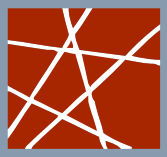


Network Architectures
and Services
NET 2010-07-1

Dissertation

Architecture and Components of secure and anonymous Peer-to-Peer Systems

Heiko Niedermayer

Network Architectures and Services
Department of Computer Science
Technische Universität München

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Architecture and Components of secure and anonymous Peer-to-Peer systems

Heiko Niedermayer

Vollständiger Abdruck der von der Fakultät für Informatik
der Technischen Universität München
zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Johann Schlichter
Prüfer der Dissertation:	1.	Univ.-Prof. Dr. Georg Carle
	2.	Univ.-Prof. Dr. Claudia Eckert

Die Dissertation wurde am 13.10.2009 bei der Technischen Universität München eingereicht
und durch die Fakultät für Informatik am 21.05.2010 angenommen.

Cataloging-in-Publication Data
Heiko Niedermayer
Architecture and Components of secure and anonymous Peer-to-Peer Systems
Dissertation, July 2010
Network Architectures and Services, Department of Computer Science
Technische Universität München

ISBN 3-937201-13-0
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)
Network Architectures and Services NET 2010-07-1
Series Editor: Georg Carle, Technische Universität München, Germany
© 2010, Technische Universität München, Germany

Abstract

Changes to lower layers of the Internet architecture are difficult, in particular on network and transport layer. Standardization and economic incentives for manufacturers and Internet service providers are necessary. As a consequence, many desirable services cannot be realized there. The application layer is more flexible. End-users can run their own services and networked applications. This led to the advent of Peer-to-Peer systems in the late 1990ies. Peer-to-Peer systems enable normal users to provide the services instead. Despite the subsequent hype and research, the Peer-to-Peer paradigm is not yet a fully mature technology. In particular, it faces severe and fundamental security issues.

This dissertation covers three major topics: Peer-to-Peer, Security, and Anonymity. We will present and evaluate components and a variety of aspects of the architecture of Peer-to-Peer systems. Many classifications of Peer-to-Peer systems are not suitable for describing or designing a system. There is often a focus on a small number of aspects, that do not cover all problems from bootstrapping over routing to security. We see Peer-to-Peer systems as a combination of a set of solutions. Each solution solves a particular problem and some common problems are presented as categories. This is not only suitable for filesharing applications, but also valid for commercial or future Peer-to-Peer applications. Description is, however, not our primary goal. We think that structuring the problem of a Peer-to-Peer system into necessary components helps to design Peer-to-Peer systems and to structure the problems that need to be solved.

Optimization is another issue. In case of Peer-to-Peer systems this includes a suitable distribution of load among the peers and the reduction of distances (latency) in the network. Systems do not necessarily end up with a uniform load. In fact, we show that most systems will not. We discuss load balancing for a distributed gaming application. A particular problem for load balancing is also that items usually do not have a uniform weight. Given the common Power Law distributions some items may have an extreme weight so that load balancing can only succeed if the handling of this item is further distributed among multiple nodes. For the optimization of distances we looked at Proximity Node Selection and evaluated its impact. We created a diagram for the selection of solutions for both problems when designing a system.

Security is a big problem for Peer-to-Peer systems and many security issues are caused by inherent properties of these systems. We categorize the security problems and introduce the Cheapriding attack. It is a variant of freeriding and operates against reputation systems. It can hardly be stopped completely. The basic idea to mitigate the attack is to adapt the benefit gained for a good action in a reputation systems as good as possible to the actual cost of the action. The attack may be detected by checking the weight of positive and negative feedback. Security in this thesis is not limited to security purely for Peer-to-Peer systems. The performance of cryptographic operations and transport protocols goes beyond the scope of Peer-to-Peer systems. We conclude that performance of symmetric cryptography is not a bottleneck on today's hardware. In our studies hash functions were often more expensive than encryption. Also from this perspective it is good that

NIST is currently organizing a competition for a new hash function standard. Public Key cryptography and Identity-based cryptography are both similar expensive and way more expensive than symmetric cryptography. However, given a moderate usage they are no problem for today's computers.

A key part in our work is about new ways to realize authentication. Our focus is again on Peer-to-Peer systems. However, the fundamental problem can also be found on human layer as well as in all Web2.0-alike networks where humans interact with each other instead of purely consuming services. Our approach fills a gap between the rather unrealistic or slightly insecure decentralized approaches and the secure centralized approaches. In the latter, a central authority exists and provides the base for authentication and other security services. We build our work on social clusters of humans that can be used to partition the network into domains. Each domain has a centralized authority. The more scalable level of the domains is used to establish trust between different domains. This requires that the protocols as well as software that uses them need to be able to deal with the uncertainty when no trust yet exists. For the authentication we developed two authentication protocols that include all necessary four parties (A and B as well as their local authorities).

Finally, anonymity is another domain where the Peer-to-Peer paradigm is a reasonable choice. We describe the fundamental operation of anonymization networks and have a special look at the system I2P. The anonymization concept MORE was in parts co-authored by the author of this thesis. We will describe the system and analyze it. MORE was developed to protect client and server and to fight pattern-based attacks. The analysis will show that even the radical approach of MORE is not sufficient to completely prevent this kind of attacks.

Zusammenfassung

Die Einführung neuer Dienste im Internet ist schwierig, sei es auf Netzwerk- oder auch Transportschicht. Dies erfordert Standardisierung in den entsprechenden Gremien. Ohne signifikante wirtschaftliche Anreize für Hersteller und Internet-Service-Provider ist darüberhinaus mit keiner weitreichenden Einführung zu rechnen. Aus diesem Grund können viele wünschenswerte Dienste, beispielsweise Internet-weiter Multicast, dort nicht realisiert werden. Die Alternative ist die Realisierung dieser Funktionalität auf der Anwendungsschicht. Diese ist deutlich flexibler und erlaubt auch normalen Anwendern die Gestaltung eigener Dienste und verteilter Anwendungen. Der Aufstieg der Peer-to-Peer-Systeme zur zeitweise größten Verkehrsquelle im Internet ist ein Ergebnis dieser Situation. Dennoch ist das Peer-to-Peer-Paradigma keine vollverstandene Technik. Insbesondere hat sie mit schwerwiegenden und fundamentalen Sicherheitsproblemen zu kämpfen.

In dieser Arbeit werden drei große Themenbereich bearbeitet und miteinander verknüpft. Dies sind Peer-to-Peer-Systeme, Netzwerksicherheit und Anonymität. Zuerst werden Komponenten und verschiedene Aspekte der Architektur von Peer-to-Peer-Systemen vorgestellt und evaluiert. Viele Klassifizierungen von Peer-to-Peer-Systemen eignen sich nur wenig zur eigentlichen Systembeschreibung oder als Architekturhilfe. Insbesondere vernachlässigen sie die ganzheitliche Betrachtung der Systeme, die vom Bootstrapping über Routing bis zur Sicherheit reicht. Aus diesem Grund werden hier Peer-to-Peer-Systeme als eine Kombination einer Menge von kleineren Problemstellungen mit verschiedenen Lösungen als einzelne Bausteine betrachtet. Diese Betrachtung eignet sich darüberhinaus nicht nur für klassische Filesharingnetze, sondern ist generisch genug, um zukünftige wie auch kommerzielle Systeme beschreiben zu können. Die Beschreibung ist allerdings nur ein sekundäres Ziel, vielmehr soll durch die Betrachtung der notwendigen Komponenten die Architektur von Systemen erleichert und strukturierter werden.

Zur Architektur von Systemen gehört nebem dem abstrakten Entwurf auch die Optimierung der Leistungsfähigkeit. Im Falle von Peer-to-Peer-Systemen beinhaltet dies u.a. die geeignete Verteilung von Last sowie die Optimierung der Abstände (Latenzen) im Netzwerk. In der Arbeit wird dazu gezeigt, dass sich nicht zwingend eine gleichförmige Belastung der Systeme einstellt. Für eine virtuelle Spieleanwendung wurde gezeigt, wie sich Last optimieren lässt. Darüberhinaus ist zu beachten, dass einzelne Elemente im System ein starkes Gewicht haben können und die Verteilung einer Aufgabe auf mehrere Knoten daher oft notwendig erscheint. Zur Optimierung der Distanzen im Netzwerk wurde Proximity Node Selection untersucht und wie auch für die Lastbalancierung ein Entscheidungsablauf für den Systementwurf erstellt.

Sicherheit in Peer-to-Peer-Systemen ist ein großes Problem, welches durch die inhärenten Eigenschaften der Systeme verursacht wird. In der Arbeit findet sich eine Kategorisierung der Probleme. Darüberhinaus wird der Cheapriding-Angriff vorgestellt, welcher als eine Variante des Freeridings angesehen werden kann. Dieser Angriff geht gegen Reputationssysteme und kann nur bedingt erfolgreich bekämpft werden. Die Anpassung der Belohnung im Reputationssystem an tatsächliche Kosten ist ein Weg, die Auswirkungen des

Angriffs zu minimieren. Sicherheit wird im folgenden dann auch unabhängig von Peer-to-Peer-Systemen betrachtet. Verschiedene kryptographische Verfahren werden auf ihre Geschwindigkeit hin untersucht. Dabei zeigt sich, dass symmetrische Kryptographie auf heutigen Rechner kein Engpass mehr darstellt. Oft sind Hashfunktionen schon teurer und es zeigt sich, dass das NIST auch aus Leistungssicht gut daran tut, einen Wettbewerb zur Ermittlung eines neuen Hashfunktionsstandards durchzuführen. Public-Key-Kryptographie wie auch identitätsbasierte Kryptographie sind in etwa gleich teuer und sollten durch ihre hohen Kosten nicht übermäßig verwendet werden. Bei moderater Verwendung sind aber auch sie kein Problem mehr bei heutiger Technik.

Ein zentraler Abschnitt der Arbeit geht um neue Verfahren zur Authentisierung. Der Schwerpunkt wird hier auch auf Peer-to-Peer-Systeme gelegt. Allerdings findet sich das darunterliegende Problem auch generell auf menschlicher Ebene sowie im Netzwerk wieder, wenn Menschen wie im Web2.0 miteinander agieren statt nur zu konsumieren. Der vorgeschlagene Ansatz baut dabei eine Brücke zwischen wenig realistischen und eher unsicheren verteilten Ansätzen und dem sicheren zentralen Ansatz, welcher auf einer zentralen Autorität beruht. Soziale Cluster von Menschen teilen dabei die Knoten des Netzwerks in Teilmengen ein, welche wir Domain nennen und welche selbst eine lokale zentrale Autorität beinhalten. Auf dieser skalierbareren Ebene wird dann Vertrauen zwischen Domänen aufgebaut. Da dies notwendigerweise mit zeitweise unsicheren Beziehungen einher geht, wird der Beziehungszustand bei der Authentisierung den beteiligten Peers und deren Software mitgeteilt, so dass diese für die jeweilige gerade laufende Anwendung entscheiden können, ob sich auch mit Unsicherheiten umgehen mögen oder die Autorisierung verweigern. Für den Authentisierungsablauf wurden dafür zwei angepasste kryptographische Protokolle mit den notwendigen vier Parteien (A und B sowie deren Autoritäten) entwickelt und evaluiert.

Anonymität ist eine weitere Domäne, in der das Peer-to-Peer-Paradigma sinnvoll eingesetzt werden kann. Hierzu wird einerseits die Funktionsweise von Anonymisierungsnetzen beschrieben und am Beispiel des Peer-to-Peer-Anonymisierers I2P näher beleuchtet. Das Anonymisierungskonzept MORE wurde teilweise vom Autor mitentwickelt und wird im abschließenden Kapitel beschrieben und nachfolgend analysiert. MORE wurde entworfen, um beide, Client und Server, zu schützen und Muster-basierte Angriffe zu vermeiden. Es zeigt sich, dass selbst der radikale Ansatz von MORE diese Angriffe nicht komplett vermeiden kann.

Acknowledgements

This thesis would not have been possible without many people in Tübingen (where it started) and in Munich (where it now ended). My special thanks go to Prof. Georg Carle for supervising the dissertation as well as to the rest of the examination committee, Prof. Claudia Eckert and Prof. Johann Schlichter. I cannot name all the numerous students and colleagues that provided valuable input in discussions and collaborations (to name a few: Marc Fouquet, Ralph Holz, Ali Fessi, Johannes Riedel, Olaf Landsiedel, Simon Rieche). Special thanks also go to Prof. Peter Hauck in Tübingen for supervising joint work on cryptographic protocols and security as well as to Prof. Klaus-Jörn Lange and PD Dr. Klaus Reinhardt in Tübingen for joint seminars on formal methods in network security. I am also thankful for stimulating discussions with fellow computer scientists in Tübingen at the Sand bus stop about logic and philosophy.

Finally, I would like to thank my family for supporting me during my time as a student and my friends for providing a world without academic discussions on computer science.

Published Work

This is a list of the subset of the author's publications that is related to certain chapters of the dissertation. Other publications that do not address content that is also addressed by this text are not listed.

Chapter 3

Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Timo Teifel, and Georg Carle. Clustering players for load balancing in virtual worlds. International Journal of Advanced Media and Communication (IJAMC), 2(4):351-363, December 2008.
Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Timo Teifel, and Georg Carle. Clustering players for load balancing in virtual worlds. In Proceedings of 1st International Workshop on Massively Multiuser Virtual Environments at IEEE Virtual Reality 2008 (MMVE 2008), Reno, Nevada, USA, March 2008.
Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Leo Petrak, and Georg Carle. Peer-to-peer-based infrastructure support for massively multiplayer online games. In Proceedings of 4th Annual IEEE Consumer Communications and Networking Conference (CCNC 2007), Las Vegas, Nevada, USA, January 2007.
Simon Rieche, Marc Fouquet, Heiko Niedermayer, Leo Petrak, Klaus Wehrle, and Georg Carle. Peer-to-peer-based infrastructure support for massively multiplayer online games. Technical Report WSI-2006-04, Wilhelm-Schickard-Institute for Computer Science, University of Tübingen, Tübingen, Germany, August 2006.
Heiko Niedermayer, Simon Rieche, Klaus Wehrle, and Georg Carle. On the distribution of nodes in distributed hash tables. In KiVS 2005 (Workshop Peer-to-Peer-Systeme und Anwendungen), Kaiserslautern, Germany., March 2005.

Chapter 4

Heiko Niedermayer, Klaus Wehrle, Thomas Schreiner, and Georg Carle. Considering security in distributed hash tables. In Abstract at 4th Würzburger Workshop IP Netzmanagement, IP Netzplanung und Optimierung, Würzburg, Germany., July 2004.

Chapter 5

Heiko Niedermayer, Andreas Klenk, and Georg Carle. The networking perspective of security performance - a measurement study -. In MMB 2006, Nürnberg, Germany, March 2006.

Chapter 6

Heiko Niedermayer, Ralph Holz, Marc-Oliver Pahl, and Georg Carle. On Using Home Networks and Cloud Computing for a Future Internet of Things. In Proc. Future Internet Symposium 2009 (FIS 2009), Berlin, Germany, September 2009.
Ralph Holz, Heiko Niedermayer, Peter Hauck, and Georg Carle. Trust-rated authentication for domain-structured distributed systems. In Proc. 5th European PKI Workshop: Theory and Practice (EuroPKI 2008), Trondheim, Norway, 2008.
Ralph Holz and Heiko Niedermayer. A Protocol for Inter-Domain Authentication with a Trust-Rating Mechanism. In 8. Kryptotag der GI-Fachgruppe KRYPTO (Workshop). Technical Report WSI-2008-02. University of Tübingen, April 2008.

Chapter 8

Olaf Landsiedel, Alexis Pimenidis, Klaus Wehrle, Heiko Niedermayer, and Georg Carle. Dynamic multipath onion routing in anonymous peer-to-peer overlay networks. In Proceedings of IEEE Global Communication Conference (GlobeCom), Washington, DC, USA, November 2007.
Olaf Landsiedel, Simon Rieche, Heiko Niedermayer, Klaus Wehrle, and Georg Carle. Anonymous ip-services via overlay routing. In Extended Abstract at 5th Würzburger Workshop IP Netzmanagement,IP Netzplanung und Optimierung, Würzburg, July 2005.
Olaf Landsiedel, Klaus Wehrle, and Heiko Niedermayer. An infrastructure for anonymous internet services. In International Workshop on Innovations In Web Infrastructure (IWI2005), 14th International World Wide Web Conference - WWW2005, Chiba/Tokyo, Japan, May 2005.

Remarks

For all our work with multiple contributors, we (= the author) focus on our input and contributions as far as possible, and provide a general description of other aspects as far as necessary for the overall understanding.

Introduction

Peer-to-Peer systems have been hyped since the advent of Napster and its impact on sharing music and other content. Scientific research has shown that the Peer-to-Peer concept can be useful for many other applications and services as well. Most prominent today is the Instant-Messaging and Voice-over-IP service Skype. On the Internet today Peer-to-Peer applications contribute to almost 50 % of the traffic.

While network research and innovation often neglect security, real systems have to deal with their security sooner or later. The term security is rather general as threats are diverse and very different in nature. The most important issue is to ensure that the network behaves as expected. Attackers or a 'wrong' entity should not obtain certain information or interfere with the information exchange.

A more subtle form of security is anonymity. A lot of information about a human being can be found on the Internet or in databases. In particular, the increasing misuse of publicly available information on the Internet leads to an increased need for anonymity. Some propose to make all information short-lived, but that is not what people want. Another option is to uncouple information from its source. This can be achieved partially via classic security mechanisms. Anonymity networks hide sender, receiver, and their relation from other entities and each other. This is useful for keeping information private where the contact information reveals details that should not be known to anyone.

Contents

1. Introduction to Peer-to-Peer Systems

1.1 Introduction

In this chapter, we will introduce the term Peer-to-Peer and present a short selection of related work on the issues discussed in the later Peer-to-Peer chapters.

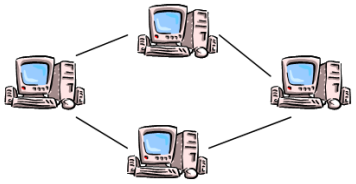


Figure 1.1: Peers form a network.

There are many definitions and thoughts about what is Peer-to-Peer and what is not Peer-to-Peer. We give a definition that is rather flexible:

Definition 1.1 (Peer-to-Peer) *Peer-to-Peer networks are networks among equals (peers) where there is no predefined distinction between client (requesting service) and server (providing service). These peers of a Peer-to-Peer network organize themselves and resources are contributed cooperatively by participating parties.*

Many people in public as well as academia associate the term Peer-to-Peer with filesharing networks. From our point of view, this is a too narrow understanding, in particular as other Peer-to-Peer systems already exist.

A curious yet also slightly problematic aspect of the term Peer-to-Peer is that many of the well-known Peer-to-Peer systems are not purely Peer-to-Peer. There are many networks where not all peers are equal. Some (e.g. Napster[Naps] or Skype[Skyp08]) may even use central servers for at least some of the tasks. This makes sense as some tasks are better performed by cooperating peers, others are better performed by central servers. As a consequence, we do not demand that everything in a network needs to be decentralized, but rather say, that some tasks are Peer-to-Peer and some are not.

Peer-to-Peer networks take advantage of resources on the edge of the Internet, say the computers of users, the users themselves, etc. Typical characteristics of Peer-to-Peer systems directly follow from this fact. End systems do not primarily serve the purpose of the Peer-to-Peer system. Thus, the resources of peers must not be exhausted by the Peer-to-Peer network. End systems are also not necessarily meant to be publicly available machines. Thus, connectivity might be a problem as well as security. Moreover, end systems are not always-on. Thus, the environment is considered to be less stable and more dynamic than for the traditional client-server case. Another issue is that the systems are run by users who are no experts. As a consequence Peer-to-Peer systems should exhibit aspects of self-organization. The system design needs to incorporate these properties.

In the following, we give some examples on how real systems decide for themselves, where to be Peer-to-Peer or where not to be Peer-to-Peer.

Napster
Napster originally was a filesharing system with a central server where peers registered themselves and their items (= shared files).

- **Not Peer-to-Peer**: Lookup, Maintenance
- **Peer-to-Peer**: Data Transfer

Internet Auctions
Typical auction sites on the Internet, e.g. eBay, operate as follows.

- **Not Peer-to-Peer**: The platform itself (auctions, accounts, information transfer, reputation system)
- **Peer-to-Peer**: Money and goods exchange, input for reputation system (users rate the behaviour of the other user after a transaction)

Skype
Skype is a current Voice-over-IP service with a broad user base. It is based on a hybrid Peer-to-Peer architecture (peers and superpeers) and servers for security and commercial services.

- **Not Peer-to-Peer**: Login, Account Management
- **Peer-to-Peer**: Lookup, User Interaction, Data Exchange

BitTorrent
BitTorrent is a concept and system for the efficent transfer of large files to a large number of users. It is used not only for private filesharing, but also for the distribution of operating systems (Linux) and software updates (e.g. by the game World of Warcraft).

