-CAN and ALM in the 10-dimensional case. DTC-Chord is independent of the number of dimensions, so it would be in the same place in Figure 7.4(a). The finger tables of Chord reduce the number of necessary hops effectively. While the CAN network is able to be further optimized by using more dimensions, the optimum number of hops for Chord is proportional to the density of the finger tables.

We investigated the performance with dimensions ranging up to 20, but we did not observe any significant improvement in performance of DTC-CAN or ALM after ten dimensions.

We now turn to evaluating the overhead of ALM, which was already shown in Table 7.2. We compare ALM against DTC-CAN. Figures 7.5(a) and 7.5(b) show how the overhead evolves as function of network size. The x-axis shows the number of simulated nodes; we started from 200 nodes and simulated up to 20000 nodes. The y-axis shows the average number of generated messages. We show several variants of ALM, each with a different number of dimensions. Note that DTC-CAN



(a) 5 Dimensional CAN with 20000 nodes



(b) 10 Dimensional CAN with 20000 nodes

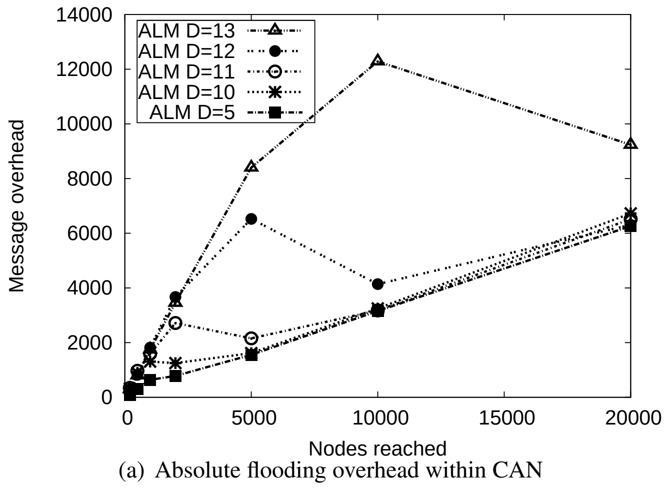
Figure 7.4: Query depth

was always able to perform optimally, i.e., as many messages as there were nodes.

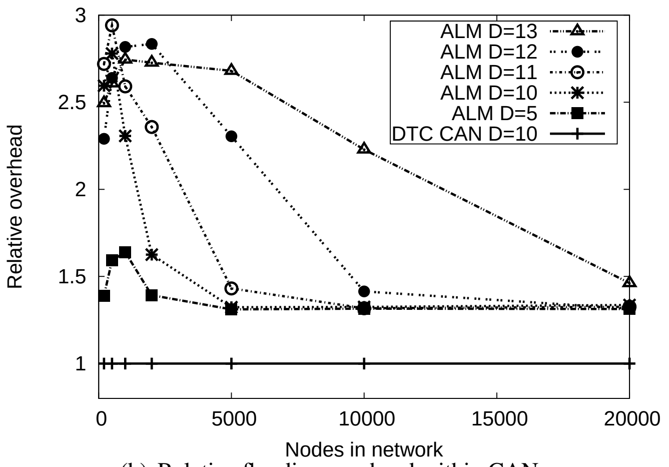
Figure 7.5(a) shows the absolute overhead, i.e., how many unnecessary messages ALM sent, and Figure 7.5(b) shows the relative overhead compared to DTC-CAN.

As the network size grows, the relative overhead of ALM in Figure 7.5(b) tends to be approximately 32%. However, Figure 7.5(a) shows the interesting behavior of ALM. For every number of dimensions, the curve has several sharp corners where the overhead changes considerably. The reason for this is as follows.

The message overhead is heavily influenced by the absolute number of dimensions and nodes. The greater the dimensions, the more nodes are needed to populate the d-dimensional ID space uniformly. As soon as $\sqrt[d]{n} \geq 2$, the overhead begins to converge to the estimated 32% (see Figure 7.5(b)). Below the critical threshold, the number of duplicate messages steadily increases with more nodes. This explains the



(a) Absolute flooding overhead within CAN



(b) Relative flooding overhead within CAN

high peaks for the smaller networks in Figures 7.5(a), and as Figure 7.5(b) shows, the resulting overhead can be up to 250%.

In summary, the DTC-based approaches have the advantage of generating only the minimum amount of traffic, while keeping the depth of the spanning tree similar to ALM. ALM, on the other hand, has a message overhead of at least 32%, in many cases up to 250%. The simple flooding approach turns out to be unusable because the enormous message overhead would cause unacceptable congestions within the overlay. However, simple flooding has the fastest response time of all compared approaches.

7.7 Robustness of DTC

Because the DTC algorithm (and other similar approaches like [RHKS01, Cea03b]) builds a tree, it is especially vulnerable to defective peers, which could either be the result of a crash or a malicious peer. If a peer does not forward the message, then all its children and their children will not be included in the spanning tree.

All DHTs have mechanisms in place to detect crashed peers and recover from those crashes, so we do not consider such system failures to be a problem. The only case a message could be lost is if a peer crashes between receiving a message and forwarding it, which is extremely rare. In all other cases, we assume that the standard DHT maintenance takes precedence over the tree spanning messages, and that in such cases, the overlay will first be healed and thus no message loss or duplication can occur.

Malicious Peer In the following, we consider the case of a malicious peer not forwarding the message correctly onwards. The case of a malicious peer modifying the message can easily be detected by picking a random public/private keypair, signing the original message, and including the public key in the query. We first evaluated the severity of this problem by selecting a randomly generated 20000-peer network and spanning a tree with the DTC algorithm over the entire network. The CAN networks in this test had ten dimensions. We varied the fraction of malicious peers and selected the malicious peers uniformly at random from all the peers. For each case, we measured the number of peers that were not part of the tree. Each parameter combination was repeated 30 times, and the results we present are averaged over all the runs.

Figure 7.5 shows the fraction of unreached peers as a function of malicious peers for three systems from Section 7.6 (DTC-CAN, DTC-Chord, and ALM). Already 10% of malicious peers are able to cut off, on average, 20% of the peers from the spanning tree. Because the DTC algorithm eliminates all duplicate messages, it is particularly vulnerable to malicious peers, but the ALM does not fare much better. Only when the fraction of malicious peers is between 10% and 50% does ALM have any marked improvement over DTC-based systems. Although not shown in Figure 7.5, the simple flooding approach is extremely resistant against malicious peers. Peers become unreached only when the fraction of malicious peers is very high (typically over 70–80%).

Effect of underlying DHTs There is an interesting point to note about the effects of the underlying DHT on the performance of DTC-algorithms. For the case of 1% malicious peers, Chord-based DTC performed much more poorly than CAN-based DTC. In case of DTC-Chord, 8.6% of the peers were not reached, whereas in a DTC-CAN, only 4.2% of the nodes were unreached. For comparison, in ALM, 3.7% of the nodes were unreached for that fraction of malicious peers. For other fractions of malicious peers, the difference between Chord- and CAN-based DTCs

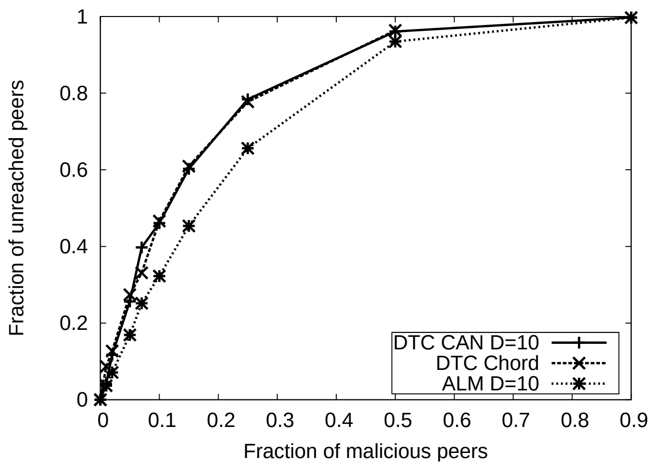


Figure 7.5: Impact of malicious peers

was in practice negligible, although usually the Chord-based system had the higher number of unreached peers.
The explanation for this is that in a 10-dimensional CAN, even if a malicious node is near the root in the spanning tree, it cannot do much damage. In contrast, if the longest finger of the root in Chord is malicious, it is able to cut half of the spanning tree. Thus, a Chord-based DTC is slightly more vulnerable to malicious peers than a CAN-based DTC.

Detect Malicious Peers - Propagation We now propose solutions for remedying the problems caused by malicious peers. The fundamental problem is that a single malicious peer can eliminate the complete sub-tree rooted at that peer from the spanning tree. Our primary goal is thus *detecting* the presence of malicious peers, with as little overhead as possible. The mechanisms below are not always able to pinpoint the malicious peer; they only aim to discover malicious behavior in the tree.

The effectiveness of DTC greatly depends on the kind of application using DTC. If we are searching for objects stored in the DHT (as in Section 7.5), then the problem simplifies greatly. This is because all peers should return a response to the root of the tree indicating whether they have any objects that match the search. By requiring *all* peers to send a response, the root can easily check the responses to see whether the complete area has been searched. Any malicious peer would easily be identified with such responses.
In the interest of scalability, the responses should be propagated back along the tree. Furthermore, each peer should sign its response with a key that is tied to its zone of responsibility in the DHT and its IP address. These signatures increase the likelihood of discovering malicious peers, because the malicious peer would have to invent the zones of responsibility and IP addresses for all the peers in its

subtree. These might overlap with legitimate zones from other parts of the tree, thus indicating the presence of a malicious peer. Furthermore, the root could also verify some of the leafs of the tree (using additional communications) to confirm that the responses were actually sent by them. Although this would reduce the effects of malicious peers, in the absence of a centralized identity management scheme, it cannot completely eliminate them.

Detect Malicious Peers - Multiple Trees Another solution is to split the root of the spanning tree and, instead of a single tree, generate several spanning trees, each covering a small area of the total tree and in such a manner that the union of the smaller trees covers the whole area. This is easy to perform, because the DTC algorithm allows us to define the area freely. The advantage is that each individual search area becomes relatively small and allows for an easier detection of malicious peers. As discussed in Section 7.5, a prefix length of four digits results in a search area of about 4000 peers. By splitting the query into ten queries, each query needs to cover about 400 peers. Requiring a response in such small trees does not present a significant overhead, and thus, malicious peers would be easier to detect. Note that the response traffic is required by the application and is thus independent of the way the spanning tree is created (DTC, ALM, flooding, etc.).

The above solutions work well in the case where the tree is spanned for a purpose that requires a response, e.g., a search. In the case of a pure one-way tree, such as broadcast, no natural feedback channel exists. Acknowledgements, similar to the responses in the search application, would work as described above, but in this case, they would generate additional traffic in the network. However, without such an acknowledgement mechanism, it is not possible to detect malicious peers. Thus, we propose using the same mechanism for any kind of application. The second solution, the creation of several, non-overlapping spanning trees, is likely to be more efficient at detecting malicious peers, but a full evaluation of the proposed mechanisms is part of our future work.

7.8 Summary

In this chapter, the multicast mechanism for the locality preserving overlay network CAN was proposed. As requirement R1 explicitly stipulates sending a message to a distinct set of peers, we designed an efficient multicast mechanism for volatile environments found in first response scenarios. However, this approach provides strong mechanisms for R3 (locality awareness) and R5 (active search) as well. CAN is able to store objects with similar properties (e.g., objects with a common prefix or location information) close to each other (see Section 7.5.1), which enables efficient searching for objects with common properties.

Table 7.3 shows the state-of-the-art multicast approaches in comparison to our proposed ALM mechanism. While the state-of-the-art options propose lightweight, application-level multicast mechanisms as well, to the best of our knowledge we are the first to propose an ALM approach for CAN without duplicate messages.

Mechanism	**Average Delay**	**Number of Messages**	**Message Ack.**	**CPU Effort**
ALM-Castro	cf. CAN	Duplicates	•	Very Low
ALM-Ratnasamay	cf. CAN	Duplicates	•	Very Low
ALM	cf. CAN	Minimal	•	Low

Table 7.3: Comparison of ALM with state-of-the-art approaches

Our approach creates a spanning tree over a part of a DHT, with no inter-node communication and using only information available locally on each node. A mapping based on region quad trees enables CAN to perform prefix and region searches on content stored in the DHT. This solution is not limited to searches, but can also handle broadcast and multicast communications. Our evaluation shows that DTC performs as expected, and the comparison to similar approaches from literature reveals that the message overhead of existing solutions is considerably higher than the optimal number of messages sent by DTC. The depths, of the spanning trees are similar in all studied cases. We have also discussed the robustness of DTC against malicious peers and presented solutions for strengthening DTC.
In the next chapter, an application model is presented; its purpose is to maintain a strict command and control hierarchy in a distributed P2P network. It relies on the functionality provided by the overlay network layer, as well as the connection topology layer.

Chapter 8

Application Layer: Distributed Command and Control Structure Management

Hierarchically structured organizations have clear and relatively short communication paths. However, one does not take advantage of this fact in modern P2P overlays. As discussed in Chapter 2 management of a command and control hierarchy is one of the key requirements for efficient and successful disaster relief. We show how a command and control order can be imposed on a structured P2P network. This approach can be used to benefit from hierarchical structure in any communication system, especially in first response scenarios.

8.1 Introduction

8.1.1 General Approach

In this work, we present Distributed Command and Control (DCC) structure management, a mechanism that imposes an hierarchical organization upon every structured P2P network. DCC is independent from the structure of the underlying network. Within first response systems, such hierarchy can be used to regulate the command-flow in a disaster management. For example, a firefighter has just connected to the P2P network and wants to communicate with the mobile headquarter. Our mechanism allows him to find and contact the headquarter without any further information.

As the communication network continuously changes, users join and leave all the time, and the imposed hierarchy has to be adapted as well. If a certain first response team disconnects from the network, the gap in the communication chain must be refilled by the most appropriate participant in the network. We use a distributed hash table (DHT) in order to manage the hierarchy in the network. The position of a peer in the hierarchy is the key, and its address is the value; in other

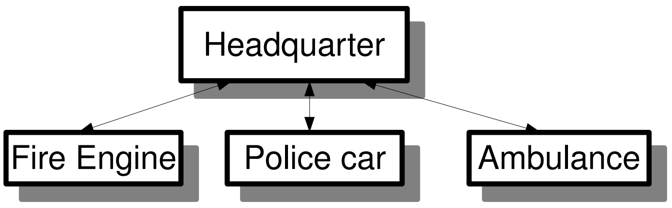


Figure 8.1: First response hierarchy

words, using a hash function, one can compute for each position in the hierarchy a corresponding node ID. That is how each peer can easily determine its own position, or find a peer that has a certain position within the network hierarchy.

8.1.2 Requirements

The main task of the 'Application Layer' itself is to provide a clean interface to the (cf. section 3) overlay network layer and to act as an environment for the full set of required applications. They are deployed into this layer satisfying one or several elicited use cases (cp. Section 2.2).
The application layer is designed to act as a general host for a set of applications. In the course of this thesis, we developed one approach, which fully covers the elicited use case R6 (hierarchy maintenance, Section 2.2). This use case requires multiple connections to different sets of nodes. In addition, substantial network traffic is required in case of shift changes (use case R2) and node failures.

R2 (Shift Change) Teams of first responders in the incident and post-incident phases work in relatively short shifts. The communication support needs to immediately integrate the new team on duty into the existing organization hierarchy and transfer left open tasks to the new team.

R10 (Hierarchy Maintenance) During the incident phase, frequent adaptations to the organizational hierarchy need to be performed. In the incident phase, more than one required adaptation per five minutes was reported in the Arlington-County action-log (see Section 2.2).

8.2 Related Work

Our DCC structure management approach, is based on distributed hash tables (see Section 3.2 for an introduction to distributed hash tables). To the best of our knowledge, it is the first approach to managing command and control structures in a DHT overlay network. Nevertheless, there are other approaches that use P2P overlay networks to accomplish a hierarchical structure for different purposes.

One such data overlay is Somo [ZSZ03], which can be used on top of any P2P-DHT network too. It collects and distributes meta data in $O(logN)$ (where N is the network size) and scales together with the hash table being used. The meta data is aggregated from the leaves to the root of a tree and afterwards disseminated along the tree. In fact, the Somo protocol may run on top of our organization hierarchy, the repair costs for crashes will even be reduced by the first-come, first-serve policy of DCC (a peer takes the highest vacant position possible).