- **Not Peer-to-Peer**: Download of .torrent file from a webserver. Peers find other currently active peers of a torrent via the tracker, which is (in the original protocol) a server[1].
- **Peer-to-Peer**: The file transfer is completely cooperative and self-organizing between all peers interested in the file.

[1]While the tracker as server is still common, there is also a Peer-to-Peer version where based on the peer's identifiers the $n = 8$ closest peers to a torrent identifier perform the tracker operation for a torrent.

Another term that is used quite frequently in the context of Peer-to-Peer networks is the term overlay.

Definition 1.2 (Overlay) *Overlay or overlay network is a term for networks that run on top of an existing infrastructure but provide certain additional functionality.[Over]*

The terms Peer-to-Peer and Overlay are not equivalent, neither by definition nor by the outcome. Nonetheless, they are often used as synonyms, in particular to avoid problems with the decentral nature and filesharing image of the term Peer-to-Peer.

Most Peer-to-Peer systems are overlays. Many overlays are Peer-to-Peer systems. The Internet itself with IP as its protocol is in fact both, an overlay and a Peer-to-Peer system. Virtual Private Networks with gateway servers are examples for overlays that are not Peer-to-Peer. Giving an example for a Peer-to-Peer system that is not a subclass of overlay is not as easy. A group of peers (say students in a lecture) may organize themselves (e.g. where they sit) in a Peer-to-Peer way, but there is no addressing or communication structure on top of the underlay.

1.2 Related Work

In this section, we briefly present work that is related to work presented in the Peer-to-Peer chapters.

1.2.1 Components and Architecture

Aberer et al [AAGG+05] presented a reference architecture for Peer-to-Peer systems. The paper actually calls it the 'Essence of Peer-to-Peer'. According to their work, overlays can be characterized according to six properties.

- **Identifier space for node and item identifiers**, e.g. interval [0,1)
- **A mapping of nodes and items to identifier space**, e.g. sha-1(IP:port), sha-1('Cool video1.mpeg')
- **Management of the identifier space by peers**, e.g. the two closest peers are responsible for an identifier I
- **Graph Embedding**, e.g. Ring topology
- **Routing Strategy**, usually greedy strategy in structured networks ('use neighbor with ID closest to target ID'),
- **Maintenance Strategy**, e.g. methods to handle join, leave, etc.

The work seems to be fundamental and complete at first. When looking at it more precisely one finds that some of the properties represent fundamental mathematical functions, while others are very vague with not one definite answer. We tried to use it for our lecture about Peer-to-Peer systems. When using it there, it was obvious that it a) is focused on DHT-like Peer-to-Peer usage with item-storage as main objective and b) that it cannot be used to differentiate the different Peer-to-Peer approaches as most systems would only differ in one property - primarily the graph embedding, and the value of the property would be more or less the system name or description. Other differences are rather arbitrary. Thus, the granularity is not fine-grained when it comes to fundamental ideas of different systems.

Even more problematic, for hybrid Peer-to-Peer networks or unstructured networks it is hardly applicable. Nonetheless, these networks are the Peer-to-Peer networks that are in world-wide real-world use.

The Peer-to-Peer Common API [DZDK+03] is another work to combine different ideas about Peer-to-Peer networks. The basic observation for the Common API was that the different DHT approaches ended up with similar yet slightly different interfaces. They also observed that typical DHTs logically end up with a two-layered structure on the Peer-to-Peer level.

As consequence, a common API (with APIs for Key-Based Routing, for DHT, etc.) is proposed that represents a standard set of functions with well-defined semantics to ease the development of Peer-to-Peer applications.

The higher layer structure provides the application functionality, in case of a DHT, the DHT API with its put and get operations. The lower layer is the key-based routing layer. All DHTs are based on the routing towards an ID. This functionality is called key-based routing. It is the core idea of the structured Peer-to-Peer approach. All so-called DHTs optimize the protcol for this layer. Some DHTs like Chord e.g. only implement this layer and leave the hash table functionality to the application. The basic functions of the KBR API are *route*, *forward*, and *deliver*. Additionally, there is a set of functions to manage and access the routing table and its state.

The Common API is centered around the concept of Key-based Routing in structured networks and not suitable for other networks. It defines a standard for interfaces, but does not deal with components of a Peer-to-Peer network. In particular it does not deal with the different ideas of the systems as, quite the opposite, it combines the similarities of a variety of systems to one common standard. The current implementations of Pastry have Common API interfaces, but there is no wide other usage.

1.2.2 Performance and Optimization

Most DHTs are based on the idea of Consistent Hashing [KLLP+97]. With respect to load balance most nodes will with high probability serve $O((1 + logn)\frac{k}{n})$ items, given k items and n nodes. The $1 + logn$ can become $1 + \epsilon$ if nodes share their intervals and, thus, average out the variance. The term 'with high probability' means that the probability of the contrary event converges fast enough to 0 with n increasing, e.g. $p_{false} = O(\frac{1}{n})$.

In practise one may not be satisfied with the balance of the load, and apply load balancing. There are various approaches for load balancing in DHTs [RLSK+03, ByCo03, RiPW04a], most notably [KaRu04]. The latter distinguishes between balancing of nodes and balancing of items. The first is important if the item IDs are uniformly drawn. The latter is primarily necessary if item IDs follow a non-uniform distribution.

Another limiting factor for performance is the multihop routing that is targeted towards an ID, but not towards the final machine on the Internet. To reduce the effects of such detours there are three primary options Proximity Identifier Selection (or geographic layout), Proximity Route Selection, and Proximity Node Selection (PNS). PNS seems to be the most preferable one as it can be included in most Peer-to-Peer systems and has been shown to be most effective [CDHR02, GGGR+03].

1.2.3 Security of Peer-to-Peer systems

One of the first observations in research during the Peer-to-Peer hype around 2002 was that security is a concern and that Peer-to-Peer networks are open to many attacks.

Sit and Morris [SiMo02] classify attacks that are possible against a Peer-to-Peer system. There can be Routing attacks, Storage and Retrieval attacks, and miscellaneous attacks. Miscellaneous attacks are attacks that do not fall into the other two categories. A node may behave inconsistently. An attacker can mount DoS attacks using churn or the overloading of nodes with requests. Another option is to send unsolicited message, e.g. to fake the reply to a message seen by the attacker.

The free and open nature of Peer-to-Peer systems allows attacks against nodes, data, routing. The consequence would be to make it simply less free and all is solved. However, fundamental attacks against Peer-to-Peer systems and their properties prohibit such a solution. The first is the Sybil attack presented by Douceur [Douc02]. The second is the Ecplipse attack which was introduced for overlay networks by Castro et al. [CDGR$^+$02].

The Sybil attack An attacker simply registers multiple times in the network. The consequence is that one attacker can take over the majority of the network. A conclusion by Douceur is that the sybil attack cannot be stopped by proof-of-work tests in a realistic setting.

The Eclipse attack can be based on the Sybil attack, but this is not necessary. The idea is to eclipse a group of nodes (or ID space) from the rest of the network, so that all connections or lookups towards the nodes go via nodes of the attacker. This can be done with multiple nodes in combination with the sybil attack. However, overlay maintenance algorithms and false information can also be used to establish an eclipse attack without relying on the Sybil attack.

There are many ideas on how to deal with these attacks. Here we cite some prominent examples [SCDR04][YKGF06][DiHa06][DLLKA05]. However, without a central authority no completely secure solution exists.

1.3 Outlook

In the following chapters we identify key components for Peer-to-Peer architectures. They allow a course- and fine-grained description of a system. Furthermore, we briefly look at the balance of load in Peer-to-Peer systems in a simple and easy-to-understand analysis. We also discuss and check results on optimizing structured Peer-to-Peer systems using Proximy Node Selection. From that, we draw conclusions for the system design. Finally,we discuss security requirements of applications and structure the security problems in slightly more detail. After these chapters that are purely Peer-to-Peer-related, we look at security in Chapters 5 and 6 and at anonymity in Chapters 7 and 8. All of these chapters will also relate their topic to Peer-to-Peer systems, but Peer-to-Peer is not their only or primary concern.

2. Components and Architecture of Peer-to-Peer Systems

2.1 Introduction

In this chapter we discuss components and architectural aspects of Peer-to-Peer systems. The components and their realization in the system can also be used to categorize and briefly describe a given Peer-to-Peer system.

2.2 Abstract functionality

In a Peer-to-Peer system, all entities are peers. In a network, however, there are several roles that differentiate the entities. One example is the distinction end-system and router. End-systems want to communicate with each other. The routers inbetween help to connect the end-systems. Another distinction is based on the usage. A client contacts a server for some purpose. The effect is a communication session between client and server. The client role has two aspects, it is the initiator and it requests a service. The server role is to be accessible and to provide a service.

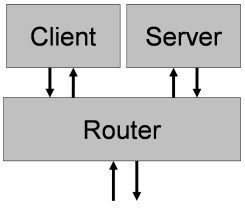


Figure 2.1: Peer as router, client, and server.

A peer combines all these roles in one entity (Figure 2.1). So, basically there are three abstract components in a peer. 1. client, 2. server, and 3. router. A Peer-to-Peer architecture needs to address all three of them.

One aspect of this finding is that not all of these roles take place on the same layer. In related work the Peer-to-Peer Common API extracts the logic of the structured Peer-to-Peer networks into one layer called Key-Based Routing (KBR). Other functionality like the hash table functionality of a Distributed Hash Table (DHT) is provided by the

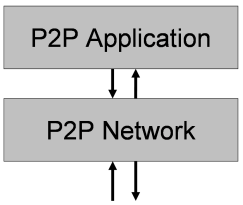


Figure 2.2: P2P Network Layer and P2P Application Layer.

DHT layer on top of the Key-Based Routing layer. For other Peer-to-Peer applications similar examples can be found. Thus, it seems to be reasonable to introduce a Peer-to-Peer network layer that provides connectivity in a way needed by the Peer-to-Peer system. Other layers on top provide distributed services to the application or are the application itself (Figure 2.2).

Peer as client: The client initiates requests in the network. The major functionality of the client part is provided by the application and is application-specific.

Peer as server: The server replies to requests in the network and it provides services to client entities. It will also interact with other server entities in order to cooperatively provide a service. While most server functionality is application-specific and part of the application, the server is covered to some extend also in the router (= Peer-to-Peer network) software. This is due to its cooperation with other peers and because of reachability issues.

Peer as router: The routing functionality is the central element of the networking side of a Peer-to-Peer system. The router needs to find paths to other entities in the network and to process queries towards one or many targets. The targets may be nodes, data or abstract elements that are addressed via an identifier (address, name, search string, etc.). The basic role of the router component is to enable all communications that the client and server components need.

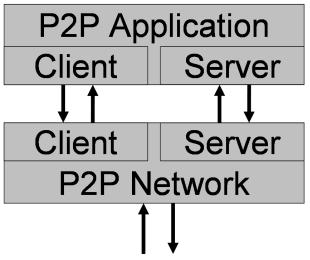


Figure 2.3: Client and server within the layers of a Peer-to-Peer system.

Finally, we combine the layered Peer-to-Peer structure containing Peer-to-Peer system and application with our components router, client, and server. The router is an essential part of the Peer-to-Peer system. Client and server are, to some degree, part of the Peer-to-Peer system component. The larger part of client and server are implemented in the application. Figure 2.3 illustrates the structure.

2.3 Components

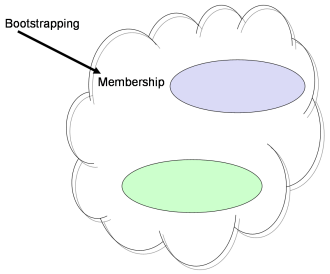
In this section, we identify components that can or need to be a part of a peer in a Peer-to-Peer system. The components are related to tasks that have to be solved by a peer,

and they contain corresponding solutions. Not all Peer-to-Peer systems have to solve all potential problems. This depends on their use-case. Ít may also be the case that they just solve a subset of these issues and that they can be used within a real system as a solution for this subset.

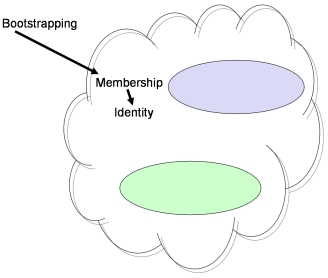
Among the tasks a Peer-to-Peer system has to solve are the bootstrapping (how to get to the system), the management of the membership, the routing, the query handling, the identification and addressing, for more advanced systems also roles - their assignment and execution.

2.3.1 Illustrating a Peer-to-Peer system

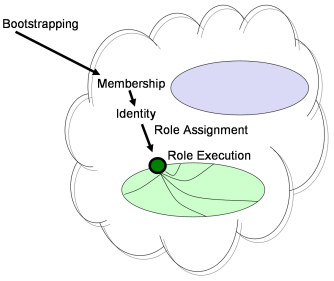
In the following we motivate many of the components we introduce in this section. The usage of a Peer-to-Peer system is shown step-by-step. Each step corresponds to a component, which we will discuss later.



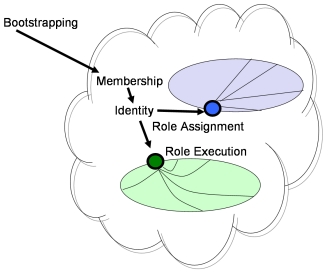
(a) A new node first needs to find the P2P network and then request membership.



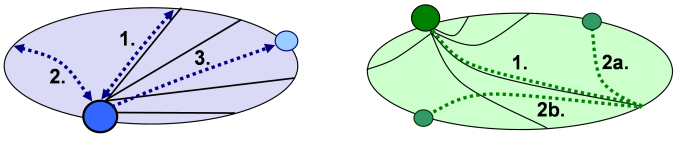
(b) A peer is assigned an identity or proves an identity. The identity is used to address the peer.



(c) After identification a peer is assigned a role and executes the role.



(d) A node may have several roles, say the role 'super peer' for network improvement and the role 'peer' as client.



(e) There may be different strategies on structuring the network (black lines). There are also various strategies for processing queries (dotted lines). In the left network queries are processed iteratively (1)(2), until the target is found (3). In the right network, the queries are sent to a neighbor node (1), which recursively processes the message and, as another strategic element, splits it into two concurrent requests to two of its neighbors (2a)(2b).

Figure 2.4: An example for steps of a peer from bootstrapping to participation.

In Figure 2.4 we give an example for components that a node needs to come from bootstrapping to participation in the Peer-to-Peer network. In the figure we have

1. **Bootstrapping** How to find to the Peer-to-Peer system.

2. **Membership** Define who is a member of the system and who is not.
3. **Identity** Name and address for a peer, later-on maybe specific ones for each role.
4. **Role Assignment** Some networks contain multiple roles, e.g. super peers, servers for subtasks, etc. There are various ways to assign a role.
5. **Role Execution** Some networks contain multiple roles, e.g. super peers, servers for subtasks, etc. The tasks and effects of a role need to be defined.
6. **Structure** Potentially each role may form a network according to its needs and tasks. This determines the structure of the network.
7. **Strategy on query** Query processing can differ in many aspects, e.g. iterative recursive, concurrent, etc.

This list is just an examaple and not complete. Other components will also be discussed in the sections that follow. They are strategy on maintenance, query types, node and link attributes, cooperation strategy, transport strategy, connectivity strategy, and security.

2.3.2 Components as Notation or System Description

A notation to describe or compare systems has to abstract over some functional aspects, but also to show enough details to express different ideas. To be more precise, a model of a system or method is built and described in a standardized form.

The notation should be suitable for all kinds of Peer-to-Peer systems and extend the modeling approaches discussed as related work in Section 1.2.1. Most important, it should be human-readable, not for anyone, but for anyone with enough knowledge in networking.

It should also support a range of different levels of detail. On the one hand, to present a method or an algorithm, one does not need details for aspects that do not matter. On the other hand, a concrete system needs to be specified with all its aspects. However, depending on the purpose of the model, each component may not be described in full detail.

The easiest way to present the components and their solutions is to list them in a list or table. We propose to only list specific components for the system or method. The rest should be left out and according to the context set to a default value or to any possible value.

Each component will be described with short key words, usually a single one describing an elementary enough solution. Some components will need a more complex description, e.g. the (routing) structure.

Here in our work, human visualization and presentation is the focus. The syntax is therefore lax as this helps to adapt to human needs. However, this could be replaced with formal representations for each description (with syntax and semantics). XML could be used as inter-exchangeable format. The result would then be more machine-readable and could be used for automatic methods in system selection, evaluation, optimization, and development.

Some notational issues:

- **'external'** = solved externally.
- **'n/a'** = not available in or does not apply for system

- **‘<connection> <rule>’** where <connection> describes what entities communicate (e.g. Peer ↔ Super Peer) and a <rule> which ones connect (e.g. ‘randomly with p=0.2’, ‘$Peer_{ID} * 2 = Super\_Peer_{ID}$’).
- ‘→’ = unidirectional communication
- ‘↔’ = bidirectional communication

2.3.3 Bootstrapping

The Bootstrapping problem (see Figure 2.5) is the question how to contact and join a Peer-to-Peer system. The problem is to have the knowledge required for this task. An external node does not naturally have the knowledge which nodes are in the Peer-to-Peer network and what their locator sets are. Thus, it needs to either have the knowledge or externally acquire the knowledge.

In most client/server Internet services, this is solved by knowing the name of a server or service and to lookup the locator via DNS. The DNS resolution is used as service on application level. The application resolves the name to a current locator and establishes a connection to the given locator. DNS again is a server approach. If we consider pure Peer-to-Peer, how could we do this without a server?

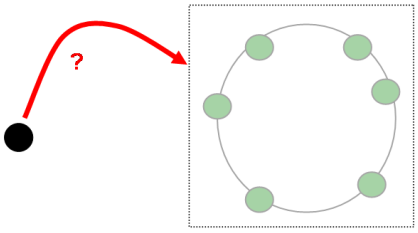


Figure 2.5: The Bootstrapping Problem.

In the following we discuss solutions to the bootstrapping problem.

Fixed set of Peers:
The software of a Peer-to-Peer system is distributed with a list containing a fixed set of peers that can be used for bootstrapping. This list may not necessarily be hardwired, but could also be configurable and updatable.
Figure 2.6 illustrates the bootstrap. A node from the list is contacted and, thus, the Peer-to-Peer network is found. In case of failure, the next node on the list is taken. The assumption is that this fixed set of nodes is always on and accessible. The advantages of this solution are that it is efficient and easy to set up. It is well-suited for the initial contact to a Peer-to-Peer network. The disadvantage is that the assumption needs to hold. The list also represents a single point of failure as the number of nodes is limited and they could be a target of an attack based on the list.

Peer cache:
A node maintains a cache with nodes it has recently seen on previous sessions. A cache usually has a limited size. There are mechanisms to add, replace, and remove peers. Candidates need to be found. Skype, for instance, only caches super peers. Kademlia maintains its buckets in an interesting way to preserve long-lived nodes. It only replaces an old node with a new node when the old node does not respond.
Figure 2.7 illustrates the Peer cache method. The cache is filled with nodes, maintained

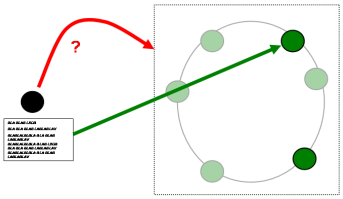


Figure 2.6: Bootstrap via a fixed list of rendezvous peers.

during runtime, and stored for future use. A future Peer-to-Peer session is bootstrapped by contacting a subset of nodes found in the Peer cache. Some of them may not be active. Once an active peer is found, the bootstrapping succeeded. The assumption of this approach is that there are long-lived nodes that will be accessible with the same locator in the future. Since this may only be true for few nodes, there also needs to be a strategy to select primarily such nodes for the Peer cache. An advantage of a Peer cache is that unlike fixed lists it automatically adapts to changes in the network. Storing a lot of nodes with their locator set also helps to increase robustness and, thus, some systems may need to maintain and store such a list anyway. A disadvantage is that entries in node caches may age rather quickly compared to the potentially extremely long time between two sessions. It is also not suitable for the initial contact to a Peer-to-Peer network.

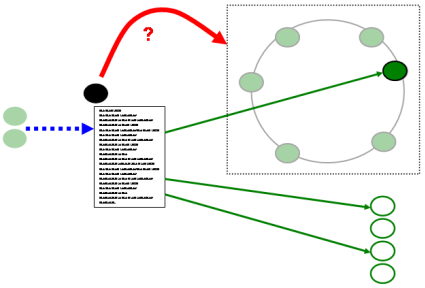


Figure 2.7: Bootstrap via a node cache containing nodes seen on previous sessions.