When working with large complex applications, using simple distributed hash tables is one of the best approaches. One such application is Place Lab [CRR+05]. It stores and provides WiFi hotspots and mobile phone masts in a database. The database is distributed on several servers (peers) that are responsible for storing and computing data. To manage this task, Place Lab uses distributed hash tables. Every server (peer) is responsible for a certain range of data. If the data in that range become too large, the range is split and another peer receives the other half. That way, a binary tree emerges whose leaves contain the data. The child node has the label of the parent concatenated with 0 or 1, given to the left child or right child, respectively. This data structure is called the prefix hash tree (PHT). PHTs need only a simple put/get/remove-interface and can be built upon distributed hash tables. PHT is an astonishing simple protocol, but has no overwhelming performance.

Cone [BVV03] is a distributed data structure especially designed for mathematical operations like the sum and min/max. Cone uses a DHT overlay network, and unlike DCC, focuses on data aggregation. In comparison, DCC is designed for to create and maintain a dynamic organization structure; it can be easily extended to exploit this structure for data aggregation and dissemination tasks.

8.3 Basic DCC-Model

The DCC structure management approach enables one to impose a hierarchy upon any underlying structured P2P-network. For this purpose, it uses only send and receive methods. Every overlay node represents one person in the disaster management, with her/his specific rank in the chain of command. The overlay node knows its rank and fills the corresponding position within the hierarchy. The rank and the position of a node in the hierarchy are represented by an unique label. The label of the peer with highest position, i.e., the root, contains one or more alphanumeric symbols. The label of each subordinate peer is composed from the label of the preceding peer, a dot and one or more alphanumeric symbols. For simplicity, we just used one decimal digit for each position. Figure 8.2 represents an exemplary scheme. The hierarchy does not necessarily need to be a completely balanced tree or to have a symmetric structure. Depending on the desired staff-line, organization leaves may be placed in different depths. The prefix values in the label of each node describe its path to the root. This path can be used for different purposes: to contact superior nodes, for targeted communication with a specific node through a common superior node or to monitor the direct superior node and apply for its position in a case of failure.

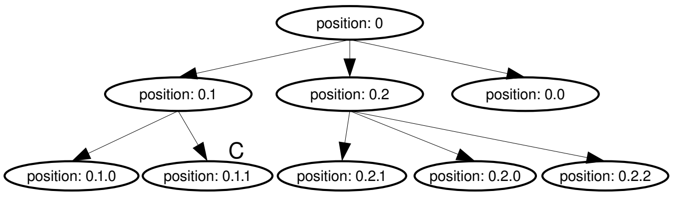


Figure 8.2: Example of staff-line hierarchy on top of a DHT-network

8.3.1 Joining

We use a DHT to build the staff-line organization. The position in the hierarchy is the key, and the value is the address of the corresponding peer. Assume that a new node *A* tries to join the network, as specified in the P2P network overlay. After *A* is connected to the network, it has to allocate its position within the hierarchy. To achieve that, *A* hashes its rank with a hash function known to all peers, to generate a network ID. Let *B* be the overlay node, responsible for the ID generated by *A*. That is, *B* administrates the position *P* within the hierarchy, which corresponds to *A*'s rank. *A* sends a request to take position *P* to *B*. The request contains: the rank of *A*, its address and the desired position. Node *A* receives a positive reply if *P* is vacant, or if the node currently occupying *P* is less appropriate than *A*. Otherwise, *B* responds with information about the node *X* occupying *P*: its rank, its address and its position (*P*). Then *A* sends a request to *X* and so on, until *A* finds its position within the hierarchy. Once *A* has found its position, it also knows its direct superior peer.

For communication within the network, it is sufficient if every peer knows its superior and its subordinate peers. However, if necessary, any peer can locate any other peer within the hierarchy. Consider Figure 8.2 again and assume that a peer *C* (position 0.1.1) wants to communicate with a remote peer (e.g., position 0.0). *C* knows that its position is one level below the position it is searching for: the position of *C* has three dots, i.e. is on level three, and the desired position only two, i.e., is on level two. Therefore, *C* sends a message to its superior, telling it that *C* seeks a peer on a given position. Then, the superior node of *C* decides on its turn, depending on its position and the requested position, whether to redirect the request to its superior or to one of its subordinate nodes. Once the requested position has been reached, a positive message is sent back along the chain. In other words, the hierarchy of the peers is used to target messages precisely. Furthermore, each peer has to make only locally based decisions.

In order to keep the hierarchy consistent, each peer occupies the most superior vacant position, even if this position does not directly corresponds to the peer's rank. Once the correct or more appropriate peer appears, the currently occupying peer goes down the hierarchy tree. In the following, we call a hierarchy without gaps a *correct hierarchy*.

8.3.2 Hierarchy Maintenance

P2P networks are like living systems; peers join, leave or even crash all the time. These three processes are generalized under the term churn. The network overlay and the hierarchy have to be robust against churn. The self-reorganizing capabilities of P2P networks provide a stable basis for DCC. The exact manner depends on the concrete overlay, but they all can effectively detect and handle new, crashed and leaving peers. Once the underlying network has been repaired, the hierarchical structure also has to be adapted. A crashed or leaving node means a gap in the hierarchy. This gap has to be filled by a subordinate peer. However, this in turn produces a new gap, until the repairing peer happens to be on the bottom most level of the hierarchy tree. The same situation emerges when a peer joins the network. It has to be placed in the most appropriate position. This, however, could mean that some peers must be demoted, until again a bottom peer has been reached. Nevertheless, in both cases, it is only one single path within the hierarchical tree that is affected. In the worst case, the number of overall affected nodes scales as $O(log_d(N))$, where N is the number of peers in the network and d is the number of subordinate peers per node. More importantly, the hierarchy always remains consistent. It is simply about redistributing hierarchical positions.

In order to detect and react to churn, our mechanism periodically checks whether the peer managing a given position is still the same. If it has changed, the proposed mechanism just routes a message to the new peer. Routing messages are also periodically sent to the superior peer to determine whether it has changed or whether the position is not suddenly vacant.

8.3.3 Messages

DCC sticks to the key-based routing interface (KBR) [DZD$^+$03] of the underlying P2P network. DCC uses a set of four different messages to communicate with the underlying network and other applications running on top of it. In this section, we describe the structure and the purpose of those messages:

- position request message
- accept position request message
- position information message
- quit position message

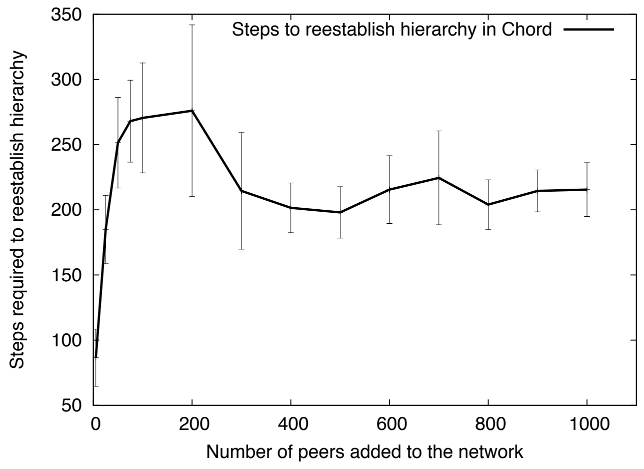


Figure 8.3: Reestablished hierarchy after node join in Chord

In case a node wants to take a position within the hierarchy, for example, to join the hierarchy or to occupy a more appropriate position, a position request message is routed. The message contains: the sender's rank, the sender's desired position and the sender's address. The message is sent to the peer responsible for the network ID corresponding to the desired position. If the requesting node can fill its desired position, it receives an accept position request message. The message contains the sender of the message and the position. If another node holds the desired position and is more appropriate, the requesting node receives a position information message. A position information message contains: the sender, the position, the address and the rank of the node holding the position. The requesting node must select another position in the hierarchy, generally a subordinate position. A node may also have to quit its position, if another more appropriate node request its position. The receiver acknowledges the request and sends back a quit position message. This message contains the sender and the position. Afterwards, the requesting node starts the join procedure described above.

Of course, the message types can be extended according to the needs of the concrete application domain, for example, for information exchange among nodes with the same superior node. Furthermore, information aggregation and dissemination protocols along the tree can be applied.

8.4 Simulation Setup

The following section describes the experimental setup we used for our simulations. We measured the time required (in simulation steps) to establish correct hierarchy after continuous peer join or churn within the underlying network. In the simulations, we placed DCC on top of the P2P networks Chord [I. 01], Symphony [MBR03]

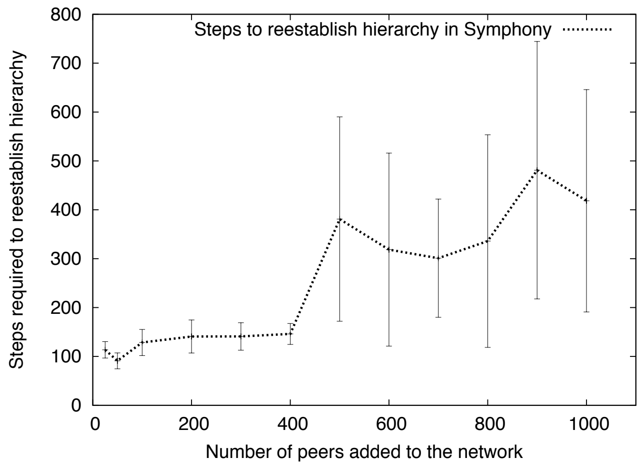


Figure 8.4: Reestablished hierarchy after node join in Symphony

and CAN [RFH+01b]. All charts are averaged values derived from repeating each experiment twenty times. Please note that CAN performs exactly like Chord if we set the dimensions $d = log_2 N/2$ [LKRG03].
To simulate first response scenarios, we used both, the P2P simulator PlanetSim [Gea05] combined with the First Response Communication Sandbox [BPSM08] (FRCS), thus generating individual physical movement data for each peer. This is the proposed simulation approach of the 'Movement and Network Model' layer (cf. Section 4).

The application layer may use, the DHT functionality provided by the 'Overlay Network' layer (cf. Chapter 4). Every overlay network provides the same interface to the upper application layer (see Section 3.3). Therefore, one can run simulations with different overlay networks using the same application. That is, changes in the environment of one of the simulators are reflected in other ones, and vice versa. Thus, FRCS simultaneously provides fine grained movement and location-aware network models. Therefore, it produces more realistic results than combining two separate simulators.

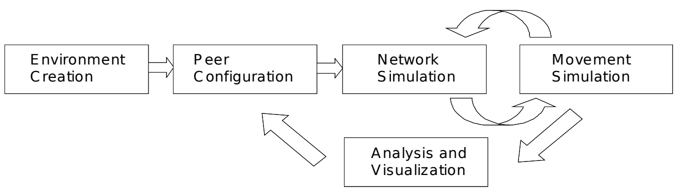


Figure 8.5: Workflow of the simulation process

Figure 8.5 shows the complete simulation process. It begins with creating the environment, which is fixed for the rest of the process. Afterwards, the peer configuration takes place, i.e., the number and the resources of the mobile peers, as well as the stationary peers are configured. The next two steps are performed in parallel; PlanetSim is running the network simulation while FRCS simulates the movement of all peers. Concrete simulation results are presented in the section.

8.5 Evaluation

In our first experiment peers, join the network randomly. We always conduct two simulation steps after a node has joined the network (one simulation step is about 100ms on average). We measure the time, i.e., the number of simulation steps, required to establish a correct hierarchy. The hierarchy tree is constrained to at most three subordinate positions per peer. Since peers join the network randomly, at least at the beginning, they are usually placed first on higher positions than their ranks. That avoids gaps in the hierarchy but produces higher reorganization costs in the beginning of the simulation process. When the peers with the correct ranks appear, reorganization takes place.

The empirical results show that the time needed to establish correct hierarchy varies only slightly with the network size (see Figure 8.3). It takes about 250 simulation steps from the time of the last joined peer until the time when the hierarchy is fully established. One simulation step is the time needed for one communication hop, which we set to 100ms. For example, in the observed scenario, the organizational structure is established and functioning after approximately 25s.

The same experiment performed on Symphony has different reorganization costs (see Figure 8.4). The time needed to reestablish the hierarchy depends on the network size, because additional routing steps are required. Small networks perform any routing within one densely connected cluster. With increasing network size, several clusters appear, which are only sparsely connected with long distance links. In that case, routing means leaving a cluster with a long distance link, entering another cluster and then locally routing to the desired target. Nonetheless, average reorganization costs are below 60s for a network with 1000 peers.

Comparing both P2P networks, the same implementation of DCC shows much better performance in the Chord network. The reorganization delay is about 60% less than that of Symphony. Furthermore, the variance of Chord is smaller than the variance of Symphony. This comes from the better structure of the underlying network of Chord. Nevertheless, the price is more links per peer and very constrained freedom when new links are established.

Our next experiment simulates a typical scenario: after the joining process is over and both the underlying network and the hierarchy are in steady state, we simulate a random crash of 5% of all peers. We set the crash rate to 5 crashes per second (i.e., one peer crashes every second simulation step). During the crash phase, the underlying P2P network is already allowed to start maintenance operations.

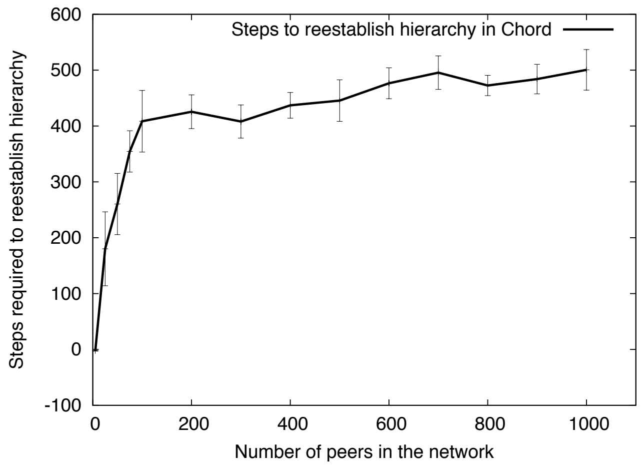


Figure 8.6: Reestablish hierarchy in Chord after failure of 5% of all nodes

The results show that the time required for Chord to reestablish the correct hierarchy after the last peer has crashed grows logarithmically with the network size (see Figure 8.6).

The same experiment is carried out on the Symphony network as well. Figure 8.7 shows the repair costs for a random crash of 5% of all Symphony peers. Given a network size of 1000 peers, on average, 800 steps, which require approximately 80s, are needed in order to fully reestablish the organizational structure. As observed in the former experiment, where we continuously added peers to the network, Symphony has higher variance in the delay for repairing the network. This is again caused by the underlying routing mechanism of Symphony.

Then, we investigate the impact of different crash rates with a fixed number of peers (500) in the network. The experiment requires an already established organizational structure and begins after both the underlying network and the hierarchy are in a steady state. We simulate crash rates between 1% and 20%.

It turns out that the time Chord required to reestablish correct hierarchy hardly increases (see Figure 8.8). Even in a situation of a crash of 20% of all peers, the network will most likely be re-established in between 40s and 60s. In other words, independent of the underlying network, the overhead for building and maintaining the network hierarchy only increases marginally with the network size and does not depend on the crash rate.

Using the same simulation parameters, Symphony shows a slightly longer delay than Chord. Though fail rates up to 20% are successfully repaired by DCC, recovering requires almost always require more time than Chord. Node failure rates higher than 15% show decreasing repair costs. This effect is even more visible using a higher fraction of node failures. Even though this effect may be counterintuitive, it occurs because the higher the number of failed nodes, the smaller the network size becomes. The organizational hierarchy substantially shrinks and therefore fewer

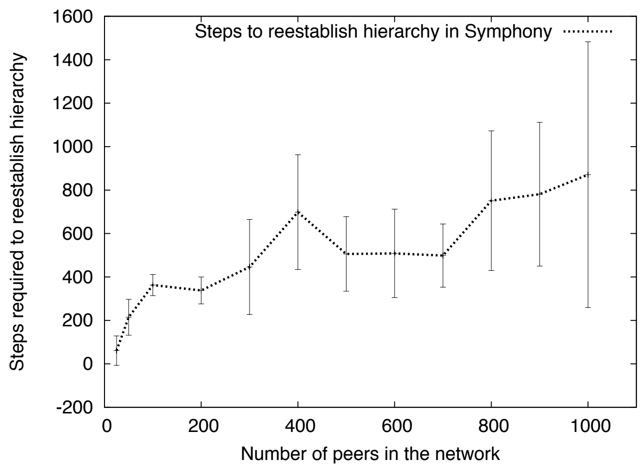


Figure 8.7: Reestablish hierarchy in Symphony after failure of 5% of all nodes

messages are needed for it to be reestablished.