Bootstrap via external server:
A node can ask a server to help it join a Peer-to-Peer system. The server will provide the node with a list of peers in the Peer-to-Peer system. In many cases the server could be simply a webserver. The server also needs a way to obtain the information about currently active nodes in the Peer-to-Peer system that can be used as rendezvous peers. Appropriate peers could report to the server or the server could scan the Peer-to-Peer system.
Figure 2.8 illustrates the use of the approach for the bootstrapping. A node contacts a server to request a set of nodes via which it can join to the Peer-to-Peer system. It then selects one node and joins the Peer-to-Peer system. The assumption of this approach is that such a server exists and that this server node is always available. One may also consider it as a special case of the fixed list with the servers as bootstrap peers. Of course, the servers perform other operations than the peers would perform. An advantage of such an approach is that it adapts to changes in the network. Another advantage is that the bootstrapping can usually be solved directly with one query. The approach can also be used for the initial contact to a Peer-to-Peer system. As the Peer-to-Peer software might be obtained from a server of the authors anyway, there is already a server that might be

used also for this task. A disadvantage is that this is a server approach which partially contradicts the idea of a Peer-to-Peer system. The server needs a way to determine active peers. Such a system may also provide a lot of information about the network to non-members. Finally, the server is a single point of failure[1].

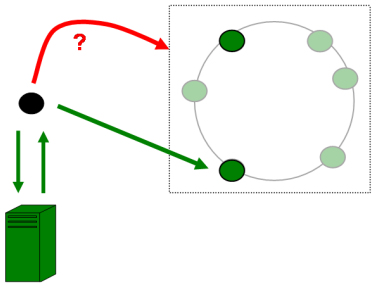


Figure 2.8: Bootstrap via a server that either knows current rendezvous peers or observes the network and provides a list of peers on request.

Bootstrap via external Peer-to-Peer system:
A node may also ask another Peer-to-Peer system for help to join another Peer-to-Peer system. In a simple setting this help may simply be a node list that was stored as a file in a file sharing or storage system. In a more advanced setting, the Peer-to-Peer system may also interact with the other Peer-to-Peer system to obtain information about nodes. Finally, the two Peer-to-Peer systems may be more closely coupled. One could be a subsystem spawned from the other. One can think of use-cases where the system the node wants to join could be a service network on a subset of nodes of the Peer-to-Peer system that is used for the bootstrapping. In such cases, the Peer-to-Peer systems are not completely separate systems.
Figure 2.9 illustrates the approach. A node contacts the Peer-to-Peer system and requests a list of nodes of the other Peer-to-Peer system that it wants to join. The Peer-to-Peer system processes the request and a node of the system will return the node list. The assumption of this approach is that such a Peer-to-Peer system exists and that the node can access the system. An advantage of such an approach is that it adapts to changes in the network. Another advantage is that the bootstrapping can be solved directly. The approach can also be used for the initial contact to a Peer-to-Peer system. A disadvantage is that such a bootstrap Peer-to-Peer system has to exist and one may argue that it might be more susceptible to attacks than a server solution and that fake bootstrap information may be obtained. Also, the Peer-to-Peer system needs a way to determine active peers of the other system. Such a system may also provide a lot of information about the network to non-members. Finally, the question may remain, how to bootstrap to this bootstrap Peer-to-Peer system.

Bootstrap via multicast, broadcast, or search in (local) network:
Figure 2.10 illustrates the approach. The node uses a lower-layer network mechanism to inform other nodes of a Peer-to-Peer system about its desire to join the system. The mechanism would preferably be a multicast to reach peers in the local network or Internet. In ad-hoc scenarios or local network scenarios this would be a broadcast message to find local nodes that are already in the desired Peer-to-Peer network. A node may also scan its local network for peers of the Peer-to-Peer network. However, this might be considered

[1]The P2P anonymization project I2P [anon] suffered from such a failure early 2008 when the main contributor to the system development left the project. The domain and server were inaccessible, yet hardcoded in the software. Noone else had the passwords to administer the systems.

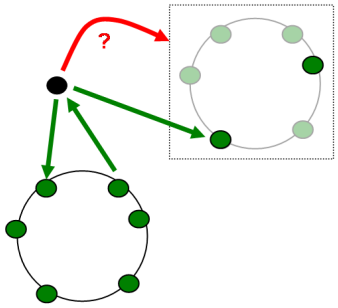


Figure 2.9: Bootstrap via another Peer-to-Peer service that either knows current rendezvous peers or interacts with the network and provides a list of peers on request.

as an attack by network administration and other users. The bootstrapping via multicast, broadcast or search in the local network has some special use-cases where it can be applied. The basic assumption is that in the local network or multicast group there are nodes of the Peer-to-Peer system.

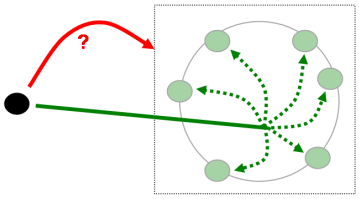


Figure 2.10: Bootstrap via multicast or broadcast mechanisms of the underlay.

Bootstrap via user input and human interaction:
The knowledge of the human user of a system may also be useful to bootstrap. The human user may, by other means, have acquired the knowledge where to connect to, e.g. via chat, email or direct communication. Figure 2.11 illustrates the approach. The system gets the locators from the human user and then tries to connect. This approach is no general solution to the problem and not suited for non-expert users. For debugging, expert users, and some special use-cases (e.g. darknets) the option to tell the software where to connect to is very useful.

Most Distributed Hash Table proposals are agnostic to the bootstrapping problem. They simply postulate the knowledge of an arbitrary node in the network. Pastry assumes that this node is a geographic close node. The assumption is that localization and social networks drive the join process of a Peer-to-Peer system. Skype uses a fixed list with bootstrap peers for the initial contact and Skype nodes use a cache of super peers which they will try first. Typical filesharing systems distribute 'server list' files via webpages. Many of them also use node caches, in particular if the approach is decentralized, e.g. on Kademlia basis.

Th bootstrapping problem is orthogonal to most other aspects of a Peer-to-Peer system and that many system ideas consider it as external problem. When building a real system, however, bootstrapping becomes important. A combination of different mechanisms is common and seems to be a good choice. The selection depends on the use-case.

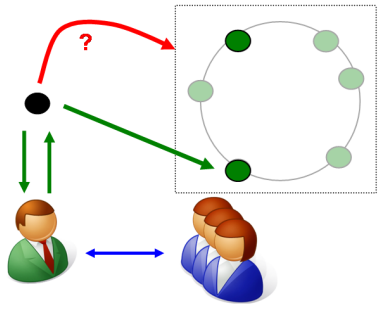


Figure 2.11: Bootstrap via human input and human interaction.

2.3.4 Membership management

In most current Peer-to-Peer systems membership is defined via connectivity to the network. Once a node is connected to one node and, thus, part of the network graph, it has successfully joined. At that moment the join process may not be over. Optimization and maintenance operations follow. These aspects are tasks of other components like structure, role, etc.

Membership itself is defined as a set. The set of all peers in the Peer-to-Peer network. Membership in general is not equivalent to connectivity and efficient routing, so we are not yet ready to call the peer a node in the graph $G = (V, E)$. It has become part of the cloud, not part of a structure. In a Peer-to-Peer system there needs to be a data structure that represents this set and, thus, the membership. On the basis of this data, nodes should be able to efficiently decide on the membership of other nodes and themselves. Usually, a set of links to a node and from a node determine its membership, say to a virtual predecessor and a virtual successor (common in key-based routing systems like most DHTs that are based on the idea of consistent hashing).

The membership management needs functions to process the basic membership operations. These operations are join, leave, and (potentially) access control. Membership management also needs a strategy to maintain the membership.

- **Join:** The new node contacts a rendezvous peer or member of the network and signals its request for membership. The network returns the acceptance and provides further information.
- **Leave:** A leave operation may be explicit or implicit if the node leaves without notification.
- **Maintain:** Check the status and maintain the corresponding data.
- **Access Control:** The membership component may also perform access control operations in combination with the security component (providing authentiation and authorization).

2.3.5 Identity

An entity usually has various identities. There are personal identities and pseudonyms of the user. The machine has addresses on ISO/OSI layers 2 and 3. The application on a machine is addressed via a layer 4 address, the port on transport layer in Internet terminology. The Peer-to-Peer system on top may reuse any of these identities. The cleanest solution is to create a system-specific identity for the Peer-to-Peer system.

Definition 2.1 (Identity) *An identity or ID of an entity is an abstract representation of the entity under which it is known by other entities. The ID is usually a number or a string. An ID is expected to be unique, while the entity may have more than one ID.*

Identities are important for addressing. The address of an entity is either equivalent to the identity or directly follows from the identity via a transformation[2].

Peers have an identity. They may have additional ones for each role they play. Furthermore, there may also be additional identities due to optimization (e.g. multiple virtual nodes for load balancing) or applications (e.g. user name, character name in a game). Besides there being identities for entities, there are also identities for items. An item is not a peer, but a data item or service that can be addressed in the network. In case of Distributed Hash Tables, this item addressing is used to implement the hash table functionality.

Identities may not be free from constraints. Systems may consider semantics for an identity. A simple example for such semantics is a hash table, where $ID = hash(\text{"}Perfect - video.mpeg\text{"})$ is the item ID assignment. Peer-to-Peer research sometimes calls this semantic routing when an identity can be derived from a semantic description. This could be the name of a video as above. This name is sufficient to locate the video (e.g. hash the name and perform lookup query). Since this is rather a one-way coupling of a name to an ID, other semantics may be used in parallel without disturbance, as long as there is a collision handling.

There are more restricting constraints, most important cryptographic identifiers[3]. Such identifiers combine an identity with a cryptographic key. The basic idea is that the identity is a public key $ID_A = PublicKey_A$. Since public keys may not fit to the desired identifier space, it is common to hash the key with a cryptographic hash function $ID_A = cryptographicHash(PublicKey_A)$. The advantage of such cryptographic identifiers is that the owner of an ID can always legitimate its ownership by proving its knowledge of the corresponding private key. These keys are the semantics of cryptographic IDs. Other semantics can not be included.

In most cases there is a limit for the bitlength of an identity. If this limit is small, handling collisions is of high importance. If this limit is large and collisions are extremely unlikely, one may skip the collision handling. However, collisions are not only a question of the size of the identifier space, but also a question of how identifiers are assigned. Only strategies like independent uniform random assignment guarantee that collisions are unlikely in large identifier spaces.

2.3.6 Roles and Role Execution

The basic idea of the Peer-to-Peer paradigm is that all nodes are equal. But as nodes provide different services, store different data, it is not necessary that all nodes perform equal operations. Hybrid Peer-to-Peer systems with super peers are common in practise. In super peer networks there are two roles, peer and super peer.

A peer may have one or more roles. It does not need be able to perform all possible roles. A role implies a certain service and operational strategy. Roles may differ in their

[2] In the context of the IP protocol, there is often the argument to split identity into identity and locator. There are many reasons for that. However, this differentiation is a cross-layer issue. Within one layer, there is one address – the identity. The locator is therefore a combination of one or multiple lower-layer identities (e.g. IP and Port). If multiple layers are used in a Peer-to-Peer system, one may need separate identities.

[3] Also called 'self-certifying IDs'

strategies for a variety of components. While in most cases a role (e.g. Peer, Super Peer) can be performed by many peers in parallel, in some cases a role may be assigned to a peer exclusively (e.g. Authority).

A peer that is executing role R is performing role-specific operations. Its responsibilities, data storage, and data processing may be different, as well the network structure, cooperation strategies, query processing. In Skype e.g. a peer is primarily a client, the super peers additionally perform routing, maintenance, and lookup operations.

Some examples for different roles:

- **Peer:** A standard entity in the network that performs the basic operations of the system.
- **Super Peer:** A super peer is a strong entity in the network. Typically, it optimizes the routing by increasing its own degree. Super peers may form a higher layer in a hierarchy that is strongly interconnected, while peers only connect to some super peers. Depending on the application of the system, super peers may also provide other additional services.
- **Authority:** An authority provides cryptographic material for security services and operates as trusted party in security protocols.
- **Peer of instance i:** There may be systems where there are separate instances. The role of such a peer is instance-specific as it is based on instance-specific data and connections.

2.3.7 Role Assignment

Prior to taking over a role. The role needs to be assigned to the peer. Depending on the purpose of the role, the assignment may be simple or complex. Simple forms are based on direct assignment at start up, by settings or randomly. Complex forms are based on experience and properties. An entity may self-assign its roles, another entity may decide on it, or other entities may elect the entity to a role.

In the following list, we present some examples for ways to assign roles.

- **Initialization / Preferences:** An initialization file may say that a peer has to perform certain roles. The peer will then participate in the network with these roles.
- **Proof / Authorization:** A peer proves that it is entitled to perform the role. On runtime it assigns itself to the role. Originally, an authority assigned the peer or owner of the peer to the role.
- **Lifetime:** The age of a node (= time since joining the system) may trigger the assignment of roles. In many networks, old nodes are considered to be robust and long-lived (Assumption: heavy-tailed lifetime distribution[4], e.g. Power Law) and are, thus, prefered for critical or important tasks within the system.

[4] A heavy-tail distribution is a distribution with a variation coefficient $c > 1$. This variation is extremely high, but this is surprisingly quite common for the distribution of lifetime and many other parameters in many real systems. In a lecture, we described this variation using a paradox. For heavy-tail distributions the following is true: "When we go to a bus stop and see that our bus has just left the bus stop, we are happy because most likely the next is to come soon after. On the other hand, the more we wait and still there is no bus, the less likely it is that a bus is coming, and many people waiting at the bus stop indicates that it will take long till the next bus is coming."

- **Strength:** Bandwidth, CPU power, free space, etc. may determine if a peer is suited to perform a certain role. Routing tasks usually need bandwidth, computational tasks may need more CPU power, some services may need space.

- **Identity:** The identity of a peer implies that it performs a certain role.

- **Trust:** Trust may be associated with a peer. According to this trust, it may perform a role or be selected to perform a role.

- **Randomly:** Random assignment is another option. A peer may randomly assign a role to itself or other network entities may do this. Random assignment may also bias the assignment according to the current need of peers with a certain role.

2.3.8 Structure (Structural and Routing Strategy)

The structure of a network may be randomly created or imposed by structural design principles. Mixtures of both options are common. A structure can be used to improve routing or to provide an application-specific information flow.

The structure component defines how peers built links to each other. Structure can be aware of roles and even be role-dependent, e.g. random graph of super peers and peers in star topology around a super peer.

In unstructured networks peers randomly connect to other peers. The identity of a peer does not influence the link selection. Pure random graphs provide a low diameter on average. In real networks social aspects or popularity add an imbalance to the network. The resulting Small-World and Power Law networks still show a low diameter.

In some networks, a long-lived structure is not necessary. Swarming does not maintain routing between peers, but peers connect with each other to perform an operation and terminate the connection when the operation is finished. Here, a peer either collects a lot of information about other peers or a central tracking component helps to find other peers.

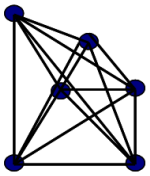
Routing Structure

The structure is one way to classify Peer-to-Peer networks. Usually, one differentiates between unstructured and structured networks. Within these classes one may then further differentiate.

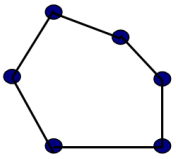
Unstructured networks primarily form themselves randomly. Structured networks establish and maintain a structure. The goal of a structure is to enable a deterministic lookup of other entities. In unstructured networks lookup of other entities usually involves searching. We will discuss this in more detail in the rest of this section.

Now, what is a structured network? Here are two simple examples for structures.

- **Cliques (Full mesh)**
  A clique is a graph where all nodes are directly connected. There is no need to lookup other nodes as all nodes are known to each node. Given n nodes in the network, each node maintains $n-1$ links and the overall number of links is $O(n^2)$. Maintenance is also expensive as each operation involves all nodes. This does not scale well. As a consequence, it is only suited for very small networks.



- **Circles**
  Nodes in a network have some kind of ID from some kind of ID space. In case of the circle we consider the ID to be one-dimensional and to be ordered. The nodes can then organize themselves in a circle according to their IDs. Each node only knows its predecessor and successor. The degree of the network is 2. In case we want to contact a certain entity we can simply find it via its ID, e.g. by forwarding it along the circle in one direction. However, we may need $O(n)$ hops to the node. Moreover, it is not very robust as an unclean leave of a node already destroys the structure and given two holes the structure can not be repaired.



An unstructured network does not organize in such predefined structures. Once a peer connected to another peer, it is in the network and a full member. In the following it will usually get to know other peers, either by randomly looking for other peers or simply by using the network. It connects to some of those peers. As a consequence, it could be anywhere in the network and any node could be connected to it[5].

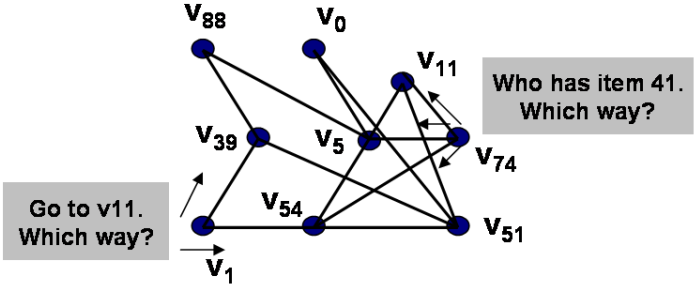


Figure 2.12: In unstructured networks, nodes cannot decide the right way towards a target. The item or node could be anywhere.

In unstructured networks, there is no organization of items and nodes. Such an organization could position items or nodes on particular points in the network. Now, to lookup a node, we have the following problem. As neighbors are random there is no way to know where a node is, but to search all over the network. This is also true for items as there is

[5]Unstructured networks focus more on locating items than on looking up peers, recent articles showed that with elegant replication of items the expected search time can be small while still keeping the number of replicas in certain bounds[TKLB07].

no information where an item is. Nodes usually store their items locally and maybe tell their neighbors what they have. Figure 2.12 illustrates this situation. For node v_1 there is no way to know if v_{39} or v_{54} is better to get to v_{11}. Also, v_{74} has no information to decide whether v_{11}, v_5, or v_{54} is closer to item 41.

One may consider the use of routing algorithms, e.g. based on distance vectors. With such algorithms the nodes learn where certain IDs are located via some information exchange. But even the application of such routing algorithms would fail with respect to complexity. IDs are flat labels. Given a random distribution a similar ID could be anywhere. As a consequence IDs cannot be aggregated and every node has to learn about every other node. This would result in extremely large routing tables and one still would have no clue where to locate nodes that are yet unknown.

From this discussion unstructured networks may sound to be a bad choice, but they also have their advantages and applications, in particular when the direct routing towards an ID is not a common operation and false negatives (node not found despite it was online) do not matter. A system based on fuzzy queries might be an example. An advantage is that there is no structure that has to be maintained whenever something changes. There is no need for a complex and expensive join and leave procedure.

- $Join_{unstructured}$: A join is completed once the node is registered at one other node (except for the need of this node to get to know more nodes...).
- $Leave_{unstructured}$: No need to rework the network, but to locally remove the link.

Now to structured networks, the main purpose of a designed structure for routing is to combine identifier and structure in order to cluster identifiers in certain regions of the graph. Given such a clustering, a node can easily decide in what direction an identifier of interest is located.

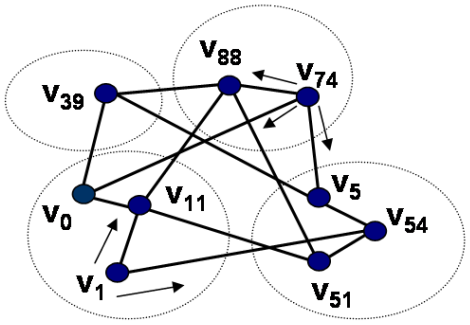


Figure 2.13: Nodes with similar IDs cluster in certain areas of the network.

Figure 2.13 shows the basic idea. Nodes with similar IDs are clustered together. Such a network organization needs a way to find nodes with similar IDs. That is what a new node has to do initially to find its cluster. It will preferably connect to nodes with similar IDs from its cluster. It will furthermore create some links to nodes with different IDs. v_1 has now no problem to find v_{11} as it is a direct neighbor. For v_{51} it can go via v_{54}. v_{74} needs to find a node close to item 41 again. This is probably v_{39}. As a next hop, it may prefer to go via v_5. However, such a procedure may also fail. In order to avoid loops and dead ends such networks would need to create routing tables with a routing algorithm. With this clustering many nodes can be aggregated and, thus, the approach is feasable and scales. Another option is to do the routing by design and use stricter structures.

So, the clustering of nodes on the basis of some structural strategy leads to a direct routing towards a target, either by design or the possibility to efficiently apply routing algorithms. Most current structures for key-based routing are designed to route efficiently and achieve a diameter similar to Small-World or random graphs. Key-based routing routes requests towards an identifier ('key') and the target is the node which is either best or good enough according to a given metric for the distance of target identifier and node identifier. One may introduce the term identifier-based routing. In that case, one does not allow a delivery to a close node, but insists that identifier and node identifier match exactly and returns an error otherwise.