All results described above are acquired with hierarchy constrained to at most three subordinate positions per peer. But how does this affect the overall performance? We perform another experiment. We fix the number of peers in the network, vary the number of subordinate positions, remove the most superior peer (which produces the highest reorganizational costs possible) and measure the time needed for the hierarchy to recover.

From graph theory, we know: for a fixed number, N, of peers in a given network, the hierarchy height h decreases when the number of subordinate positions grows, and vice versa. For s subordinate positions, it holds that $h = log_s N$. Therefore, when the most superior peer fails, the larger h is, and the faster the hierarchy recovers to a correct state. The time required for the hierarchy to recover depends on its height. Thus, the recovery time scales as $O(logN)$.

Figure 8.9 shows results for both Chord and Symphony networks with a fixed number of 500 peers and under perfect conditions, i.e., no churn and no congestion. The Chord network shows faster recovery time, which is caused by the underlying faster routing algorithm. In addition, our experiments have shown high variance in the experiments carried out on Symphony. Due to the way the routing algorithm of Symphony is built, the average delay may vary more than those of Chord. In the worst case, i.e., two subordinate nodes per peer, Symphony recovers from failures of the root node within 70 seconds, and Chord recovers in 40 seconds on average. Increasing the number of subordinates from two to four greatly reduces the delay for reestablishing the hierarchy: by 40% within the Chord network and by 30% within the Symphony network.

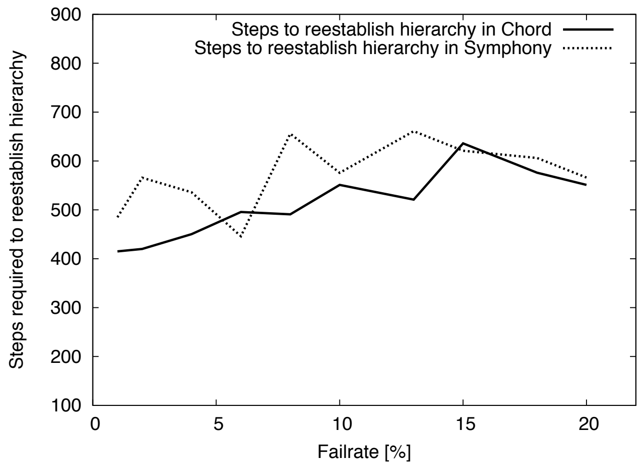


Figure 8.8: Reestablish hierarchy under failure rate up to 20%

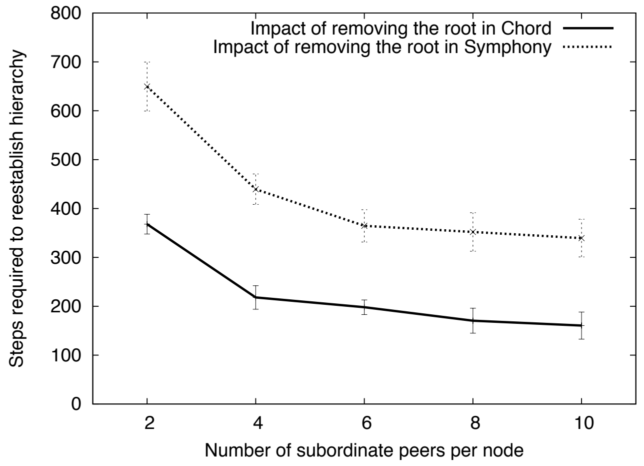


Figure 8.9: Impact of the number of subordinate positions for reestablishing hierarchy after root-node failure

8.6 Summary

The main contribution of this chapter is a novel approach for Distributed Command and Control Structure Management, an application that builds a hierarchical structure upon a given structured P2P network. It is tested with a wide set of P2P networks.

It prevents unnecessary coordination cycles between participating teams. Since each peer knows its position, its superior peer and its subordinates, the command and control path is predefined. This allows for more precise message routing, as each peer can decide whether it should send a message down or up in the hierarchy. Nonetheless, the superior nodes can monitor communication traffic among their subordinate teams.

The overhead for joining, crashing or leaving the network scales logarithmically with network size, N. More precisely, it depends linearly on the height of the hierarchy h, given by $h = log_s(N)$, where s is the number of allowed subordinates per peer. That is why s should be chosen according to the particular scenario. The higher s grows, the more administrative overhead per peer, but the less time the network requires to return to a steady state.

Property	**Somo**	**Cone**	**PHT**	**DCC**
DHT Independent	●	●	●	●
Repair Costs	High	High	cf. overlay	cf. application
Access Delay	Low	Low	cf. overlay	Low
Scalability	cf. overlay	Low	cf. application	cf. application

Table 8.1: Comparison of DCC with hierarchical access approaches for distributed hash tables

One should not forget the overhead produced by the underlying P2P overlay during fluctuations in the network. The size of this overhead depends on the particular overlay. Table 8.1 shows DCC in comparison to its directly related approaches. Unlike PHT, our application is based entirely on key-based routing. That is, each position in the hierarchy is always bound to a peer. To overcome this drawback and make our application as flexible as PHTs, one can introduce *Position Nodes*. They are just placeholders for different positions in the hierarchy, but in fact are administrated by the same peer. Furthermore, when a conflict for a specific position occurs, peers can simply exchange Position Nodes and resolve the conflict with zero overhead.
Our application is placed above a given P2P overlay and communication goes through a specified interface. That is why the application layer has to perform additional tasks like determining timeouts of peers. If the application were integrated within the P2P overlay itself, a central timeout would have been sufficient. However, exactly because of this communication interface, our application is easily extendible and can be used with a broad range of existing P2P overlays.

In addition, the staff-line organization is established explicitly with local knowledge. Every peer has a unique hash value, generated from its label, which determines its responsible node. In case of a network breakdown, this local knowledge can be used whenever the network is available to reconstruct precisely the same organizational hierarchy.

Chapter 9

Conclusion and Outlook

The main contribution of this dissertation is on the one side the delineated four layer architecture for a novel and distributed first response communication approach and on the other side the proposed mechanisms for the most prevalent challenges of each layer. The four layers, called 'Network and Movement Model', 'Connection Topology', 'Overlay Network' and 'Application Layer' have been identified and adapted to the harsh environment found in first response situations. We made significant contributions at each layer and were able to either reduce message load per peer or to increase robustness, and while satisfying the elicited requirements on each layer of the P2P communication approach.

In this Chapter, we first summarize the main contributions of this thesis. In Section 9.2, we compare the contributions of each layer with the state-of-the-art options. In Section 9.3, we discuss potential for future work.

9.1 Contributions

The main contributions of this thesis are:

Overall Concept The concept of the proposed architecture is to adapt each layer individually to its challenges found in first response situations. Our approach is designed to support the division of the communication approach into areas (R11), and explicitly supports devices with limited range, typically IAN and JAN networks (R12 and R13). As it is fully built based on 802.11 and Internet standards, inter-organizational communication could be built on top of the proposed architecture (R8). The proposed architecture may replace the classic voice-only communication, known from the two-way-radio set with a flexible framework for all kinds of applications (R7) required for communication support. On the one side, each layer faces individual challenges; on the other side, failures in one layer may have consequences in the next layer. For example, network partitioning will be detected in the connection topology layer, but has dramatic influence on the overlay network layer and finally, on the application layer. Therefore, all layers have been adapted carefully to the application scenario in order to minimize the risk of network outages and to prevent cumulative but opposing countermeasures.

Movement and Network Model We developed a simulation environment for the 'Movement and Network Model' layer, which on the one hand estimates movement of first responders and civilians (R3) and simultaneously simulates the corresponding mobile communication infrastructure (R4), see chapter 4 for the detailed approach.

The proposed simulation approach does not restrict the adjacent network topology in any way (R10). It is a novel combination of movement simulation and P2P networking simulation, with the possibility of reciprocal influence of both simulation parts.

Connection Topology The novel approach BridgeFinder provides a mechanism for detecting critical paths between sparsely connected clusters of mobile peers, i.e., IAN (R12) nodes and JAN (R13) nodes (see Chapter 5). This approach is used in the Connection Topology layer to detect imminent partitioning of the distributed communication network.
A network consisting of several isolated parts is not usable in a reasonable way; therefore, this mechanism can be seen as a key enabler for the elicited use cases.

Overlay Network: Multicast First Responders need to send messages to a distinct group of people or to a specific location (R1). We propose an efficient ALM in the Overlay Network layer (see Chapter 7). Messages with optional acknowledgment can be sent to either all nodes or to a defined subset of peers within the network.

The multicast mechanism provides a way to find groups of objects with similar properties (R5), e.g., proximate locations (R3), similar duties or adjacent IDs. The approach guarantees that no duplicate messages occur, which saves bandwidth and energy. The state-of-the-art approach for the DHT network CAN provides similar performance, but duplicate messages cannot be avoided.

Overlay Network: Search and Lookup The P2P network called PathFinder is the first overlay combining efficient full-text search and key lookup (see Chapter 6.5) in one overlay (R5). It uses the virtual nodes concept, which allows for dynamic load balancing (R4). Each node may choose one out of many available disjoint paths to an arbitrarily chosen target (R10). Our results show that PathFinder performs comparably or better than existing overlays. Because PathFinder is based on a random graph, we fully comply to state-of-the-art exhaustive search mechanisms as well. PathFinder provides both, key lookup and exhaustive search.

Application Layer Teams of first responders in the incident and post-incident phases work in relatively short shifts. The new team on duty needs to be immediately integrated into the existing organization hierarchy (R2). DCC, a novel approach for managing distributed command and control structures (R10), is able to provide hierarchical data access in structured P2P overlay networks. It prevents unnecessary coordination cycles between participating teams. It allows precise message routing, as each peer can decide whether it should send a message down or up in the hierarchy.

9.2 Comparison Summary

In Table 9.1, we list the main contributions of this thesis structured into the proposed layered architecture. At each layer, a technical comparison with directly related work was performed. For a detailed analysis, please refer to the corresponding chapters of this thesis.

9.3 Outlook

In this section, we show in which ways our approach could be enhanced and suggest some future research directions.

Command and Control Structure Management of crisis response situations have not yet adapted to new communication possibilities. Already established management schemes are based on a strict command and control approach. This is applied almost worldwide, but with a new communication system, nearly every team is able to access an overview of the entire crisis at any time (cf. Chapter 8). Future management approaches should allow communication between peers. Decisions will be made more rapidly, because they are not slowed down by a bottleneck somewhere in the organizational hierarchy. The flexibility provided by the next generation of crisis response communication systems must be considered already today when designing crisis response processes.

Interoperability We propose a communication approach for first responders using common hardware and an open layered architecture, which is the cornerstone for interoperability. However, assume that disaster relief organizations share the same hardware, and that inter-organization communication were easily possible. Then, it remains a significant challenge to adapt organizations to this flexibility. Communication interfaces between a huge set of specialized organizations will be required.

Mixed Mode P2P The research community has seen several examples of DHTs [KK03, DZD+03, MNR02, RD01b, I. 01, ZKJ01, RFH+01a, Pla99, MBR03]. While all of them have individual runtime characteristics and resource requirements, they tend

Layer	Approach	Properties			
		DHT Independent	**Repair Costs**	**Access Delay**	**Scalability**
Application Layer					
	SOMO	●	High	Low	Overlay dependant
	Cone	●	High	Low	Low
	DCC	●	Application dependant	Low	Application dependant
Overlay Network *Structure*		**Probabilistic Search**	**Key Lookup**	**Access Delay**	**Churn Resistance**
	BubbleStorm	●	○	Probabilistic	High
	Chord	○	●	High	Low
	Pastry	○	●	Low	Low
	Viceroy	○	●	Lowest	Low
	PathFinder	●	●	Low	context sensitive
		Average Delay	**# Messages**	**Acknowledgement**	**CPU Effort**
Overlay Network *Multicast*	ALM-Castro	CAN dependant	Duplicates	●	Very Low
	ALM-Ratnasamay	CAN dependant	Duplicates	●	Very Low
	ALM	CAN dependant	Minimal	●	Low
		Detect Global Criticality	**Construction Steps**	**Local Error Rate**	**Global Error Rate**
Connection Topology	CAM	○	1	Low	high
	DMCC	●	2	No error	tunable
	BridgeFinder	●	1	n.a.	low
Movement and Network Model		**Support of Scenarios**	**Network Interaction**	**Behavior Support**	**Exchangeable Movement Models**
	GlomoSim	●	○	○	●
	ANSim	●	○	○	●
	FRCS	●	●	●	●

Table 9.1: Contributions and technical comparison with related work

to follow a simple greedy routing scheme. The common idea is that with every hop in the overlay network, a key lookup message needs to be numerically closer to its target ID. It is natural to ask whether future DHTs, instead of following a certain approach, should have more freedom to choose their neighbors.

PathFinder An open question is the impact of the known (locally calculated) overlay network topology of PathFinder. An attacker can compute the complete topology and object assignments, so if there are any weaknesses in the topology, these can be discovered. For example, if the topology were to have two large clusters with only a few links between them, an attacker could try to partition the network. We do not expect this to be a problem in large networks with sufficiently many neighbors per virtual node. Evaluating the exact impact of this remains an open issue.

Unstructured Command Hierarchy Further development of DCC should try to abandon the required distributed hash table. This may be realized when every new peer communicates with at least one old peer. With the help of the old peer, the new peer should be able to determine peers in the network that have similar ranks. That is, each peer has to find its hierarchical incident node in the existing hierarchy. However it is not clear how much message overhead is required, and the robustness against churn may be subject of future research activites.

Bibliography

[AJB99] Albert, Jeong, and Barabasi. Diameter of the World Wide Web. *Nature*, 401:130–131, 1999.

[A.S03a] A.S. Bahora et al. Integrated peer-to-peer applications for advanced emergency response systems. part i. concept of operations. In *SIEDS, 2003 IEEE*, pages 255–260,261–268, 24-25 April 2003.

[A.S03b] A.S. Bahora et al. Integrated peer-to-peer applications for advanced emergency response systems. part ii. technical feasibility. In *Systems and Information Engineering Design Symposium, 2003 IEEE*, pages 261–268, 24-25 April 2003.

[AS03c] Baruch Awerbuch and Christian Scheideler. Peer-to-peer systems for prefix search. In *In Proceedings of the Symposium on Principles of Distributed Computing*, pages 123–132, 2003.

[BAS04] A.R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. SIGCOMM. *Commun. Rev*, 34(4):353–366, 2004.

[BKK09] Dirk Bradler, Lachezar Krumov, and Jussi Kangasharju. Hierarchical data access in structured p2p-networks. In *Communication and Networking Simulation Symposium*, Mär 2009.

[BKM08] Dirk Bradler, Jussi Kangasharju, and Max Mühlhäuser. Evaluation of peer-to-peer overlays for first response. *Mobile Peer-to-Peer Computing MP2P'08, in conjunction with the 6th IEEE International Conference on Pervasive Computing and Communications (PerCom'08)*, Apr 2008.

[BKM09a] Dirk Bradler, Jussi Kangasharju, and Max Mühlhäuser. Demonstration of first response communication sandbox. In *IEEE Consummer Communications and Networking Conference*, Jan 2009.

[BKM09b] Dirk Bradler, Jussi Kangasharju, and Max Mühlhäuser. Optimally efficient multicast in structured peer-to-peer networks. Jan 2009.

[BKMR07] O. Beaumont, A.M. Kermarrec, L. Marchal, and E. Riviere. VoroNet: A scalable object network based on Voronoi tessellations. *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10, 2007.