Besides basic goals like being able to directly forward a message to a target, there are also secondary goals. Robustness is usually achieved by adding more links than necessary. Overhead may be reduced by either using less links, or by reusing routing tables of other nodes. Quality-of-service can be improved by optimizing the structure for important requests.

For basic connectivity neighbor (e.g. predecessor and successor) links are established, for efficient routing there are short-cut links, and for robustness there are additional neighbor links. Some structural templates:

- **Ring:** The nodes are considered to be ordered in a ring, each node being connected to predecessor and successor. As short-cuts there are additional links to nodes in roughly exponentially increasing distance $(2^i, i = 0..ld(|identifyerspace|))$.
- **Tree:** In structured Peer-to-Peer systems, the tree is a virtual construct. The first layer splits between nodes with IDs starting with '0' and nodes with IDs starting with '1'. The subsequent layers split each subtree accordingly. For each layer l in the tree, a node needs to know a node with an identifier that shares a common prefix with it for the $l-1$ previous layers and differs on the l-th bit.
- **K-nary Tree:** Instead of a binary tree, build a k-nary tree and connect to nodes for each of the k-1 different prefixes on each layer.
- **Bucket:** Instead of one link for one direction, maintain or connect to a set of links into that direction.

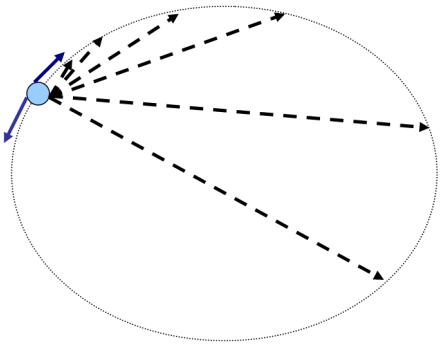


Figure 2.14: Ring-based structure like Chord

The basic complexity of state and lookup for such approaches is $O(log(n))$ with n being the number of nodes. The complexity of join and leave operations is also depending on the structure, in particular the number of links to be created and the amount of exploration

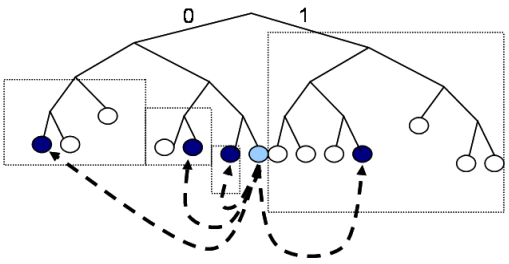


Figure 2.15: Tree-based structure like Pastry

that is necessary to find appropriate peers. Common examples are Chord [SMKK+01], Pastry [RoDr01], and Kademlia [MaMa02].

There are also othe graph structures like Butterfly graphs and DeBruijn graphs that achieve $O(1)$ state with still $O(logn)$ lookup. Usually, however, each link helps to reduce distance, so if O(1) means less links, the distances in the graph will be larger on average, especially as the difference between logarithmic and constant complexity needs large numbers to really make a difference. Butterfly graphs are used in the Viceroy (Viceroy [MaNR02]) DHT, but the o-notation hides its large factors that make it less efficent in practise. DeBruijn graphs are better in theory, but usually need assumptions not suitable for Peer-to-Peer networking like synchronized operations. Proposals to use DeBruijn graphs are D2B and Koorde [FrGa03, KaKa03].

Structures need to be created and maintained when the network changes. During the join operation a node acquires information for building the structure. This primarily means relevant other nodes. They can be found via lookup for particular identifiers, via routing information of other nodes ('use their knowledge for links into similar directions'). For maintenance a node has to check and update the routing entries from time to time using similar methods.

Application-specific structure

Systems may also form structures directly according to the needs of the desired service. A service may have a specific information flow, which can be transferred into a structure among the peers. The network, thus, reflects this flow and as a consequence, its use makes it easier for the application and its operation is more efficient.

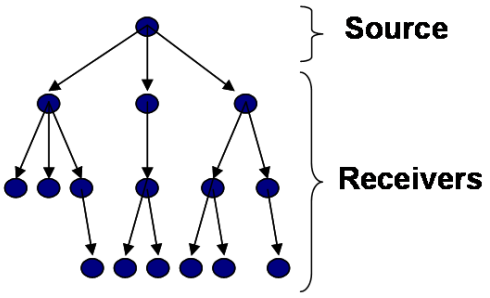


Figure 2.16: Multicast Distribution Tree as structure.

A common example for such structures is the distribution tree of a multicast application (see Figure 2.16). Another example could be an event service, where event observers send

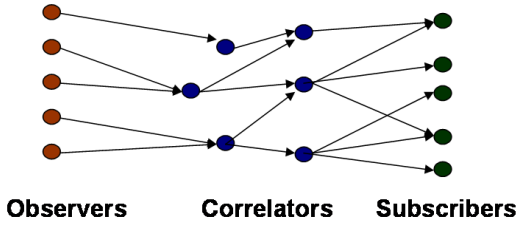


Figure 2.17: An event service as structure.

their observations to event correlators. The event correlators analyze the data, filter and aggregate it, and send out events to their corresponding subscribers if data matches their subscription (see Figure 2.17). A simple Peer-to-Peer VPN may create a full mesh between all nodes, a VPN with a VPN server will organize its peers in a star topology around the server.

There are many more possible examples. Collaborations may use special structures according to the joint operations. Some applications may need trusted peers or super peers that collect and process information of their peers. We proposed such an approach in our studies [RFNW+05, RWFN+08a] on Massively Multiplayer Online Roleplaying Games (MMORPGs).

One may generalize this. Every Peer-to-Peer application where there are different roles that contribute to the information flow for cooperative tasks should have a specific structure. This flow of information determines how the network is structured.

Notating a structure

This is a short section on how to notate a structure. Essentially, notate a structure simply in a way that fits to describe it easily. We usually use the following in our examples in a later section. Basically, we first say which roles connect to each other (Peer and Peer, Peer and Superpeer, etc.) and state the rule under which peers of each role interconnect.

Role Relation Role $\Longrightarrow$ *Rule*

An example:
Peer $\leftrightarrow$ *Peer* $\Longrightarrow$ for all i=0,1,... link to a Peer P with common prefixlength i
Superpeer $\leftrightarrow$ *Superpeer* $\Longrightarrow$ link to successor in ID space
Peer $\leftrightarrow$ *Superpeer* $\Longrightarrow$ randomly

Unidirectional connections are indicated with unidirectional arrows, bidirectional connection with bidirectional arrows.

2.3.9 Metrics

Most systems need an understanding of distance, distance between identifiers, distance between items, distance between node and item, distance between nodes. A distance metric is a function that outputs a distance value on the input of two identifiers. Mathematically a metric is defined through the following properties:

Definition 2.2 (Metric) *Let X be an arbitrary set. A function $d\colon X \times X \to \mathbb{R}$ is called a metric if for arbitrary elements x,y, and z in X the following properties hold:*

1. $d(x,x) = 0$ *(identical points have distance 0)*

2. $d(x,y) = 0 \Rightarrow x = y$ *(non-identical points have distance greater 0)*

3. $d(x,y) = d(y,x)$ *(symmetry)*

4. $d(x,y) \leq d(x,z) + d(z,y)$ *(triangle inequality)*

In a Peer-to-Peer system, metrics are important for some elementary operations. They are needed for the placement of items, and for the routing. Some systems may not need them if they do not have to deal with distances of identifiers. In case of routing, one necessary metric is a distance metric to determine distances in the identifier space. Another metric could be based on distances in the underlay. In that case the properties of a mathematical metric may not be satisfied (primarily the triangle inequality). However, in many cases when such measurement-based values are used, they are used locally without a global view of the system. In such cases there is no strict need for a real metric, as they simply provide information about something being lower or higher and therefore preferable over the other or not.

The standard metric for numbers and vectors is the euclidian metric. In multidimensional spaces there are many other metrics that are frequently used besides the euclidian one. For the Peer-to-Peer networks with their usually one-dimensional identifiers, most metrics including the euclidian metric end up being equivalent to the absolute value of the difference of the identifiers.

Definition 2.3 (Euclidian Distance) *The euclidian distance is* $d(x,y) = \sqrt{(x-y)^2}$. *In one dimension the metric is equivalent to* $d(x,y) = |x-y|$.

Instead of computing the distance on the numeric value of an identifier, one can also consider the identifier a bitstring. The Hamming distance counts the number of bits that differ. Let $\oplus := XOR$.

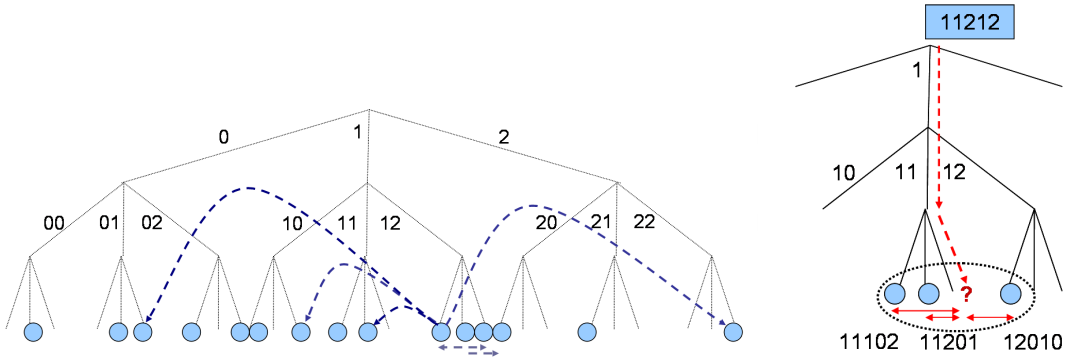
Definition 2.4 (Hamming Distance) *The Hamming Distance is defined by* $d_{Hamming}(x,y) = \sum_i x_i \oplus y_i$

Identifiers are usually used like a number. As a consequence, single bits represent different values and thus they are of different significance. The XOR metric combines difference in bits and the numeric distance. The XOR metric achieves this by evaluating the numeric importance of a bit difference.

Definition 2.5 (XOR Metric) *The XOR Metric is defined by* $d_{XOR}(x,y) = \sum_i 2^i(x_i \oplus y_i)$

One can also base metrics on characters. With respect to the bit encoding, usually a fixed number of bits, say m, together represent one character. The number of different characters is then 2^m. The prefix metric counts the number of characters at the beginning of the two identifiers that match. It determines the length of their common prefix.

Definition 2.6 (Prefix Metric) *The Prefix Metric is defined by* $d_{prefix}(x,y) = length - of - prefix(x,y) = \min_{i,i\geq 1}\{(x_i <> y_i)\} - 1$



(a) An example for the routing table of a node in Pastry.

(b) Item Placement

Figure 2.18: Pastry combines prefix and euclidian metric.

Pastry[RoDr01] uses two metrics. It uses the prefix metric for its long-distance routing. However, the prefix metric cannot be used to distinguish between nodes with the same common prefix. So, for item placement and routing in the neighborhood of a node, Pastry uses the euclidian metric.

The routing table of a Pastry node is filled with one node from each subtree that has a prefix distance of 1. Among neighbors (leaf set in Pastry's routing table) there is no distance in prefix, but there is a numerical distance if their identifiers are interpreted as numbers. Figure 2.18 displays the use of the two metrics. While advancing in the tree is based on prefix-based routing (prefix metric), the location of items and local routing is performed using the euclidian metric. For the item 11212 there are two nodes that are close with a common prefix length of 2 and one additional node that is numerically close, but has a shorter common prefix with the item. As the example is to base 3, the distance of 11212 to 11201 is 4 (decimal) and the distance to 12010 it is 7 (decimal).

2.3.10 Maintenance

Peer-to-Peer systems are in a constant state of change during their operation. Nodes may join and leave all the time. As they are part of the network structure and provide services, Peer-to-Peer systems need to adapt to the changes and maintain their operability. It needs to preserve stored information and state, reassign services in the network as well as rework its structure.

There are two primary ways to deal with the lifetime of information.

- **Softstate** The softstate of information limits the lifetime and information without refreshment is removed after an expiration date.
- **Hardstate** The hardstate of information permanently keeps information unless it is changed.

A hardstate approach is simple and at first efficient. However, it needs a careful design to ensure that outdated information is removed. Otherwise, it will store more and more information and request more and more space like a memory leak. A softstate approach usually requires more messages as information needs to be refreshed even without any change. Its advantage is that outdated information will automatically be removed and that the systems resets itself after some time.

Structural maintenance needs to deal with join and leave operations. It should not expect all leave operations to be announced. Sudden failures will cause unexpected leaves. There are a variety of strategies to resist failures and to reduce the complexity of leaves.

- **Replication** Information is stored on k nodes (replicas) instead of only one. After a failure it is restored from the remaining replicas. Besides information, services can be replicated accordingly.
- **Buckets** Buckets are an idea similar to replication, but with respect to routing. Instead of maintaining one connection for each direction, a node additionally stores $k-1$ (bucket size k) additional nodes for each direction in its routing table, so that they can be used as backup or alternative for the connection into that direction.

Change detection needs to check information from time to time if it is still valid. In case of routing tables and neighbors, this is essentially a ping operation that is performed from time to time. A peer may also inform other peers when it is causes a change (e.g. by changing to another IP address). This may happen beforehand or afterwards.

- **Check Regularily** Nodes regularly check their neighbors (still there?) and structure (still correct/still best link for that direction?). This approach may vary in the interval between maintenance operations, and how much is checked each time. Usually, only some neighbors and routing entries are checked at a time.
- **Check On Access** A node checks parts of the structure when it uses a link in that direction or interacts with other nodes from that direction.
- **Communicate self-triggered change** In a case when a node changes in a way that maintenance has to be done, it signals this to all relevant nodes. This may be a leave operation (goal: graceful leave), a change of its locator, etc. The other nodes are informed and take appropriate actions.
- **Mixed Strategy** Combining the different strategies, e.g. regular check among neighbors in ID space, on access check and optimization for long-distance links, announce changes a short time before they apply when possible.

The following actions may be triggered by changes in the network.

- Perform repair on structure.
- Update locators on change.
- Copy and/or restore information.

There is a tradeoff between exact knowledge and redundant knowledge in dealing with maintenance and faults. One strategy is to detect changes and adapt immediately. In that case regular checks are performed at a high rate and once a fault is detected, alternatives (alternative nodes in case of node failure and routing) are looked up in the network. The opposite strategy is to know a lot of redundant information. One primarily checks on access and maybe at a very low rate regularily. If something fails, the redundant information is used for processing the request and update operations ensure that soon after enough redundant information is available.

One can compare the two strategies to the strategies of Chord and Kademlia. Chord's stabalization performs regular checks on neighbors and a random sample of the finger

table (routing table in Chord). Kademlia uses buckets and fills them with nodes from the corresponding subtrees. It checks routing entries when it sees other nodes that could be added. In Kademlia there is then the strategy to replace the least recently seen node in the bucket with the new candidate node only if the old one does not reply.

2.3.11 Queries

A Peer-to-Peer system may support different kinds of queries. A query may be a message from A to B, but also a complex search for an entity with specific properties.

- **Exact Query** An exact query directly targets one entity or item with the corresponding identifier.
- **Range Query** A range query targets all nodes where a property, e.g. the identifier, is within a certain range.
- **Fuzzy Query** A fuzzy query asks for one node or item where a property, e.g. the identifier, is close to the given value according to a metric.
- **Complex Query** Complex queries may allow to search mask, data-base like, etc.
- **Anycast Query** An anycast query asks for one node with a given property.
- **Multicast Message** A multicast query asks for all nodes within a set of nodes called multicast group. Usually, multicast implements a group membership protocol.
- **Broadcast Message** A broadcast query addresses all nodes in a system.

Structured networks like DHTs are optimized for exact queries. Range queries and fuzzy queries hardly profit from the structure that is maintained. This is one reason why unstructured networks dominated and still dominate in filesharing where users look for files close to a given search string. Anycast, multicast, and broadcast can profit from an arbitrary routing structure. However, they often use a structure of their own. Anycast is quite common in Peer-to-Peer systems with replication of knowledge and services, e.g. find one of the peers operating similar to a tracker for a torrent in trackerless BitTorrent.

2.3.12 Query Strategy

There are various strategies how queries are processed. A fundamental distinction is the recursive or iterative query processing. In recursive processing a query is passed from node to node till it is solved. In iterative processing the originator interactively passes the query to other nodes and they report to the initiator. Recursive processing is usually more efficient as less messages need to be sent and permanent problems with NAT are avoided. Iterative processing has the advantage that the initiator has more control over the process, which can be useful for security.

Concurrent query processing sends a query to multiple (say α) nodes instead of one node. Concurrent and iterative means that the query is passed to α nodes on each iteration. Concurrent and recursive means that the query is passed α nodes, which each perform a non-concurrent recursive query processing.

The next issue is how to determine the target or the next hops. The decision is usually based on an identifier. This identifier either represents an entity, an item, a task, or a group. The so-called semantic routing of some Peer-to-Peer networks embeds semantic properties into this identifier.

A common routing strategy is the greedy routing. The next hop is selected as the closest to the target according to a metric. This is common in structured systems where identifiers cluster in certain region and greedy routing, thus, directly leads to a path towards the target. Greedy routing is a good option in all cases where optimized routing tables exist, either by design (structured Peer-to-Peer) or by incorporating routing algorithms. An important requirement for all these systems is that there is a scalable way to build a routing table, i.e. to combine a set of nodes into one routing table entry.

Another option is to perform a random search or variants of limited flooding. This is common when there is no way to know exactly which nodes to ask. This is common in unstructured networks or for complex queries where the identifier is not sufficient to determine a target.

A query may be split into multiple parallel queries. This may be rooted in the query type (e.g. at inner node in multicast tree) or for other reasons as performance (e.g. ask more nodes per roundtrip in a random search) or robustness (e.g. concurrently ask server and peers in CoSIP[FNKC07]).

In case a system allows different types of queries, the strategies for different queries may differ. Different roles and structures with and inbetween them may also cause mixed or multiple query processing strategies.

Incorrect query results are another issue. This is also affected by the structure of the overlay (Section 2.3.8). A request may not find the item or node of interest. A *false negative* describes the case when a request does not find its target, despite the target node existing in the network. Unstructure networks may not flood a request to all nodes. Thus, they may produce false negatives by design. Structured networks may produce them due to inefficiencies or maintenance problems (e.g. caused by churn). A *false positive* would be the case when the request confirms the existence of the target, despite it not being there. This could happen when the target node leaves before providing the service.

2.3.13 Node Attributes

A node may have attributes that are important to describe or understand the system. We usually consider a node to have the attribute ID with ID_{node} as its value. Until now we implicitly assumed that there is such an ID. Now, we make it explicit. Other implicit attributes of a node would be IP-address, ports, socket, ...

The type of an attribute is often implicitly clear. It could be an integer or real number. it could be a string. It could be a set or list.

Attributes may not only be per entity, but also per role. We note them as 'role.attribute'. If the attribute is defined as abstract as this, we consider each role to have an individual attribute like this, e.g. 'role.ID'. We replace the general term 'role' with a particular role to define role-specific attributes, e.g. 'superPeer.ID'.

While some attributes are not under control of the Peer-to-Peer system (e.g. IP address), other attributes are and need to be assigned. The assignment can be noted as in typical programming languages using 'attribute = value'. 'hash()' and 'random' are typical functions needed in Peer-to-Peer systems in this context. Here again, 'hash()' and 'random' represent general function types and can be replaces with more concrete functions like '$sha-1()$' or '$uniform([0, 2^{40}-1])$'. e.g. 'superPeer.ID = hash(IP:Port)', 'ID = random'.

There are some typical node attributes. This list is not complete and most of them are implicit attributes that we would not list when describing a system.

- **Identity:** Identity of the node in the system. In most cases a long numeric value.

- **Abstract / user identities:** Identity of user or identity in application. It can be a numeric value, but string is more common.
- **IP:** IP address
- **Port:** local port
- **Bandwidth:** either the bandwidth of the Internet connection of the node, or the bandwidth the node is willing to commit to the Peer-to-Peer network.
- **Trust:** Nodes may rate other nodes and assign them a trust or reputation value. In centralized cases, the trust rating of a node is usually global. In decentralized cases, the value is local and each node may have a different rating for the corresponding node.

The node attributes field should not be used for state variables. The routing table is not a node attribute.

2.3.14 Link Attributes

For some networks that optimize distances or service quality it necessary to assign attributes to a link. The field link attribute is used to define such link attributes. As with node attributes, link attributes should only be used when they are used by other components to perform some system-specific operation.

Next we list typical link attributes. The list is not complete. Link attributes are in most cases quality-of-service-related.

- **Distance:** Latency between the endpoints of a link.
- **Bandwidth:** Either the bandwidth of the internet connection of the node, or the bandwidth the node is willing to commit to the Peer-to-Peer network.