[BL07] J.Q. Bao and W.C. Lee. Rapid deployment of wireless ad hoc backbone networks for public safety incident management. pages 1217–1221, Nov. 2007.

[BMT$^+$98] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song. Parsec: A parallel simulation environment for complex systems. *Computer*, pages 77–85, 1998.

[Bol01] Béla Bollobás, editor. *Random Graphs*. Cambridge University Press, 2001.

[BPSM08] D. Bradler, K Panitzek, I. Schweizer, and M. Muehlhaeuser. First response communication sandbox. *11th Communications and Networking Simulation Symposium*, 2008.

[Bra01] U. Brandes. A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[BSAL09] Dirk Bradler, Benjamin Schiller, Erwin Aitenbichler, and Nicolas Liebau. Towards a Distributed Crisis Response Communication System. In *Proceedings of the Conference on Information Systems for Crisis Response and Management*, 2009.

[BST89] H.E. Bal, J.G. Steiner, and A.S. Tanenbaum. Programming languages for distributed computing systems. *ACM Computing Surveys (CSUR)*, 21(3):322, 1989.

[bTSC02] Prepared by Titan Systems Corporation. Arlington county after-action report on the response to the september 11 terrorist attack on the pentagon. 2002.

[BVV03] R. Bhagwan, G. Varghese, and G. Voelker. Cone: Augmenting DHTs to support distributed resource discovery. Technical report, Citeseer, 2003.

[CDK05] G.F. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems: concepts and design*. Addison-Wesley Longman, 2005.

[Cea03a] M. Castro et al. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *INFOCOM 2003*, volume 2, pages 1510–1520, March 2003.

[Cea03b] M. Castro et al. SplitStream: High-bandwidth multicast in a cooperative environment. Lake Bolton, NY, October 2003.

[CI09] G. Camarillo and IAB. Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability. RFC 5694 (Informational), November 2009.

[CK03] Guanling Chen and David Kotz. Context-aware resource discovery. In *PerCom*, pages 243–252. IEEE Computer Society Press, March 2003.

[CRB$^+$03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like P2P systems scalable. *in conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, 2003.

[CRR$^+$05] Yatin Chawathe, Sriram Ramabhadran, Sylvia Ratnasamy, Anthony LaMarca, Scott Shenker, and Joseph Hellerstein. A case study in building layered dht applications. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 97–108, New York, NY, USA, 2005. ACM.

[CSRU05] R. Chen, R. Sharman, H.R. Rao, and S. Upadhyaya. Design principles of coordinated multi-incident emergency response systems. In *IEEE international conference on intelligence and security informatics*. Springer, 2005.

[Dan06] Ch. Dangalchev. Residual Closeness in Networks. *Phisica A*, 365:556–564, 2006.

[DGH$^+$87] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

[Dir08] Dirk Bradler and Jussi Kangasharju and Max Mühlhäuser. Systematic First Response Use Case Evaluation. *Workshop on Mobile and Distributed Approaches in Emergency Scenarios*, 2008.

[DLS$^+$] F. Dabek, J. Li, E. Sit, J. Robertson, M.F. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput.

[dWG03] C. de Waal and M. Gerharz. Bonnmotion: A mobility scenario generation and analysis tool. *Communication Systems group, Institute of Computer Science IV, University of Bonn*, 2003.

[DZD$^+$03] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. *Lecture Notes in Computer Science*, pages 33–44, 2003.

[EGKM04] P. T. Eugster, R. Guerraoui, A. M. Kermarrec, and L. Massoulie. From Epidemics to Distributed Computing. *IEEE Computer*, 37(5):60–67, 2004.

[FB96] N. Freed and N. Borenstein. *RFC 2045:Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, November 1996.

[Fre77] L. C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, March 1977.

[Gea05] P. García et al. PlanetSim: A New Overlay Network Simulation Framework. *Lecture Notes in Computer Science (LNCS), Software Engineering and Middleware (SEM)*, 3437:123–137, 2005.

[GHW07] C. Greenhill, F.B. Holt, and N. Wormald. Expansion properties of a random regular graph after random vertex deletions. *Arxiv preprint math.CO/0701863*, 2007.

[GKM$^+$07] Kalman Graffi, Aleksandra Kovacevic, Patrick Mukherjee, Michael Benz, Christof Leng, Dirk Bradler, Julian Schröder-Bernhardi, and Nicolas Liebau. Peer-to-peer forschung - überblick und herausforderungen. *it - Information Technology (Methods and Applications of Informatics and Information Technology)*, 46(3):272–279, Jul 2007.

[Gna02] O.D. Gnawali. *A Keyword-Set Search System for Peer-to-Peer Networks*. PhD thesis, Massachussets Institute of Technology, 2002.

[Gro04] Groove Networks. U.S. Army Corps of Engineers Awards Groove Networks \$4.1 Million Multi-Year Contract for Virtual Teaming Solution. *U.S. Army*, September 2004.

[Gua92] John Guare. Six Degrees of Separation. *Dramatists Play Service*, 1992.

[GV06] A. Gellert and L. Vintan. Person Movement Prediction Using Hidden Markov Models. *Studies in Informatics and Control*, 15(1):17, 2006.

[Hel] H. Hellbrück. ANSim-Ad-Hoc Network Simulation Tool.

[HHNL07] Y. Huang, W. He, K. Nahrstedt, and WC Lee. Requirements and system architecture design consideration for first responder systems. In *2007 IEEE Conference on Technologies for Homeland Security*, pages 39–44, 2007.

[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 *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, 2003.

[HK91] MG Hluchyj and MJ Karol. Shuffle Net: an application of generalized perfect shuffles tomultihop lightwave networks. *Lightwave Technology, Journal of*, 9(10):1386–1397, 1991.

[I. 01] I. Stoica et al. Chord: A scalable peer-to-peer lookup service for Internet applications. San Diego, CA, August 27–31, 2001.

[Inc02] N. Inc. The napster homepage. *Online: http://www. napster. com*, 2002.

[int05] intel. Intel pledges 1500 pcs, wireless access points, technical support for hurricane katrina disaster relief efforts. September 2005.

[J. 02] J. Rosen et al. The future of command and control for disaster response. *Engineering in Medicine and Biology Magazine, IEEE*, 21(5):56–68, Sept.-Oct. 2002.

[JBRAS03] A. Jardosh, E.M. Belding-Royer, K.C. Almeroth, and S. Suri. Towards realistic mobility models for mobile ad hoc networks. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 217–229. ACM New York, NY, USA, 2003.

[JFY05] Yuh-Jzer Joung, Chien-Tse Fang, and Li-Wei Yang. Keyword search in dht-based peer-to-peer networks. volume 00, pages 339–348, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[JGZ03] S. Jiang, L. Guo, and X. Zhang. LightFlood: an efficient flooding scheme for file search in unstructured peer-to-peer systems. *in International Conference Parallel Processing*, pages 627–635, 2003.

[JTWW02] M.B. Jones, M. Theimer, H. Wang, and A. Wolman. Unexpected complexity: Experiences tuning and extending CAN. *Microsoft Research, Tech. Rep. MSR-TR-2002-118*, 2002.

[JY06] Yuh-Jzer Joung and Li-Wei Yang. Multi-dimensional prefix search in p2p networks. volume 0, pages 67–68, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[Kan07] Kangasharju, J. et al. ChunkSim: Simulating Peer-to-Peer Content Distribution. *Communications and Networking Simulation Symposium*, 2007.

[KB] Y. Kulbak and D. Bickson. The emule protocol specification. *eMule project, http://sourceforge. net.*

[KDG03] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based Computation of Aggregate Information. *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2003.

[KK03] M.F. Kaashoek and D.R. Karger. Koorde: A simple degree-optimal distributed hash table. *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, pages 98–107, 2003.

[KKD04] D. Kempe, J. Kleinberg, and A. Demers. Spatial Gossip and Resource Location Protocols. *Journal of the ACM (JACM)*, 51(6):943–967, 2004.

[KKH$^+$06] A. Kovacevic, S. Kaune, H. Heckel, A. Mink, K. Graffi, O. Heckmann, and R. Steinmetz. Peerfactsim. kom-a simulator for large-scale peer-to-peer networks. *Technische Universitat Darmstadt, Germany, Tech. Rep. Tr-2006-06*, pages 2006–06, 2006.

[KLBA] F. Klemm, J.Y. Le Boudec, and K. Aberer. Congestion control for distributed hash tables. In *Fifth IEEE International Symposium on Network Computing and Applications, 2006. NCA 2006*, pages 189–195.

[KM02] T. Klingberg and R. Manfredi. The gnutella protocol 0.6, 2002.

[KMW04] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Unit Disk Graph Approximation. *Proceedings of the 2004 Joint Workshop on Foundations of Mobile Computing*, pages 17–23, 2004.

[Kov07] Kovacevic, A. et al. Benchmarking platform for peer-to-peer systems. *it - Information Technology*, 49(5):312–319, Sep 2007.

[KTL$^+$09] A. Kovacevic, A. Todorov, N. Liebau, D. Bradler, and Ralf Steinmetz. Demonstration of a peer-to-peer approach for spatial queries. In *Proceedings of Database Systems for Advanced Applications (DASFAA'09)*, Apr 2009.

[KWA$^+$01] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. Seti@ home-massively distributed computing for seti. *Computing in Science & Engineering*, 3(1):78–83, 2001.

[Lam03] L. Lamport. Quarterly quote. *ACM SIGACT News*, 34, 2003.

[LKR05] J. Liang, R. Kumar, and K. W. Ross. The KaZaA overlay: A measurement study. *Computer Networks*, 49(6), 2005.

[LKRG03] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 395–406, 2003.

[LLH$^+$03] J. Li, B.T. Loo, J. Hellerstein, F. Kaashoek, D.R. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[LXKL06] Xiaomei Liu, Li Xiao, Andrew Kreling, and Yunhao Liu. Optimizing Overlay Topology by Reducing Cut Vertices. In *Proc. of the 2006 Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 1–6, Newport, Rhode Island, 2006. ACM.

[MB07] B.S. Manoj and Alexandra Hubenko Baker. Communication challenges in emergency response. *Commun. ACM*, 50(3):51–53, 2007.

[MBR03] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: distributed hashing in a small world. In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. Cambridge, MA, March 2002.

[MM07] B. Milic and M. Malek. Analyzing large scale real-world wireless multihop network. *IEEE Communications Letters*, 11(7):580–582, 2007.

[MM09] B. Milic and M. Malek. Properties of Wireless Multihop Networks in Theory and Practice. *Guide to Wireless Ad Hoc Networks*, pages 1–26, 2009.

[MNR02] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, 2002.

[Ngu] Nguyen, N.T. et al. Learning people movement model from multiple cameras for behaviour recognition. *Joint IAPR International Workshops on Structural and Syntactical Pattern Recognition and Statistical Techniques in Pattern Recognition*, pages 315–324.

[of06] Department ofHomelandSecurity. Public Safety Statement of Requirements for Communications & Interoperability - Qualitative. *In The SAFECOM Program*, 1, 2006.

[Pla99] CG Plaxton. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. *Theory of Computing Systems*, 32(3):241–280, 1999.

[QCC07] Tongqing Qiu, E. Chan, and Guihai Chen. Overlay Partition: Iterative Detection and Proactive Recovery. In *IEEE International Conference on Communications 2007 (ICC'07)*, pages 1854–1859, 2007.

[RD01a] 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)*, Heidelberg, Germany, November 2001.

[RD01b] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[RFH$^+$01a] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. *Proceedings of the 2001 SIGCOMM conference*, pages 161–172, 2001.

[RFH$^+$01b] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. San Diego, CA, August 2001.

[RHKS01] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. *Proceedings of NGC*, 2001.

[RHRS04] Sriram Ramabhadran, Joseph Hellerstein, Sylvia Ratnasamy, and Scott Shenker. Brief announcement: Prefix hash tree - an indexing data structure over distributed hash tables. In *ACM PODC*, 2004.

[RLS$^+$03] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. *Proceedings of IPTPS*, 2003.

[RV03a] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Middleware*, 2003.

[RV03b] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. *Proceedings of International Middleware Conference*, pages 21–40, 2003.

[S. 04] S. Rhea et al. Handling churn in a DHT. In *Proc. of the 2004 Usenix Annual Technical Conference*, June 2004.

[S. 05] S. Krishnamurthy et al. A statistical theory of chord under churn. In *IPTPS*, pages 93–103, 2005.

[S$^+$01] R. Schollmeier et al. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. *Peer-to-Peer Computing*, pages 101–102, 2001.

[Sab66] G. Sabidussi. The Centrality Index of a Graph. *Psychometrika*, 31:581–603, 1966.

[Sam84] H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[SG00] C. Stauffer and WEL Grimson. Learning patterns of activity using real-time tracking. *Pattern Analysis and Machine Intelligence, IEEE*, 22(8):747–757, 2000.

[SGG02] S. Saroiu, K. P. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing networks. In *Proceedings of Multimedia Computing and Networking*, San Jose, CA, Jan 2002.

[SGG03] S. Saroiu, K.P. Gummadi, and S.D. Gribble. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Systems*, 9(2):170–184, 2003.

[SLS06] Min Sheng, Jiandong Li, and Yan Shi. Critical Nodes Detection in Mobile Ad Hoc Network. *Advanced Information Networking and Applications*, 2:336–340, 2006.

[SMB01] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. *in Proceedings IPTPS*, 2001.

[Smi03] S. Smith. From Napster to Kazaa: The battle over peer-to-peer filesharing goes international. *Duke L. & Tech. Rev.*, 2003:8, 2003.

[SW05] R. Steinmetz and K. Wehrle, editors. *Peer-to-Peer Systems and Applications*. LNCS. Springer, 2005.

[Tho06] A. Thomann. Skype, A Baltic Success Story. *Emagazie Newsletter–Credit Suisse*, 2006.

[TKLB07] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. In *SIGCOMM*, Kyoto, Japan, August 2007.

[TLB07] W. W. Terpstra, C. Leng, and A. P. Buchmann. Brief Announcement: Practical Summation via Gossip. In *PODC*. PODC, 2007.

[V$^+$01] A. Varga et al. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM2001)*, pages 319–324, 2001.

[www06] www.nortelgov.com. Positioning paper wimax for government-grade secure mobility. July 2006.

[WZ04] R. Wattenhofer and A. Zollinger. XTC: A Practical Topology Control Algorithm for Ad-Hoc Networks. *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, 2004.

[YDRC06] Y. Yang, R. Dunlap, M. Rexroad, and B.F. Cooper. Performance of full text search in structured and unstructured peer-to-peer systems. *IEEE INFOCOM 2006*, 2006.

[ZBG98] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. *ACM SIGSIM Simulation Digest*, 28(1):154–161, 1998.

[Zea04] B. Y. Zhao et al. Tapestry: A resilient global-scale overlay for service deployment. *JSAC*, 22(1):41–53, January 2004.

[Zim80] H. Zimmermann. OSI reference model–The ISO model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28(4):425–432, 1980.

[ZKJ01] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. *Computer*, 74, 2001.

[ZSZ03] Z. Zhang, S. Shi, and J. Zhu. Somo: Self-organized metadata overlay for resource management in p2p dht, 2003.

[ZZ03] L. Zhuang and F. Zhou. Understanding Chord Performance. Technical Report CS268, Department of Computer Science, UC Berkeley, 2003.

Erklärung[1]

Hiermit erkläre ich, die vorgelegte Arbeit zur Erlangung des akademischen Grades "Dr. rer. nat." mit dem "Peer-to-Peer Concepts for Emergency First Response" selbständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel erstellt zu haben. Ich habe bisher noch keinen Promotionsversuch unternommen.

Darmstadt, 26. April 2010 Dirk Bradler

[1]gemäß §9 Abs. 1 der Promotionsordnung der TU Darmstadt

Wissenschaftlicher Werdegang des Verfassers[2]

10/2000 – 05/2006	Studium der Wirtschaftsinformatik an der TU Darmstadt Abschluss: Diplom Wirtschaftsinformatiker Diplomarbeitsthema: Deploying and Managing Distributed Services in a Peer-to-Peer Network
seit 06/2006	Wissenschaftlicher Mitarbeiter im Fachbereich Informatik an der TU Darmstadt

[2]gemäß §20 Abs. 3 der Promotionsordnung der TU Darmstadt