2.3.15 Cooperation Strategy

One motivation for the advent of Peer-to-Peer systems is the aspect of cooperation between end-systems and users. Peer-to-Peer systems may be based on different strategies for cooperation. Peers may provide resources for communication (bandwidth), processing, and storage.

In Key-Based Routing systems the peers provide a small amount of bandwidth and they process lookup requests towards a key. For the hash table functionality in a DHT, they provide storage for the pointers to the real locations of a key.

Strategy for cooperation becomes more important when a) the cooperation of peers should contribute significantly to the realization of the service and to its performance, b) the peers need to be motivated to provide enough resources. Swarming is an example for case a), incentives are a method for b).

In Swarming Peer-to-Peer systems peers with a common interest cooperate to fulfill their desired service, e.g. download the same large file or streaming the same video. The idea is taken from biology. Peers (e.g. bees in biology) perform simple and local operations, which due to the overall swarm result in the service. The peers profit from their cooperation as the swarming may significantly improve performance and resilience in comparison to a server. BitTorrent [Cohe03] is a well-known example for the concept of swarming.

An important idea for achieving a cooperation of the peers is to include incentives for cooperation. This incentive may be the overall improved service. However, a single peer may profit even more if it does not cooperate and simply freerides. The peer uses the system without providing the service. Thus, one may need to motivate the peers to cooperate and contribute. Incentive mechanisms like Tit-for-Tat or reputations system can be used to prefer peers that cooperate over peers that do not or do less. Tit-for-Tat means that if a peer cooperates with another peer, the other peer will give the peer a better service, e.g. allow a higher bandwidth for downloading chunks of a file. It is clear that incentives can be realized in swarming as each peer can serve each peer. For many other Peer-to-Peer systems and applications, a peer may never serve peers that provide a service it requests. There, other means of determining the cooperation of a peer are necessary to realize incentives. As a consequence, most non-swarming systems do not implement incentives.

2.3.16 Transport Strategy

There may be various ways to transport data from A to B. In many cases a Peer-to-Peer network solely serves the purpose of a lookup service and no data transport from A to B is considered. In such cases, once A found B it will directly connect to the current locator of B.

In case data transport and distribution are considered, this can be done in different ways. Data can be transfered via bulk transfer. It can be split into many parts. In case of multiple recipients, the parts may be distributed not only from the source, but also between receivers.

Instead of sending data directly, data may also be stored in the network. The intermediate nodes will either forward the data later-on to the receiver(s), or the receivers will query for the data once they are interested. One could call the latter ‘Store-and-Query’ transfer. While this discussion may sound academic for standard communication between A and B, it fits well to storage and anonymity systems. A similar idea can be found in a current EU project on future Internet [TaTS08], which discusses to replace current forwarding with the public-subscribe principle.

Basically, there are the following options:

- Direct connection after first contact.
- End-to-end communication over overlay network.
- Parallel connections via overlay network.
- Storage of data/data chunks in the network for subsequent retrieval.

2.3.17 Connectivity Strategy

Peers may form the network graph in many ways. This component deals with the setup and the type of a link between two nodes.

There are a variety of options for connection settings:

- TCP connection
- TCP connection on request
- UDP with keep-alive

- UDP
- ...

There are also technical problems for connectivity. The main problems are caused by middleboxes like Network-Address-Translators (NAT) and firewalls. NAT and firewalls reduce the reachability of a node. Ports, applications, and external connection establishment may be blocked by a firewall. If a node is connected to the Internet via NAT, its global IP address is different from its local IP address. Some protocols may have problems with different locator sets when they use IP addresses unaware of the existence of a NAT. Even more of a problem is that a node behind a NAT cannot be addressed from outside, at least not before it either configured the NAT (bound a port to its local locator) or connected to another node (this 'hole' can not always be used for other connections). As a large fraction of machines on the Internet are behind NAT, connectivity strategies are a fundamental part of any real system. To some degree, they are orthogonal to most other ideas of a Peer-to-Peer system. However, connection establishment gets more complicated and systems where peers continuously contact a lot of different nodes will suffer.

There are methods to deal with problems caused by middleboxes.

- **Middlebox Configuration** In home networks middleboxes like NAT or firewall are usually configurable. UPnP is a protocol for middlebox configuration and common in home devices.
- **Usage of Standard Ports** In many cases overlays listen on special ports to avoid conflicts with other applications. Listening on standard ports helps to traverse firewalls.
- **Relay Nodes** Another node may serve as relay. A node communicates to the rest of the overlay via a relay node. The relay node simply forwards the messages to the corresponding directions. This is interesting for traversing NATs, firewalls and to provide access for small devices. An IETF standard for relaying is TURN.
- **Hole Punching** There are also techniques to make a NAT open a port and use it for other connections. These methods are called hole punching. There are also IETF standards for hole punching, e.g. STUN and ICE.

2.3.18 Security

Security is not a primary objective with respect to the design of common Peer-to-Peer systems. Security is out-of-scope and orthogonal to most of the system design. Yet, many real-world systems with low security requirements implement at least subtle security measures. Systems with higher security requirements usually add a central authority to the system to achieve the desired security-goals.

Security needs some kind of **security anchor** onto which secure operations base their decisions. This is usually an authority, maybe the initiator of the Peer-to-Peer system. The authority is represented by keying material, either keying relations with all other peers or a public/private key pair with a known public key. As such a central authority is contrary to the idea of a Peer-to-Peer system, many systems avoid such an authority by either considering only low security or by performing distributed web-of-trust approaches.

To achieve some of the security goals the Peer-to-Peer system needs to apply cryptography or include network security protocols to achieve them. The basic problem again is how to bind that to the security anchors of the system or, at least, the communication partners.

A typical problem in the Peer-to-Peer case is the initial contact problem where knowledge and trust has to be established and is yet not existing.

Some networks may want to apply Access Control and only allow access for a defined set of nodes. The prerequisite for this is Authentication and Authorization.

While other security goals may be obsolete, Peer-to-Peer systems need a strategy for dealing with P2P-related attacks like the Sybil attack, the Eclipse attack, poisoning attacks, etc.

We will deal with security in Peer-to-Peer systems in later chapters. Chapter 4 presents and categorizes security issues of Peer-to-Peer systems.

2.4 Applied on real systems

Here, we show that current systems can be divided into these components and, that we can briefly describe the essentials of the systems.

2.4.1 Skype

The Skype software and Peer-to-Peer system is well-known due to its revolution in bringing a Voice-over-IP to the masses. It is a closed system with no open specification. However, there has been some research to reveil Skype's operation, e.g. [BaSc06, BaSc04, BiFa06].

System	**Skype**
Bootstrap	Fixed set plus cache
Identity	Skype name (string)
Roles	Peer P, Super-Peer SP, Authority A
Role Assignment	Peer $\Longrightarrow$ any Super-Peer $\Longrightarrow$ Bandwidth and other criteria Authority $\Longrightarrow$ fixed
Membership	start as role Peer
Structure Strategy	P$\leftrightarrow$P $\Longrightarrow$ none P$\leftrightarrow$SP $\Longrightarrow$ P connected to a SP P$\leftrightarrow$A $\Longrightarrow$ none SP$\leftrightarrow$SP $\Longrightarrow$ randomly, strongly meshed (Details not completely public) SP$\leftrightarrow$A $\Longrightarrow$ none A$\leftrightarrow$A $\Longrightarrow$ none
Query	exact
Query Strategy	repeated random search over a number of random SPs
Security	- Authority as Security Anchor - Authentication and Access Control via A - Keys in software to authenticate software - Confidentiality via Encryption

2.4.2 Chord

In the academic world, Chord [SMKK+01] is probably the most well-known key-based routing system.

System	**Chord**
Bootstrap	external
Identity	ID = SHA(IP) or random
Roles	Peer
Role Assignment	Peer $\Longrightarrow$ any
Membership	- start as Peer - defined over network
Structure	$P \leftrightarrow P \Longrightarrow$ link to $P + 2^i$
Maintenance Strategy	regularily (every x seconds) ping and stabilize (check routing entries)
Query	Exact
Query Strategy	Greedy, recursive
Security	no Authentication and Authorization

2.4.3 Pastry

Pastry [RoDr01] is a tree-based key-based routing system. It uses k-nary trees. It employs locality optimization and tries to reduce maintenance overhead.

System	**Pastry**
Bootstrap	external
Roles	Peer
Role Assignment	Peer $\Longrightarrow$ any
Identity	random ID
Membership	- start as Peer - defined over network - assumes bootstrap via geographically local peers - use the routing tables of nodes contacted during the lookup of own identity to create own routing table
Structure	$P \leftrightarrow P \Longrightarrow$ link to a P in each subtree with common prefix length i $P \leftrightarrow P \Longrightarrow$ link to P with k closest preceeding and succeeding IDs $P \leftrightarrow P \Longrightarrow$ link to random nodes with low RTT found via join/routing
Distance metric	Prefix length, Euclidian for local decisions
Query	Exact
Query Strategie	Greedy, recursive
Link-Attribute	RTT
Security	no Authentication and Authorization

2.4.4 Kademlia

Kademlia [MaMa02] is based on the XOR metric. The XOR metric unifies the prefix and euclidian metric that are both used in Pastry into one that fits to routing and item positioning. Kademlia uses buckets instead of one link into one direction to improve resilience.

System	**Kademlia**
Bootstrap	external
Identity	random ID
Roles	Peer
Role Assignment	Peer $\Longrightarrow$ any
Membership	- start as Peer - defined over network
Structure	$P \leftrightarrow P \Longrightarrow$ link to k nodes N_k with $P + 2^i < XOR(P, Nk) < P + 2^{i+1}$ = k-bucket
Maintenance Strategy	-check least recently seen node in bucket when seeing new node for buckets - replace if old node is not responding.
Query	Exact
Query Strategy	Greedy, iterative, concurrent
Security	no Authentication and Authorization

2.4.5 BitTorrent

BitTorrent[Cohe, Cohe03] is a Peer-to-Peer system for efficient file transfer to many clients requesting the file. First, we list the tracker-based BitTorrent.

System	**BitTorrent**
Bootstrap	.torrent file
Roles	Peer, Tracker
Role Assignment	Peer $\Longrightarrow$ any Tracker $\Longrightarrow$ fixed
Identity	random ID
Membership	register at tracker
Query	Tracker returns list of nodes, Peers connect for chunk exchange
Cooperation Strategy	- Peer exchange data chunks - TitforTat and variants as incentive
Security	Integrity secured via hash values in .torrent file.

Second, we now list the trackerless BitTorrent where the tracker is replaced with a group of peers of a global BitTorrent Peer-to-Peer system.

System	**Trackerless BitTorrent**
Bootstrap	.torrent file
Roles	Peer
Role Assignment	Peer $\Longrightarrow$ any
Identity	random ID
Membership	- register at tracker peers.
Structure	Kademlia
Query	one of eight closest peers to torrent ID
Query Strategy	lookup via Kademlia

2.4.6 EDonkey

EDonkey is an example for the superpeer-based filesharing systems. It is based on distributed servers that operate independently. Peers offer files and publish them via their servers.

System	EDonkey
Bootstrap	Server list files (server.met)
Roles	Peer, Servers
Role Assignment	Peer $\Longrightarrow$ any Server $\Longrightarrow$ if server
Identity	random ID
Membership	via network
Structure	Connect to a set of servers given in server list.
Query	for Substring, returns files and peers with chunks of file
Query Strategy	Ask servers
Cooperation Strategy	Peers offer completed chunks to others

2.4.7 SpoVNet

SpoVNet[WBHH+08] is a joint research project of the german universities Karlsruhe, Mannheim, Stuttgart, and Tübingen. It is sponsored by the Landesstiftung Baden-Wuerttemberg. SpoVNet uses an overlay approach to bridge heterogeneity and provide quality of service. It is an integrated approach with cross-layer information acquisition and optimization, event and multicast services, an underlay and a service abstraction. An offspring of the project is the Ariba Underlay Abstraction[HMMB+09]. The SpoVNet description here relates to the architecture at the time when the author left the SpoVNet project in 2008. The final project reports and publications that are to come may slightly differ.

System	SpoVNet
Bootstrap	external as well as local any/multi/broadcast
Roles	Initiator, Delegate, Peer
Role Assignment	Peer $\Longrightarrow$ any Initiator $\Longrightarrow$ fixed Delegate $\Longrightarrow$ selected by Initiator
Identity	random cryptographic identifier
Membership	defined via membership to base overlay
Structure	base overlay plus multiple service overlays
Link Attributes	RTT, Bandwidth, etc.
Node Attributes	CPU, RAM, Storage, Load, etc.
Security	- cryptographic SpoVNet identifiers - cryptographic node identifiers - initiator as authority for policies - optional: delegation of authority, distribution of authority - authentication and authorization via authority

The basic connectivity in SpoVNet is provided by the SpoVNet Base Overlay. It allows to lookup the current locator of a node given by an identifer. Most services will use direct links for their own structure. The link is then maintained by the SpoVNet base, e.g. to hide mobility. The Base Overlay provides identifier-based routing.

System	SpoVNet Base Overlay
Roles	Peer
Role Assignment	Peer $\Longrightarrow$ any
Identity	cryptographic identifier
Membership	defined via network
Query	Exact identifier lookup (only exact node IDs)
Query Strategy	Greedy

Services in SpoVNet, like the multicast service or event service, build their own overlays. These services usually do not build a structure to enable routing, but a structure according to their flow of information. This structure is optimized on the basis of cross-layer information to provide the requested service quality with high probability if this is possible.

System	**SpoVNet Overlay / Service**
Bootstrap	via SpoVNet Base Overlay
Roles	service specific
Role Assignment	service specific
Identity	from base overlay, maybe additional for service
Membership	registered within service functionality
Structure	service specific

2.4.8 Freenet

Freenet[CSWH01] is a censor-resistant system for anonymous communication and data exchange. Peers provide storage, which is used to store encrypted data chunks from data stored in the system. Peers can, thus, store and request data. Only peers that know the combination data and corresponding decryption key can read the plaintext data. In version 0.7 Freenet changed the topology to a darknet approach. As links are fixed in such networks, Freenet adapts the IDs to optimize distances in the graph. The basic operation is that two nodes from a random contact switch their IDs when this seems beneficial.

System	**Freenet v0.7**
Bootstrap	from a friend, e.g. via IRC
Roles	Peer
Role Assignment	Peer $\Longrightarrow$ any
Identity	- Public key as identity - GUID as network identifier = random ID, GUID changes to adapt to network structure
Membership	defined over network
Structure	$P \leftrightarrow P \Longrightarrow$ link to human friends of user Assumption of Small-World graph, optimize IDs not links.
Query	Exact
Query Strategy	Steepest Descent

The censor-resistance is based on the concept of plausible deniability. The nodes cannot know what they store, requests do not tell their content, and a message can be forwarded or initiate by the predecessor. The assumption is now that any node can deny responsibility for contents as well as messages that pass the node. However, one may still suffer the consequences despite being most likely innocent. That is one reason for the darknet approach, as friends will (most likely) not misuse these observations.

2.5 Relation to Architecture and Design

A Peer-to-Peer system that reaches out for the real world cannot only solve of one particular subproblem. The design of such a system has to incorporate all steps and necessary components within its life cycle. An incomplete solution will not work or it will be awkward to use. The latter is true for many published or even real systems that did not succeed on a large scale due to the consequencial lack of usability.

The components presented in this chapter represent a set of building blocks for Peer-to-Peer system design. Unlike current Peer-to-Peer teachings, the component view directly includes roles and in particular different roles for different peers. This is meant to make the architect aware of roles in the design process, and in particular make her design the roles

within a system and not monolithicly put all responsibilities and operations together in one entity. The component view can help to determine and solve the necessary problems. The components are not fixed, one could add new components when they seem to be missing. Still, this is rather easy due to its lax structure. However, if automation and formalism would be introduced, this would become more difficult.

All the necessary building block have to be in the architecture. The components can help to structure the system and its tasks. An system architect can select appropriate solutions for particular components. In particular one does not need to be innovative everywhere, but may reuse ideas. Some components are independent and design decisions may be taken freely. Of course, there is interaction between many components, in particular between the core components for the system's own functionality.

In the future, one may go beyond the current state and plug together solutions and components for the architecture or even implementation of systems. Except for components that are mostly application-specific, one may solve the above-mentioned interaction with parameterization and interfaces for system-specific adaptation.

3. Performance and Optimization

3.1 Introduction

The design of a system usually includes the optimization towards the targeted application. There may be many criteria that should be satisfied best possible. It is common that basic concepts or out-of-the-box systems are not optimal. The fundamental reason is that they do not exploit characteristics of the application in their design. Furthermore, they may be based on assumptions that are all not true for the application. As example, Distributed Hash Tables typically assume that only small values (e.g. links) are stored and transferred. Another issue is that systems may be optimized for other scenarios.

3.2 Performance and Optimization Methods

A typical performance metric is the response time for a request. This may be measured as the number of hops to the peer of interest, which ignores the latency between the various peers. This latency can be more important than the hop count.

Inefficiencies in routing can be severe when the link latencies are extremely different. A short hop count in a Peer-to-Peer network may lead to a connection via distant nodes. A typical example is to route from Munich to Stuttgart via Tokyo while a less optimal route according to the hop count may lead from Munich to Stuttgart via Frankfurt and Heidelberg. Locality-awareness is a solution to the problem. There are various ways to do this. The general idea is to establish more links that have a short latency than links with average or long latency.

Another issue is the response time of the target node and intermediate nodes, which maybe high if the request needs a lot of processing or the load is high. Load is not necessarily CPU load, but also consumption of the available bandwidth or other resources. Load is important as it changes and potentially degrades the performance of a system despite being optimal with respect to standard design and typical distance metrics.

There is yet another issue. Say, we have 10 items and 5 nodes. We may think about averaging out the number of items on all nodes and that maybe each node should end up with 2 of the items. But is that always a good solution? Let us consider the following. Let 9 items be encrypted private notes that are of no use for other users and that the remaining item is a popular file everyone who visits our Peer-to-Peer network wants. So, is the load really distributed equally when this item is on only one node, while it produces 90 % of the

requests? We conclude that popularity is also important for load. Zipf's law[AdHu02][1] says that the popularity is a power law. The consequence is extreme load for few items, the rest is insignificant. Balancing the load is therefore not enough in many cases as even one item may overload a node. Distribution of responsibility becomes necessary. Of course, not every application follows Zipf's law and many applications will request items and services in a better more balanced way. Besides of that, considering the original idea of a DHT that finds the links to sources of an item, the lookup may be rather unimportant compared to the transfer of the item itself, which might be a large file. Thus, is a skewed load within a DHT a problem? Considering resilience of the lookup and system the answer should be yes.

In the rest of this chapter we discuss some issues concerning load and locality. There are also other goals that could be used for the optimization of a system. Robustness or resilience are typical examples for goals that usually include the establishment of more links, replication of knowledge, and diversity. Now, remembering the latency optimization, this may conflict with these goals as for resilience we want German data to be in Japan and Argentina, while with respect to latency we do not want this.

3.3 Load Distribution and Load Balancing

Most structured Peer-to-Peer systems are based on the idea of consistent hashing[KLLP+97]. An expectation is that to some degree they balance load by design[2]. However, practical and theoretical evaluation has shown that the load distribution is still not always satisfying[RNGW05, NRWC05].

In the rest of this section we first discuss the distribution of nodes, which is quite important for the distribution of load. After that we discuss the distribution of load. Finally, we briefly discuss load balancing and its implications. The results will be usefull to draw conclusions for the design of Peer-to-Peer systems.

3.3.1 What is Load?

Basically, load could either be computational load, memory load, or communication load. As a Peer-to-Peer system is running on volunteering end systems, no system should demand an inappropriate amount of support from volunteers. A fair use of volunteered resources is a must.

Our considerations primarily aim at structured Peer-to-Peer systems and their idea of a distributed hash table. A typical measure for load on such systems is the number of items for which a node is responsible. In the standard DHT use case with only reference pointers the required space is small for each item. So, unless the number of items per node is large, space is not a major problem. But the load imposed by the number of requests is also of importance. Load is not only bad because it can make a computer unusable. It already suffices that it may slightly reduce the computer's quality of service. Higher load also increases energy consumption and adds costs to the volunteers[3]. The motivation of a user to contribute could be reduced if the user has to give more of her resources than others.

[1] George Kingsley Zipf (1902-1950) was a linguist at Harvard University working on statistics in languages and demographics. His original results are from the 1930s.

[2] The load on most nodes will remain within $1 + O(log\, n)$ times the average load. With replication of items on multiple nodes, this can be averaged out. If all nodes replicate all items, then all nodes have the same load.

[3] In the keynote of the IEEE Peer-to-Peer 2008 conference in Aachen Henning Schulzrinne discussed the freeriding nature of Peer-to-Peer traffic and that economic costs prohibit the further growth of free Peer-to-Peer systems. Ecological costs should also be considered as Peer-to-Peer usage may prevent the hibernation of an otherwise unused computer. To draw the conclusion, the common Peer-to-Peer assumption of using existing otherwise unused (but paid) resources does not (completely) hold.

If load in a Peer-to-Peer system is extremely skewed the advantages of having a decentralized distributed system vanish. This is rather obvious as a distributed system with one extremely popular item would degrade to a single server if only that single peer with that item can serve all the requests.

Items may not only be simple data items. Modern Peer-to-Peer system abstract from the idea of a simple hash table. They distribute and replicate not only small items, but also a multitude of services and complex operations. An item in that case may be a task for a service or the service itself, e.g. being source for the streaming of a video.

The most common measure for load in Peer-to-Peer literature is the number of items that a node stores. This is a rather simple measure and very generic. We do not need any information about the items and, in particular, about system details or even hardware. However, that is also a drawback as items may vary a lot in popularity. Popularity can be solved to some degree by giving popular items a higher weight, e.g. by counting or adding them multiple times in the system. More difficult to solve is the heterogeneity of peers. The same absolute load may have a different impact on peers when available resources on some peers are larger and on some the resources are smaller than required by the load. One could think of systems with PDAs, cell phones, and new and old PCs.

The number of requests on a peer is a measure that can also include popularity. Counting the overall traffic on a peer is another option. A practical measure is to count the amount of bandwidth used. As bandwidth is usually limited by a given value (say, limit Peer-to-Peer system upload to 64 kbps), one can easily determine the load relative to the maximum acceptable load. Yet, bandwidth comsumption of a system is not easy to model.

Computational intensive systems will count CPU usage or, less fine-grained, tasks that were processed within recent time frames.

In our statistical analysis we will stick with the standard item count as measure for the load. While other measures are also interesting, they are not suited for a general and generic analysis.

3.3.2 Modelling Node Distribution

The first step is to model how nodes distribute over the ID space. There is also the first misunderstanding. While their IDs may usually be drawn uniformly from the ID space, this does not imply a uniform density of nodes on the ring.

Distribution of Nodes

The distribution of nodes in a structured Peer-to-Peer system heavily depends on the way node IDs are assigned. Some systems propose to use uniform random ID assignment. Another common proposal is to use a hash of the locator (e.g. IP address, 160 bit SHA-1 hash in case of Chord). This has two implications. First, the positioning in that case can hardly be distinguished from the random ID assignment with all values being equally likely. Second, the node ID assignment for one node is independent from the node ID assignments for other nodes.

As a consequence, the first assumption of our model is that the ID of a node is a random variable following the uniform distribution. Second, the node IDs are considered to be independent. The Peer-to-Peer system consists of n nodes and without loss of generality, we assume that the node we look at has ID $= 0$ (Fig. 3.1(a)). As a consequence, the interval is determined as the minimum over the node IDs of the other nodes (Fig. 3.1(b-c)).

To determine the distribution of the interval size we assume $n-1$ experiments (IID, uniform random variable) and take the minimum of these experiment.

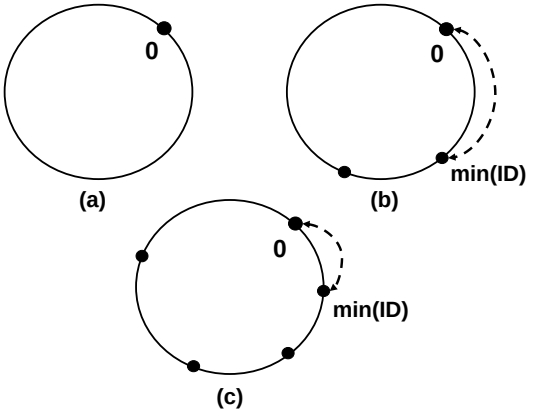


Figure 3.1: Interval size as minimum of IDs of all other nodes.

Continuous Model

First, we use a Continuous Model, i.e. we consider the ID space to be real-valued in the interval $[0, 1)$. The motivation for this continuous model is that it is easier to analyze than a discrete one and the ID space is large compared to the number of nodes in it.

The continuous uniform distribution is defined as:

$$U(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x < 1 \\ 1 & x \geq 1 \end{cases}$$

For the statistical analysis it does not matter if the node at the beginning or the end of the interval is responsible for the data. Thus, we can assign the interval to the node that makes it easier to model. In our case the predecessor of an interval is the owner of the interval.

Let L be the distribution of the interval size. It is given as minimum of $n-1$ experiments:

$$L(x) = 1 - \prod_{i=1}^{n-1} (1 - U(x)) = 1 - (1 - U(x))^{n-1} = \begin{cases} 0 & x < 0 \\ 1 & x \geq 1 \\ 1 - (1-x)^{n-1} & else \end{cases}$$

And the probability density function:

$$l(x) = \frac{dL}{dx} = \begin{cases} (n-1)\,(1-x)^{n-2} & 0 \leq x < 1 \\ 0 & else \end{cases}$$

As expected, for $n = 1$ (only one node) the interval size is 1, i.e. the complete ID space. For $n = 2$ (2 nodes) the interval size is distributed with U(0,1). With more and more nodes in the ID space the interval size is approaching 0.

Figure 3.2 shows the density $l(x)$ of the distribution. Interestingly, this graph has a similar form as graphs plotting empirical item distributions, e.g. in [RNGW05]. There are many nodes with little load and the higher the load the fewer the number of such nodes. Ideal would be a different distribution. Ideal would be a single peak with all nodes at the average load.

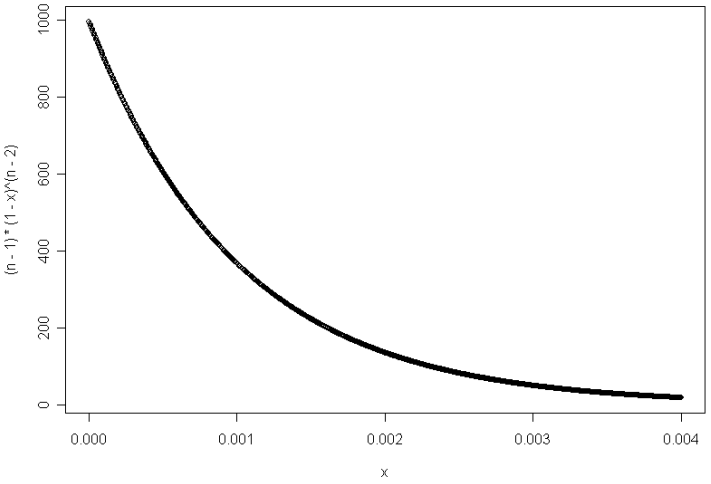


Figure 3.2: The continous density function l(x) with $n = 1000$. The average interval size in that case is $\frac{1}{n} = 0.001$

.

As a next step, we want to calculate the variance of the node distribution. Obviously, the average of the distribution is $\frac{1}{n}$.

$$VAR(L(x)) = \int_0^1 \left(x - \frac{1}{n}\right)^2 l(x)dx$$

$$= \int_0^1 \left(x - \frac{1}{n}\right)^2 (n-1)(1-x)^{n-2}dx$$

$$= \int_0^1 \left(x^2 - \frac{2}{n}x + \frac{1}{n^2}\right)(n-1)(1-x)^{n-2}dx$$

We now solve the integral for the three summands ($A = x^2$, $B = -\frac{2}{n}x$, and $C = \frac{1}{n^2}$) independently.

$$\int_0^1 Al(x)dx = \int_0^1 x^2(n-1)(1-x)^{n-2}dx$$

We now apply the integration by parts rule with $f(x) = x^2$ and $g'(x) = (n-1)(1-x)^{n-2}$. Consequently, $f'(x) = 2x$ and $g(x) = -(1-x)^{n-1}$

$$= \left[x^2(-1)(1-x)^{n-1}\right]_0^1 - \int_0^1 2x(-1)(1-x)^{n-1}\,dx$$

We now apply the integration by parts rule with $f(x) = 2x$ and $g'(x) = -(1-x)^{n-1}$. Consequently, $f'(x) = 2$ and $g(x) = \frac{1}{(n)}(1-x)^n$

$$= 0 - \left[2x\frac{1}{(n)}(1-x)^n\right]_0^1 + \int_0^1 2\frac{1}{(n)}(1-x)^n\,dx$$

$$= -0 + \left[\frac{-2}{n(n+1)}\left(1-x\right)^{n+1}\right]_0^1 = -0 + 0 - \frac{-2}{n(n+1)} = \frac{2}{n(n+1)}$$

Now, we similarly calculate the integral for the other summand.

$$\int_0^1 Bl(x)dx = \int_0^1 -\frac{2}{n}x\left(n-1\right)\left(1-x\right)^{n-2}dx$$

Again we use partial integration with $f(x) = -\frac{2}{n}x$ and $g'(x) = \left(n-1\right)\left(1-x\right)^{n-2}$. Then $f'(x) = -\frac{2}{n}$ and $g(x) = -\left(1-x\right)^{n-1}$.

$$= \left[-\frac{2}{n}x\Big(-1\right)\left(1-x\right)^{n-1}]_0^1 - \int_0^1 -\frac{2}{n}(-1)\left(1-x\right)^{n-1}dx$$

$$= \left[-\frac{2}{n}x\Big(-1\right)\left(1-x\right)^{n-1}]_0^1 - \int_0^1 -\frac{2}{n}(-1)\left(1-x\right)^{n-1}dx$$

$$= 0 - \left[-\frac{2}{n}\frac{1}{n}(-1)\left(1-x\right)^{n}\left(-1\right)\right]_0^1 = 0 - \frac{2}{n^2}$$

Now, for the third summand.

$$\int_0^1 Cl(x)dx = \int_0^1 \frac{1}{n^2}\left(n-1\right)\left(1-x\right)^{n-2}dx$$

$$= \left[-\frac{1}{n^2}\left(1-x\right)^{n-1}\right]_0^1 = \frac{1}{n^2}$$

Thus, for overall variance we get,

$$VAR(L(x)) = \frac{2}{n(n+1)} - \frac{2}{n^2} + \frac{1}{n^2}.$$

Standard deviation is thus roughly $\frac{1}{n}$ and variation roughly 1. This is similar to the variation of the exponential distribution (exactly 1).

Discrete Model

In this section we transform our continuous model into a discrete one. The only difference is that we consider the ID space to be integers in the interval $[0, 2^m - 1]$.

The discrete uniform distribution is defined as:

$$U_{discrete}(i) = \begin{cases} 0 & i < 0 \\ \frac{i}{2^m} & 0 \leq i < n \\ 1 & i \geq 2^m \end{cases}$$

Minimum of $n-1$ experiments:

$$L_{discrete}(i) = 1 - (1 - U_{discrete}(i))^{n-1} = 1 - \left(1 - \frac{i}{2^m}\right)^{n-1}$$

The probabilities can be derived similar to the pdf of the continuous model:

$$l_{discrete}(i) = L_{discrete}(i+1) - L_{discrete}(i) = \left(1 - \frac{i}{2^m}\right)^{n-1} - \left(1 - \frac{i+1}{2^m}\right)^{n-1}$$

We will now further solve this equation and show that like $l(x)$ $l_{discrete}(i)$ is a polynomial of degree $n-2$.

$$= \sum_{j=0}^{n-1} \binom{n-1}{j} 1^j (-1)^{n-1-j} (\frac{i}{2^m})^{n-1-j} - \sum_{j=0}^{n-1} \binom{n-1}{j} 1^j (-1)^{n-1-j} (\frac{i+1}{2^m})^{n-1-j}$$

$$= \sum_{j=0}^{n-1} \binom{n-1}{j} (-1)^{n-1-j} (\frac{i}{2^m})^{n-1-j} - \sum_{j=0}^{n-1} \binom{n-1}{j} (-1)^{n-1-j} (\frac{i+1}{2^m})^{n-1-j}$$

$$= \sum_{j=0}^{n-1} \binom{n-1}{j} \left(\frac{-1}{2^m}\right)^{n-1-j} i^{n-1-j} - \sum_{j=0}^{n-1} \binom{n-1}{j} \left(\frac{-1}{2^m}\right)^{n-1-j} (i+1)^{n-1-j}$$

$$= \sum_{j=0}^{n-1} \binom{n-1}{j} \left(\frac{-1}{2^m}\right)^{n-1-j} i^{n-1-j} - \sum_{j=0}^{n-1} \binom{n-1}{j} \left(\frac{-1}{2^m}\right)^{n-1-j} \sum_{k=0}^{n-1-j} \binom{n-1-j}{k} i^k$$

Now, we can rewrite it to

$$= \sum_{j=0}^{n-1} \binom{n-1}{j} \left(\frac{-1}{2^m}\right)^{j} i^{j} - \sum_{j=0}^{n-1} \binom{n-1}{j} \left(\frac{-1}{2^m}\right)^{j} \sum_{k=0}^{j} \binom{j}{k} i^k$$

$$= \sum_{j=0}^{n-1} \binom{n-1}{j} \left(\frac{-1}{2^m}\right)^{j} i^{j} - \sum_{k=0}^{n-1} \binom{n-1}{k} \left(\frac{-1}{2^m}\right)^{k} \sum_{j=0}^{k} \binom{k}{j} i^j$$

$$= \sum_{j=0}^{n-1} i^j \left[\binom{n-1}{j} \left(\frac{-1}{2^m}\right)^{j} - \sum_{k=j}^{n-1} \binom{k}{j} \binom{n-1}{k} \left(\frac{-1}{2^m}\right)^{k} \right]$$

For the summand with $j = n-1$ we end up with

$$i^{n-1} \left[\binom{n-1}{n-1} \left(\frac{-1}{2^m}\right)^{n-1} - \binom{n-1}{n-1} \binom{n-1}{n-1} \left(\frac{-1}{2^m}\right)^{n-1} \right] = \left(\frac{-1}{2^m}\right)^{n-1} - \left(\frac{-1}{2^m}\right)^{n-1} = 0$$

For the summand with $j = n-2$ we can show that the formula is non-zero.

$$i^{n-2} \left[\binom{n-1}{n-2} \left(\frac{-1}{2^m}\right)^{n-2} - \sum_{k=n-2}^{n-1} \binom{k}{n-2} \binom{n-1}{k} \left(\frac{-1}{2^m}\right)^{k} \right]$$

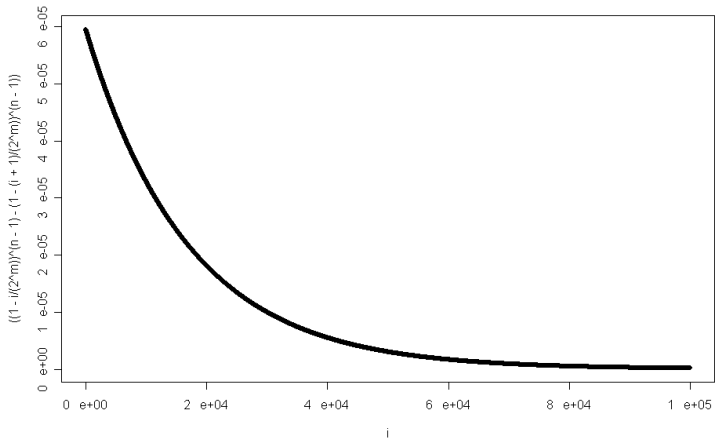


Figure 3.3: The probabilities $l_{discrete}(x)$ with $n = 1000$ and $m = 24$.

$$= i^{n-2}\left[\binom{n-1}{n-2}\left(\frac{-1}{2^m}\right)^{n-2} - \binom{n-2}{n-2}\binom{n-1}{n-2}\left(\frac{-1}{2^m}\right)^{n-2} + \binom{n-1}{n-2}\binom{n-1}{n-1}\left(\frac{-1}{2^m}\right)^{n-1}\right]$$

$$= i^{n-2}(n-1)\left(\frac{-1}{2^m}\right)^{n-1}$$

So, we end up at a polynomial with degree $n-2$.

$$l_{discrete}(i) = \sum_{j=0}^{n-2} i^j \left[\binom{n-1}{j}\left(\frac{-1}{2^m}\right)^j - \sum_{k=j}^{n-2}\binom{k}{j}\binom{n-1}{k}\left(\frac{-1}{2^m}\right)^k\right]$$

Fig. 3.3 shows $l_{discrete}(i)$, i.e. the probabilities for intervals of size i. The form is similar to the continuous $l(x)$.

3.3.3 Load Distribution

In the Peer-to-Peer literature on load balancing one usually considers the load to be the number of items a node is storing. In a typical DHT application, the item is stored on the node that is responsible for a hash value of the item name. We can call this hash value the identifier of the item. Given this setting, this is also equivalent to a uniform random ID assignment.

From the perspective of a node the assignment of an item to itself is a random process. The probability p of being assigned an item corresponds to the percentage of ID space the node owns. The items are all independent. So, the load of a node follows the binomial distribution with m items and hit probability p.

$$P(Load = k) = \binom{m}{k} p^k (1-p)^{m-k}$$

p, however, is not fixed. It corresponds to each node's interval size, which follows the node distribution from the previous section.

The variation of the binomial distribution is $\sqrt{\frac{1-p}{mp}}$. As a consequence,

$$m \to \infty \Rightarrow \text{the variation } c_{binom} \to 0.$$

It will, thus, perfectly adapt to the size given by the node distribution. Now, assuming that the expected interval size is $\frac{1}{n}$, we can write the variation slightly different for intervals with average size. Then $p = \frac{1}{n}$ and the variation $c_{binom,1/n} = \sqrt{\frac{1-\frac{1}{n}}{\frac{m}{n}}} = \sqrt{\frac{n-1}{m}}$. So, for an average interval and nine times more items than nodes ($m = 9n$), the variation is approximately $\frac{1}{3}$. Of course, this is just a crude approximation and for large intervals the variation will be lower and for small intervals larger. This, however, is also good for our understanding as primarily nodes with small intervals differ significantly from the expected load given their interval size. While the more relevant nodes with large intervals and thus most likely higher load follow the node distribution.

We therefore conclude that the node distribution determines the load distribution if load is high (many items per node) and the item assignment follows a uniform pseudorandom process.

3.3.4 Load Balancing

In the previous sections we analyzed the balance of the load. In this section, we briefly introduce methods to balance a skewed load.

Methods

We introduce two basic ideas for the balancing of load and furthermore we introduce application-specific methods as alternative based on an example for online gaming. The first idea to balance load is to move work from one node to another. The second idea is to share responsiblity. Details follow in the rest of the section.

Basic idea 1: Distribute Work

If there is an imbalance, one can e.g. react to the imbalance and move work from an overloaded node (heavy node) to a node with less load (light node). The other option is to proactively smoothen the load so that an online movement of work is not necessary. Both options can be combined, which seems to be the best solution for load-sensitive systems.

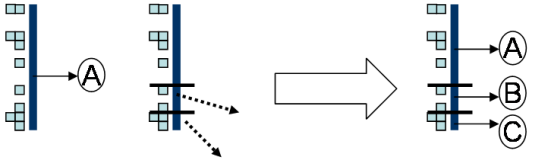


Figure 3.4: Overloaded peer A distributes work to peers B and C. Parts of the intervals are transfered to other peers.

The most well-known reactive method is the Virtual Server[RLSK+03] approach. Each node controls multiple intervals of the ID space. It is thus multiple times in the network. The recommendation is that each node controls $O(logn)$ intervals with n being the number of real nodes. The basic idea is now to move intervals from heavy to light nodes. Nodes can determine if they are heavy or light according to their load compared to the load of the neighbors. To find each other, nodes can randomly check other nodes. The authors of the Virtual Server method propose to use a directory server to find light/heavy nodes.

A simpler and mathematically better analyzed approach is the Item Balancing of Karger and Ruhl[KaRu04]. Nodes randomly connect each other, say node A and B. Let B be the heavier node. If $load_A < \epsilon\, load_B$, then A moves to the interval of B. Both nodes fairly divide the interval, so that they share the same number of items. As a consequence, $load_{A,new} \approx load_{B,new}$, and for A's previous neighbor C $load_{C,new} = load_C + load_A$.

Now to the proactive methods, from our own analysis of the last section we concluded that the balance of nodes within the ID space is most important for the load. We therefore propose to *position nodes according to the load*. If items are uniformly distributed, the nodes should also be uniformly distributed. If the item distribution is different, the node distribution should follow. The positioning of the nodes should happen in a way that with high probability the load will be balanced.

The Node Balancing of Karger and Ruhl[KaRu04] is a proposal to balance the nodes for the case of uniform item distribution. Their approach actually proposes to use node balancing in the uniform case and item distribution in the non-uniform case. Considering the goal of node balancing, what we want is that each peer ends up with $\frac{1}{2}$ the ID space for $n = 2$, each node with $\frac{1}{3}$ for $n = 3$, etc. If we do that, we would need to reorganize all nodes for each change in the network. This is no option. The authors propose a compromise that reduces fragmentation into small intervals yet operates locally. Each nodes virtually has $O(log\, n)$ virtual servers. However, contrary to the virtual server approach only one of the servers is actually in the network for each peer. This so-called active virtual server is selected according to interval size. Let us consider the address space as rational numbers $0, \frac{1}{2^{bitlength}}, \ldots, \frac{2^{bitlength}-1}{2^{bitlength}}$. The node selects the active node according to the interval with the smallest divisor (= only significant bits set), and among those the smallest interval. This gives the order $1 < \frac{1}{2} < \frac{1}{4} < \frac{3}{4} < \frac{1}{8} < \frac{3}{8} < ..$ As for their Item Balancing the authors mathematically analyze their approach and prove that it balances nodes with high probability. Strictly applied it may, however, force a lot of changes of node position during their lifetime.

Basic idea 2: Share Responsibility

The second idea is to share responsibility. Not only one node is responsible for an item, but the item is distributed to many nodes and they all can answer the query. Thus, the method is to *share an item and its load with other nodes*. This has two effects. First, it helps to fight load of heavy popular items. Second, if applied correctly, it can also average out imbalances of the load.

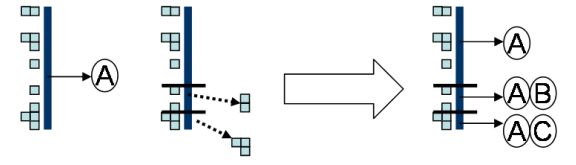


Figure 3.5: Overloaded peer A shares work with peers B and C. Some of the items / parts of the interval are maintained cooperatively.

One of the first load balancing proposals for DHTs can be classified as a such an approach. This is the Power of Two Choices method[Mitz96, ByCM03]. The basic idea here is to position not the nodes but the items multiple times in network. In the article of the proposal this is done by using different hash functions to determine the item IDs for an item. For the lookup, nodes make a query for either one of the item IDs of the item of interest. Thus, the load of an item is distributed among the peers responsible for the item.

However, the primary effect of this approach is that it better adapts the item distribution to the node distribution as it doubles the number of items m and, thus, reduces the variance of the distribution for one node and its load. The approach, however, does not help when the node balance is skewed. For averaging the number of items, this approach does not help as it applies the idea of sharing in a wrong way. Thus, it should be combined with node balancing unless it is only used to mitigate the load of popular item.

A better idea for the sharing of items is the idea of replication. Replicas in most DHT approaches are shared with nodes in the neighborhood. These nodes could be the k direct neighbors. Another option is to cache lookup results for recent lookups as replicas. Replication for load balancing requires that the replicas need to be on the lookup path of requests to the item. Chord e.g. proposed to replicate to the successors. This helps to recover from failures, but it does not balance the load as none of these replicas can be found by a lookup request. A better approach is to store the replicas in the neighbor set as in Pastry. All of these nodes could be on the lookup path due to the Pastry lookup algorithm (tree-based structure). Thus, the replicas can average out the load. Furthermore, as replicas are stored on a fixed number of neighbors, their interval size does not matter and, thus, it always averages out load as load distribution does not converge to the node distribution.

Another idea is not to distribute the items in the network, but to *share an interval with other nodes*. The original CAN paper[RFHK01] discussed to have more than one peer per interval for non-load balancing reasons. A load balancing proposal that combines this idea with ideas from item balancing and virtual server approach is the Thermal Dissipation approach by Rieche et al.[RiPW04b]. The approach defines a minimal number of nodes for an interval and a maximal number. Intervals can split and merge. The authors use the notion of light and heavy 'servers' (nodes) to trigger these actions.

Application-specific Load Balancing - Example: Massively Multiplayer Online Games

Besides these basic fundamental ideas for balancing load, there are application-specific methods. They will usually be a combination of the ideas above applied in a manner that suits the needs of the application.

In joint work with Simon Rieche, Marc Fouquet, Leo Petrak and the students Jirko Cernik and Timo Teifel we studied how Peer-to-Peer systems could contribute to the increasingly popular field of Massively-Multiplayer Online Roleplaying Games (MMORPGs)[RWFN$^+$07, RWFN$^+$08a]. In such games users roam around the world of the game and interact with other users and entities in the world that are close to their own location in the game world. As a consequence, the server of such games needs to maintain state about their items, their properties, and their positions. It needs to inform each user about the entities it sees, and it manages all their transactions (transaction of items, fights, etc.).

The Quality of Experience of a user depends on if information about the new state ('update') is transfered in time. Updates should not pass many nodes before they reach their destination. Updates for a node should also not be queued too long in the server if too many other updates for other nodes need also to be sent. We conclude, any set of peers that interact in the game world should have a common server—if possible. As there can be no strict separation and sets will overlap, for many sets multiple servers will handle the operations. The more servers, however, the more messages have to be sent for synchronization and inter-server transactions among the servers.

Zone-based Load Balancing

Now, the straight-forward idea to make the system Peer-to-Peer is to divide the map into zones where a peer is responsible for a zone. Figure 3.7 shows four zones, each with a

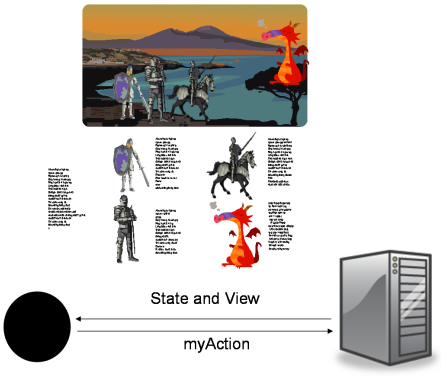


Figure 3.6: A peer in an online roleplaying game interacts with the game and other players via the server. The server is trusted and in control.

server operating for the clients in the zone. The figure also shows what is common in today's MMORPGs. The zones are separated and in the game mountains or the sea as borders may prevent overlap of a player's line of sight and another zones. At fixed gateway points the players may move from one to another. This is not the most beautiful option and the zone-based Peer-to-Peer proposals consider such overlaps and interaction across the zone boundaries. There are some proposals for zone-based approaches[Lu04, IiHK04]. Our zone-based approach takes its idea from the Content Addressable Network (CAN) [RFHK01] and its zones are rectangles that are divided either vertically or horizontally, and merged accordingly. The servers are virtual servers and a peer may host more than one virtual server depending on game status and load.

As interaction across zone boundaries is expensive and problematic due to latency we additionally tried to optimize the split so that the map is divided in a way that the interaction across the borders are minimized. The basic idea is to divide a large zone into smaller zones not at crowded but at lonely places.

We studied some ideas how to split the CAN intervals accordingly. We evaluated our approach on the basis of simulation with OMNeT++. We used artificial traces for the player movements as well as traces from the browser roleplaying game Freewar[4]. First, it seemed that the load balancing degrades when we optimize the split. The variation of the load was higher. The reason was that the triggers did not further optimize the load when it was satisfying for all servers. Here we see a fallacy load balancing can run into. You can put a lot of effort into making it completely equal, but there is no profit from it, yet overhead increases. For the optimized split variants, the overall load of inter-server communication was greatly reduced. This is a result we first did not appreciate as the original scenario was a server farm and internal traffic was not our focus.

From today's point of view our conclusion to prefer the original CAN split might have been wrong. However, it seemed to be more robust against parameter changes and for a practical system stability can be more important.

As already mentioned, in our approach we wanted to create a practical approach that could become realistic in the near future. For that reason we were not yet satisfied with these ideas. A fundamental problem is cheating. Even in a server setting there are many ways to cheat and the prevention is hard[5]. As a consequence, we propose a super peer

[4]http://www.freewar.de

[5] A common way to cheat is to delay gaming messages in order to create a reordering of critical events. In a shooter this would be to delay position updates in order to blur one's position to avoid being hit by a

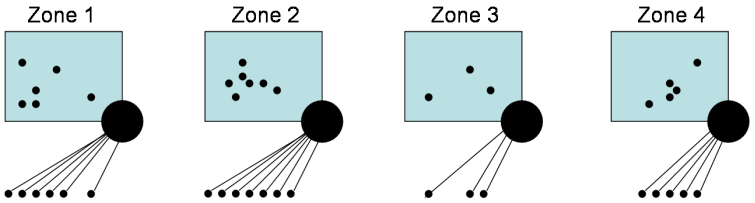


Figure 3.7: Servers with their respective zones and peers.

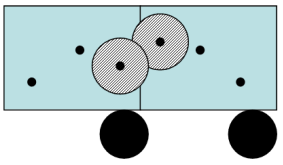


Figure 3.8: If zones are not strictly separated, the peer's line of sight and their action radius may cross zone boundaries.

approach. A player peer (or 'client peer') is a client to the super peer that is responsible for its place in the game world. The super peers that perform the distributed operation of the server are trusted peers, e.g. provided by the game provider[6].

Clustering of Multiplayer Groups

A common concept of roleplaying games is that groups of users play together (e.g. go to the same place) and that there are places that are popular (either due to the story or from what is located there – like a bank). Now, when we do not want interaction and transactions across multiple servers, one might want to consider zones of users instead of zones on the map of the game. And that is one of our contributions to this MMORPG research effort. Now, the idea is that we use clustering to group players and that the zones are these clusters. The borders of the clusters are not as strictly set as zones on the map. For stability reasons we allow to have a small overlap between clusters to avoid unnecessary transfer of users.

Clusters are built on the basis of small distance between peers. If a cluster is too large or its load too high it can be split and the split is done by selecting a split with maximum distance from a sample set of six potential clusters. The number comes from the usage of extreme points (left/top and right/bottom, etc.) as the initial nodes for the two clusters in the clustering algorithm. From the sample set the split where the two clusters have the largest distance is selected.

As expected, the cluster based approach is not as good at equalizing the load, but it performs better at reducing foreign position changes (= communication across borders). With respect to handovers of players between servers there was no significant advantage for one of the approaches. However, the cluster-based method moved less players in most of the simulations.

bullet. In role-playing games automation is another common way to cheat, i.e. by using bots to automate certain actions.

[6]We do not care about them being provided, but any set of trusted peers that operate faithfully is compatible with the idea. The servers could also be peers in the game. However, we do not deal with conflicting interests, which could be handled by assigning peers to server tasks for distant zones.

Table 3.1: Comparison of the Cluster-based approach and the CAN-based approach

Trace	Method	Handovers	FPC messages
freewar2	Cluster	8038	48441
freewar2	CAN	4813	75325
group4	Cluster	5540	26687
group4	CAN	12457	130778
group5	Cluster	22542	198653
group5	CAN	13441	205806
group6	Cluster	13093	90528
group6	CAN	18467	280210
pursue3	Cluster	32427	379707
pursue3	CAN	35685	192167
pursue b1	Cluster	8877	45239
pursue b1	CAN	7521	84829
rand1	Cluster	6730	24500
rand1	CAN	21312	260076
rand3	Cluster	14889	79492
rand3	CAN	28198	379392

To conclude, we have discussed the load balancing for an application, here Massively Multiplayer Online Roleplaying Games. We have seen that optimizing the balance beyond necessity can add complexity and increase load. For the game, there is no consensus what is best, in particular as clustering is computationally more expensive. We think that in more interactive and group-oriented gaming clustering will be the better choice. For further information on these research efforts and related work we refer to the publications [RFNW+05, RFNP+06, RWFN+07, RWFN+08b, RWFN+08a].

When do we need Load Balancing?

There can be two reasons for no need. First, the load within the system is insignificant. Second, there is no significant imbalance (load imposed by balancing >= profit from balancing).

One needs a model for the load of a system to decide if it can be significant. The idea is to develop a good understanding of the properties of the system and its usage. In particular it should be clear that the assumptions for the insignificance of load do not change during the lifetime of the system. As the system may progress, they may not hold anymore at some point in time. In particular the possibility of hotspots should be discussed and there should be a relief for hotspots even if they do not correspond to high load. One reason is that resilience can profit from it and that users may feel better if their resources are not abused.

If there is enough load, one may now ask if there is a significant imbalance in the load. There are various sources of imbalance. One source is the *node distribution*. The number of items or tasks a node has to handle relates directly to the size of its interval. This assumes a uniform item distribution and can be mitigated by balancing the nodes more uniformly over the ID space. Yet, another source of imbalance is the *imbalance of tasks* (Item Distribution) within the ID space. So, some regions in the ID space may need more nodes as more items are stored there. Another source is the *imbalance of popularity of items or tasks*. Some items may trigger more requests than others. Another similar source is the *imalance of complexity of items or tasks*. The filesize of an item or the processing of

a task can differ. The mitigation for both of these sources is to share responsibility, either by enabling a set of nodes to process a request or by cooperative storage and processing.

Advantages of load balancing are a more equal distribution of the load. The load for heavy (potentially overloaded) nodes is reduced, hopefully to an acceptably low load. The resilience of a system also profits when more nodes really contribute to its operation and can replace each other.

3.4 Locality

Overlay routing mechanisms send packets via paths where they do not necessarily reduce the distance to the target on each hop in terms of the target's location in the underlay. Links between neighbors can also be quite long without necessity[7]. Another node in the overlay might be as good in overlay terms and better with respect to underlay latency.

Locality is not purely a question of geography as the introductory example of the chapter with a German connection via Japan might suggest. One other reason is the existence of heterogeneous access technologies. There are some access technologies (most wireless technologies, DSL without fastpath) where the first hop within the underlay already contributes significantly to the latency. Another reason is that each provider has its own network and that they interconnect to networks of other providers only at some interconnection points.

To determine how an overlay performs in comparison to the underlay and other overlays, one needs a way to measure its routing inefficiencies. The most common measure for this is the stretch.

Definition 3.1 (Stretch) *For nodes A and B in an overlay, the stretch of their connection via the overlay is given by*

$$Stretch_{A \to B} = \frac{distanceOverlay(A \to B)}{distanceUnderlay(A \to B)}$$

The stretch for a Peer-to-Peer overlay network V is the average stretch for all node combinations.

$$Stretch_V = avg_{A,B \in V} Stretch(A \to B)$$

The definition of stretch does not necessarily limit stretch to distance based on latency. There are also definitions of stretch where the distance is based on the hop count in the underlay. We call them $Stretch_{Delay}$ and $Stretch_{Hop}$ later in the chapter.

Another measure for locality is the average distance of a link in a system. However, this measure is neither normalized, nor does it take path lengths into account. It is possible to build stupid systems that would be optimal with respect to this measure, but that do not provide a connectivity between all peers. Example could be a system that only maintains short links and skips all long links. Thus, the network may partition or due to an increased number of hops the overall latency may be worse.

Similar to load balancing one might also include popularity and weight link distances or paths to nodes with their popularity. Locality-awareness is not for free. One needs to find nodes with short distance, maintain the information, and maybe increase the routing table or even rework the overall network.

[7] The underlay may also contain long links, their length is commonly not unmotivated, but a fact from connecting a long distance in the real world (e.g. crossing the Atlantic ocean).

In the following sections we introduce methods to optimize locality and discuss subtasks that have to be solved when applying them. Finally, we present a short evaluation of Proximity Node Selection and end with recommendations for locality-aware Peer-to-Peer design.

3.4.1 Methods

Peer-to-Peer systems consist of end-systems that are potentially located all around the world. Peers within the core of the network are rather unlikely[8], they are typically at the edge of the Internet. So, there is no natural hierarchy, no peer that has to be passed anyway when communicating with other peers. As a consequence, there is no preferred peer, any close peer in a near network may be a good choice. One could also consider a locality-optimal network as a network with a set of local clusters. Peers route messages within their cluster, if possible, and then traverse the core to the cluster of the target node. The methods discussed in the rest of this section try to the topology a bit towards this ideal. Of course, reality with its cost-efficiency trade-offs does not allow perfect routing with 'stretch close to 1'. There are also differences within the nodes of these virtual local clusters. Some nodes have a better connection and may therefore be located closer to the core than other nodes[9]. They may also be prefered by their neighbors as well as by distant nodes, but a peer would not chose them to get to a known distant peer.

In the following we introduce the basic approaches to implement locality-awareness into a Peer-to-Peer system. The general goal of all approaches is to replace long links with short links in a way that the replacement does not, on average, affect overall overlay routing. So, we need to replace long links with 'equivalent' short links.

The first idea one might have to implement a locality-aware system is to maintain a set of close nodes and add them as additional links to the routing table. If a packet is forwarded, the node selects the next hop from the routing table on the basis of underlay distance in combination with progress in overlay ID space. Thus, the peer selects the route on the fly on the basis of proximity. It is not necessary (although better) to enlarge the routing table with additional nodes to achieve such a location-aware routing. One can also only use the nodes of the unaltered routing table to trade-off progress in ID space with cost in the underlay. Let us use Chord as example. The ith entry of the finger table of a node with ID ID_{node} is the first node in the interval $[ID_{node} + 2^i, ID_{node} + 2^{i+1})$. The fingers to the intervals preceeding the i-th interval could also be a good choice. One may now select one of them (i-1, i-2, ...) instead when its current RTT is much lower than the RTT to the peer from the correct interval. This concept is called *Proximity Route Selection (PRS)*.

Yet if we have multiple candidates into one direction, if a node is a good next hop for one direction — why keep all other routes and not select this best one? This is not necessary. The answer is *Proximity Node Selection (PNS)*. For each direction in the routing table a node fills its routing table with the closest node it finds. PNS does not need to consider and keep additional routes. It determines the best and selects this one without on-the-fly decision and trade-off. In the literature PNS is preferred over PRS[GGGR+03]. The advantage of PNS over PRS in the research of Gummadi et al. comes from the fact that they only use the nodes of the normal routing table without additional neighbor set. There is no change in the size of the routing table. The consequence for PRS is that the routing table is small and the few nodes (say 16 out of 65,000 as in their example) in the routing table are not likely to be close neighbors. PNS profits from searching for candidates for

[8] Except for very special Peer-to-Peer systems, which try to cooperate with and include the ISP, or maybe in a new future Internet.

[9] Among other topological reasons, this is caused by different and high latencies of some access technologies, e.g. DSL or UMTS

each direction and then selecting the best one. As a consequence, PNS checked $16 * k$, say 64 with $k = 4$, peers for proximity. It is, thus, less a question of PNS or PRS than rather a question of sampling the network for good nodes or not.

So far we introduced only approaches that optimize the routing in the way that they try to find closer nodes for each direction. Locality-awareness can operate also the other way around. Nodes can select their ID on the basis of locality. This is called *Proximity Identifier Selection (PIS)*. PIS systems can be based on IP addresses as identifiers[GeRBF+03]. Another option are Internet Coordinate Systems that compute coordinates (identifiers) based on latency measurements[DCKM04]. A node is positioned in a k-dimensional space according to the measurements.

Proximity Identifier Selection interferes with overlay algorithms, so that many systems cannot use it[10]. PNS and PRS are easier to integrate into the typical structured Peer-to-Peer systems. However, PIS can also be used in parallel to the overlay network and its identifiers. It may be used to directly find close nodes that can be used to implement PNS or PRS in an overlay that is not compatible with PIS.

3.4.2 How to find near nodes?

The first question that arises when designing locality-aware systems is how to find near nodes.

A simple yet efficient idea is to randomly *scan the network* for potential candidates. In many structured Peer-to-Peer systems, a node may select any node as routing table entry that is from the corresponding interval in the ID space. Any node within this part of the ID space is fine. The peer can find nodes within this interval with lookups for IDs within in this interval. An exhaustive search of the ID space would find the best node with an ID from that interval. In large networks such an exhaustive search is too resource-intensive.

As a consequence, a node may only search for a fixed number k of nodes and select the best among them. There are basically two options. One can lookup IDs. In that case there is the danger to hit a node multiple times and some nodes will have more hits due to their large interval. Another option is to randomly select one and its $k - 1$ successors if they exist. If node IDs are random, there should be no locality-relevant statistical difference between successors and random nodes[11].

Another option is to create a *distributed data structure* apart from the overlay and maintain local and distant nodes within this structure. Karger and Ruhl describe such an approach in [KaRu02]. Similarily, one may use an Internet Coordinate System.

Very efficient with respect to overhead is to use the dynamics and usage of the Peer-to-Peer network to find near nodes with little additional effort. When nodes interact the *nodes share their knowledge and check their distance*. If the distance is small, they register as neighbors and update their routing tables. An example: Pastry assumes that a node joins the network via a local rendezvous peer. This may be because social networks might drive the join process (e.g. a website in the local language). Let us assume the new peer and the rendezvous peer are close. The rendezvous peer already knows other close nodes (neighbor set in Pastry) and it already applied locality mechanisms on its own routing table. The rendezvous peer transfers all of its knowledge to the new peer. So, the new peer already has a number of good candidates for the routing table entries it has to fill. The next step

[10] Examples: Load Balancing may force a peer to have multiple IDs all over the ID space. Identifiers may also be chosen by content (content-based addressing) or as cryptographic identifiers (derived from the public key of a the peer).

[11] A counterexample would be node IDs on the basis of IP addresses. Note that hashed IP addresses do not matter. Any node ID generation that preserves the locality properties of the IP address would matter.

for the new peer is to lookup its location in the overlay. As a consequence it gets to know the neighbor set and routing entries of all the intermediate peers that are, hopefully, also closer to it than average peers. As a consequence, it knows many potential candidates for its own routing table[12] and the candidate set is expected to be better than a set of arbitrary peers.

3.4.3 Evaluation of PNS

To evaluate Proximity Node Selection (PNS) we simulated PNS on a variant of Chord that relaxes the finger table entries to any node of the appropriate interval. Our evaluation is static and we do not simulate join and leave procedures. However, we used realistic router topologies for the underlay. We were interested in the candidate selection as well as how different measures affect the outcome.

For our simulations we created sets of topologies. As a basis we used different router topologies that were generated by the topology generator Brite[MLMB01, MLMB]. We used Waxman and Barabasi-Albert models to generate the topology. These routers are not the peers in our simulations. Instead we added peers to the edge routers. The assignment was according to the Barabasi-Albert rule ('The rich get richter') to achieve a Power Law distribution of nodes connected to each router. We used from 5 up to 20 routers per 5 up to 20 autonomous systems in the topology. The number of peers ranged from 250 to 16,000. For identical settings (identical topology and parameters) we used 10 simulation runs, and due to computational times 5 or 1 for the large scenarios. We aggregated over the variable parameters, e.g. average delay stretch over all experiments with 1,000 nodes and no PNS.

Question 1: How much does PNS improve the stretch?

We compute the delay stretch over all experiments with appropriate parameters. The delay stretch is computed by comparing the distance between nodes in the network for a large number of nodes. Table 3.2 lists the results.

Table 3.2: $Stretch_{Delay}$ with and without PNS

Number of Peers	250	1000	4000	16000
No PNS	5.43	6.58	7.72	8.85
PNS	3.64	3.86	3.96	4.09

In comparison to [CDHR03], which propagates the use of PNS, we experienced higher stretch values despite taking the optimal from each interval. This can probably be explained by their usage of Pastry instead of Chord, i.e. k-nary tree instead of binary tree. We showed, however, that PNS can significantly improve the stretch. Comparing the numbers we can more than half the distance to the target and, therefore, reduce the overall usage of network resources. This is true for Pastry and modified Chord and applies similarly to other structured Peer-to-Peer systems with some degree of freedom in selecting peers for the routing table. The stretch, however, is far away from the ideal stretch of 1 sketched in the introduction of the section of locality-awareness. This difference seems to be hard to overcome as the overlay routing operates on semantics completely different than the underlay. The knowledge that an ID corresponds to a node in a peer's proximity may only be found on distant nodes.

[12] In Pastry the new peer simply copies certain parts of the routing table of the rendezvous peer and each intermediate peer to a certain corresponding part of its own routing table.

Question 2: What happens if PNS optimizes for hops instead of delay?

Let us remember how PNS operates. PNS measures the delay and selects among all candidates the ones with the least delay. Now, what if we measure the number of underlay hops instead, say by the use of traceroute. The results are shown in Table 3.3. Using the underlay hopcount for optimization is not as good as using the delay. The advantage over no PNS is only small.

Table 3.3: $Stretch_{Delay}$ for PNS optimizing delay as well as underlay hops

Number of Peers	250	1000	4000	16000
No PNS	5.43	6.58	7.72	8.85
PNS on Delay	3.64	3.86	3.96	4.09
PNS on Hops	4.90	5.79	6.77	7.59

Now, we used hopcount to optimize delay and it was less effective. But we can also use the hopcount to optimize the underlay hopcount. Table 3.4 shows the results. The stretch is now computed on the basis of hops in the underlay. The picture is similar, this time PNS on the basis of hops is better. However, PNS on delay is already much better than no PNS, contrary to the results with the PNS on hops for delay.

Table 3.4: $Stretch_{Hop}$ for PNS optimizing delay as well as underlay hops

Number of Peers	250	1000	4000	16000
No PNS	5.00	6.14	7.25	8.38
PNS on Delay	4.24	4.73	5.17	5.57
PNS on Hops	3.54	3.89	4.25	4.58

To conclude the paragraph on this question, PNS on hops optimizes hops and PNS on delay optimizes delay. The other value may also be optimized as a side-effect, say low delay implies that there are not too many hops. It is a general truth that optimizing a parameter optimizes the parameter but not necessarily other parameters.

Question 3: What is the impact of the size of the candidate set?

PNS cannot scan all peers in an interval to find the best. The largest interval is half the Peer-to-Peer overlay. As a consequence, PNS can only use a fraction of the interval as candidates for its decision. The size of the candidate set can be set to a given value k. We select k candidates from each interval, measure the distance and take the closest. Of course, for small intervals with less than k nodes we scan all nodes. The candidates can either be drawn randomly[13] from the interval or k successors can be used (in most systems ID and location should not correlate).

Table 3.5: $Stretch_{Delay}$ for different sizes k of the candidate set

k	0	2	3	4	5	6	7	8	10	20	all
250 peers	5.43	4.40	4.11	3.94	3.82	3.79	3.73	3.69	3.65	3.56	3.54
1000 peers	6.58	5.44	4.97	4.73	4.53	4.44	4.36	4.25	4.20	4.01	3,86
4000 peers	7.72	6.31	5.69	5.33	5.14	4,95	4.85	4.77	4.64	4.35	3,96

[13]Note that random selection may need to handle the selection of a node twice. If this is not prevented, the candidate set for small intervals may not include all possible nodes.

Table 3.5 shows the results. $k = 0$ is without PNS. A small candidate set already improves the locality and most gain is already realized $k = 4$ or $k = 5$. It also shows that for large networks, the candidate set needs to be larger to converge to the optimum value based on all nodes.

Question 4: Are there other side effects?

Yes. PNS with delay optimization prefers links to nodes closer to the virtual core of the network. The indegree distribution of the peers is unbalanced and may cause hotspots. For studies with 4000 peers we obtain the following values. Without PNS, the maximum of the indegree was 79.8 and the 95%-quantile was 32.9. For delay-optimizing PNS we got, maximum 435.7 and 95%-quantile 54.5. Hop-optimizing PNS was not affected and was even better than no PNS. The maximum was 46.3 and the 95%-quantile 22.9. PNS on the basis of IP addresses showed similar results. The better value for the latter PNS variants may be caused by the fact that their selection strategies reduce the impact of large intervals. Large intervals increase the indegree as most likely more nodes link to an ID in the interval.

3.4.4 When do we need Locality Optimization?

Locality optimization is more or less a quality of service issue. If the response time of a system does not matter, locality optimization is not important. If the system only consists of local nodes, locality will not matter. Depending on the measurement methodology locality optimization may also neglect processing and queueing delays. If they are dominant, load balancing and not locality optimization is the key to reduce the response time.

There is also the question where to apply the locality optimization. A valid argument against DHT locality optimization and load balancing is that the lookup is short compared to the overall download. Thus, optimization has to work on the download and not the lookup. This is, of course, the limited view of a particular application. The lookup response time becomes relevant if user interaction depends on its results. In this download example, the user may select the final download, which will happen in the background. The user, however, waits for the lookup result and its response time turns out to be extremely relevant.

Locality optimization and resilience help as well as hurt each other. Locality-aware systems reduce the length of paths and, thus, potential problems with intermediate links. However, it does reduce the diversity of the neighbors, which can also hurt. Security concerns may also recommend diversity, e.g. locality optimization might make the eclipse attack against local nodes easier.

Here are some properties that indicate that locality optimization should be used.

- Nodes may be distributed globally, so that latencies will differ.
- Access technologies slow down the response time of some nodes (e.g. wireless).
- The response time matters, e.g. user interaction waits.

3.5 Relation to Architecture and Design

We summarize the discussion from the previous sections in this chapter in two flow diagrams that help to decide on load balancing and locality-optimization.

A general issue for both and also other ways for the optimization of Peer-to-Peer system is the question, whether to do this Peer-to-Peer or on a server. If a Peer-to-Peer system is

reasonably small and complexity, application, resilience, or privacy issues do not prevent this, a server can optimize with global knowledge and provide best solutions. There is a class of so-called supervised Peer-to-Peer systems where a server manages the network and could perform such operations. If this is not Peer-to-Peer enough, one might select a peer as temporary server. In the following we will give recommendations for pure Peer-to-Peer methods as we discussed in the last sections. However, similar ideas can be used for the server-based approach as well. In particular, the server could also be used within the Peer-to-Peer methods, e.g. to mediate between light and heavily-loaded peers.

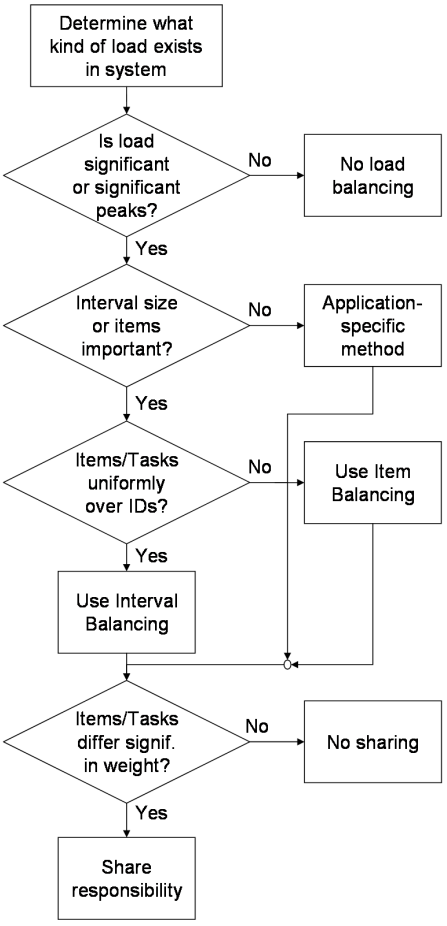


Figure 3.9: Flow diagram for deciding on load balancing for a Peer-to-Peer system.

Figure 3.9 shows the flow for load balancing. The first step to determine what kind of load can exist in the system and its use-cases. This includes resources that will be used and how much they will be used and how this will be distributed (informally). The assumptions of the system should also be stated.

On this we can now answer the questions. The first question (Load Q1) is if there is any load of importance in the system. Even if this is low on average, we should still consider significant peaks that could happen. If there is load like this, we continue. Otherwise, load balancing might not be necessary. We recommend to reconsider and write down the assumptions for this decision.

The next question (Load Q2) is if load is related to interval size or the items within an interval. If yes, we will continue. If no, we need to consider application-specific methods and the last question (Load Q4).

The next question (Load Q3) is about the distribution of items to IDs. If it is uniform, we recommend Interval Balancing, otherwise we recommend Item Balancing. In both cases, we continue to the last question.

The last question (Load Q4) considers the weight of items or tasks. The weight could be caused by complexity of the task or, most likely, its popularity. If there are significant weight differences, we recommend to share responsibility. If it fits to the application, we recommend to simply make the replicas accessible for the normal lookup. If we answer with no, load balancing is not necessary.

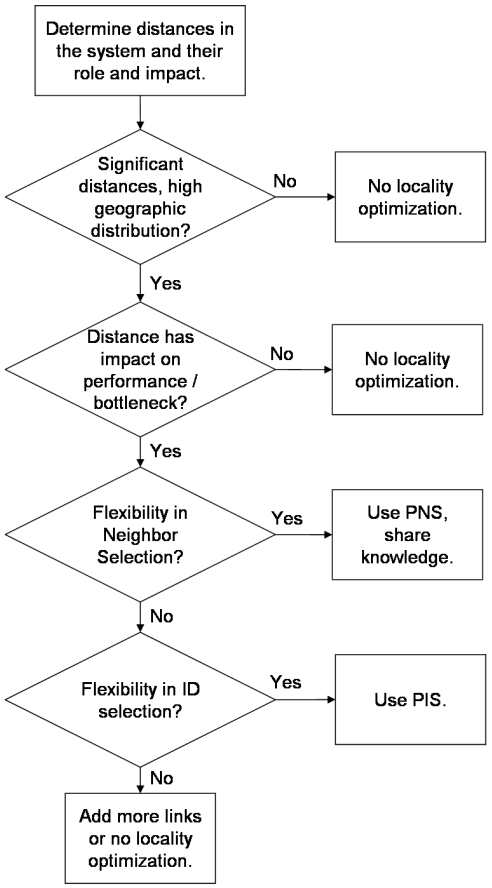


Figure 3.10: Flow diagram for deciding on locality optimization for a Peer-to-Peer system.

Now to the locality optimization. Figure 3.10 shows the flow for locality optimization. The first task is to determine the geographic distribution of the nodes and their potential differences in distance. Determine the role that they may play and how much and where they may affect the system.

The first question (Loc Q1) is about the differences in distance. will there be significant differences or at least the peers are geographically distributed? If no, then locality optimization might not be necessary.

The next question (Loc Q2) asks what impact these differences may have on overall performance and response times. Are they significant or noticeable? If not, then locality optimization might not be necessary.

The next questioon (Loc Q3) is if the neighbors can be selected in a flexible way, i.e. a node has a choice. If yes, we recommend the use of PNS.

The next question (Loc Q4) is about the flexibility to assign IDs. If the system can do with the IDs what it wants and this has no impact on the application and other criteria, then one might use PIS. If not and no other solution has been proposed, we might still be able to add more links or cache paths after a query, etc. In some cases, however, we will not be able to optimize the locality.

In this chapter we discussed methods to optimize Peer-to-Peer systems towards some goals. Not all potential optimization goals are compatible with each other, e.g. robustness and performance may conflict. Optimization also needs flexibility to be able to make decisions and improve the system. We briefly looked at two comparably general goals - load and locatility. As a result, in this last section of the chapter we gave recommendations for the design and architecture.

4. Security and Peer-to-Peer systems

4.1 Introduction

A simple creative experiment - ”What is security?”, we simply asked colleagues. The main points were ”control access to my stuff”, and ”no rag with my PC or communication”.

In her real life a user builds a house and separates her territory from the territory of others. It is not only convenience, but also security and privacy that drive the need for a room, an apartment, or even a house. This is the situation in the human world. And we can find this situation again in the so-called virtual world. Similarly, the user uses a computer or even more a networked computer. The computer is her room and her place to live. And again there is the need to keep her place the way she wants.

Technically speaking, the control over personal data, personal computers, and personal services is an important issue for any Internet user. They all can be categorized under the term security. Security itself is an even broader term, which makes it a bit meaningless without further information. There are a variety of potential security goals and not all of them are fully compatible. Just naming the most prominent ones: Authentication, Access Control, Confidentiality, Integrity, Nonrepudiation, Availability, and Anonymity. Other understandings of security include attack detection, anomaly detection, and system security.

Discussions about security are also meaningless without understanding the threats to a system. The source of danger is the attacker or intruder. The possibilities, interests, and the overall power of the attacker may differ in different models. Security methods that are based on cryptography usually aim at perfect security. This means that an attacker cannot break the method with reasonable effort. Of course, this is limited to scenarios where all the assumptions for the method hold. Yet, they hold in many real-world scenarios. Among the security goals provided by such methods are the classic security goals - authentication, confidentiality, and integrity protection. Typical assumptions include the existence of a trusted party.

Peer-to-Peer systems as well as ad-hoc networks cannot support all typical assumptions, a single trusted party does not fit to their design principles. One may also argue that these assumptions do not fully hold in any real system either, as in the end all entities in a system may socially be peers. Some security goals themselves lack perfect methods due to their definition. Examples are availability and anonymity. Security in these cases is not perfect. A strong attacker could drop all messages to threaten availability.

The Peer-to-Peer paradigm can be described as a shift *from coordination to cooperation*, *from centralization to decentralization*, and *from control to incentives*[StWe05]. Peer-to-Peer systems shift responsibility from the network or a server to end-users and their systems. There are many reasons why this can be beneficial. However, for security there is the question of responsibility and control. The Peer-to-Peer paradigm seems to be good when all peers cooperate, but what if not? What are the opportunities of a malicious node or an outsider? What about implementing fundamental security operations in a Peer-to-Peer manner? We will discuss this in this chapter.

First, we introduce security basics and discuss their realization with a focus on Peer-to-Peer systems. Next, we present and structure Peer-to-Peer-related attacks. These are security issues that are special and typical for Peer-to-Peer systems. One of the main new concepts of Peer-to-Peer systems as sketched in the beginning was that some systems try to replace security enforcement with incentive mechanisms that reward good behaviour. We briefly discuss incentives and reputation systems in general and then we present our Cheapriding attack and a potential attack detection and mitigation mechanism. Finally, we relate the chapter to general design and architecture of Peer-to-Peer systems.

4.2 Security Basics

Security — that is encrypting data to make it unreadable for others. Or is it? There are many misconceptions about security. The corresponding security goal of the example is called confidentiality. But the non-disclosure of private data is only one of many problems and confidentiality is only one of many security services. Another common misconception is that simply using a standard security protocol like TLS on whatever layer in the network stack makes a system secure. While it is true that re-using an existing standard security protocol should be preferred over designing a new one, the protocol will usually not directly solve the security problem of the system. It becomes useful once the prerequisites hold (knowledge and distribution of keys, meaningful identities, security goals). The primary question is therefore to embed the security into the system and then use appropriate protocols.

4.2.1 Goals and Models

Now first to the classic security goals. This includes the major ones, the CIA (Confidentiality, Integrity, and Authentication), as well as some others.

- **Authentication**: An entity claims to have a certain identity. Authentication has the goal to prove this claim. But there are more facets to authentication. Authentication can attribute an identity to an entity (Entity Authentication) on the one hand, but it can also show that data was sent by a particular entity (Data Origin Authentication) on the other. This distinction is a distinction on the use of authentication. Chapter 6 goes more into details. To relate authentication to other security goals, it can be said that the realization of most other security goals needs authentication.
- **Access Control**: An entity is granted access to a resource only if it is authorized. Authorization usually succeeds authentication and it is the process to grant this access. As a consequence, a resource can allow or disallow an entity to use it. In many cases access control is less fine-grained. It is common to connect entities with a role and for the fewer number of roles there are rules stating the access rights. This is called role-based access control.
- **Confidentiality**: Confidentiality is about hiding the content of communication. The goal is to protect data from unauthorized disclosure. Data that was sent to the wrong

entity, even in a protected confidential way, is disclosed by being readable for an unauthorized entity. As a consequence, confidentiality usually needs authentication.

- **Integrity**: Integrity is the goal to ensure that data that was received is exactly the data that was sent. Integrity can be threatened by attackers as well as errors in the network. Since integrity relates to a source and its unchanged data, it usually is combined with authentication. Data origin authentication is a related similar goal (different focus, same methods).

- **Nonrepudiation**: The goal of nonrepudiation is to ensure that an entity cannot deny its actions. Other involved entities can prove to a third party that this entity did something. In cryptographic terms nonrepudiation is to allow this proof to a third party, opposite to only being able to prove this action to each other. The first needs asymmetric cryptography, for the latter symmetric cryptography is sufficient. In real systems, however, an action may be more complex than a single message or a single data item. Keys may become invalid or corrupted. Time and context are then important, which complicates nonrepudation significantly.

- **Availability**: Availability deals with the possibility of a communication or the usage of a service. The goal is to ensure that the service is available for usage. It is threatened by high load and errors, but also by attacks. Attacks that directly attack availability are called denial-of-service (DoS) attacks.

- **Anonymity**: Anonymity is about hiding communication. The communication patterns shall not reveal which entities communicate with each other, neither to themselves nor to third parties observing or attacking the communication. But, anonymity does not try to conceil the fact that there is communication.

For completeness we also mention that the discussion of security also needs the discussion of potential attackers. For cryptographic goals like authentication or confidentiality the most common model is the Dolev-Yao model[DoYa83], which can be put down to the network being the attacker. The Dolev Yao attacker is a rather strong attacker, e.g. being able to intercept or change all messages in the network. It can read, modify, block, or initiate messages at will. However, the attacker is not able to break cryptographic primitives. Yet, it is an extremely strong almost unrealistic attacker. When we do not exactly know what an attacker will do, and this is usually the case, such a strong model is favourable. Many security methods are strong enough to keep this attacker out. They usually operate on assumptions like a trusted party that everyone knows (address, key, etc.) and trusts. Given this setup, the entities can create proofs for the goals like authentication, confidentiality, or integrity by using this trust anchor. For pure Peer-to-Peer systems one may not find such a setting. The basic assumption of the trusted entity for all peers does not hold. As a consequence, the peers cannot perform the proofs and on that basis defeat the Dolev-Yao attacker.

So, for Peer-to-Peer systems the strength of the attacker has to be relaxed. Typical relaxations include a geographical limit for attacker. It can only partially control the network. Another option is to let the attacker only see samples of the messages, limit the attacker to passive attacks or restrict her to only few active attacks. Dolev-Yao model and its variations are important for systems that consider an attacker in the network (no trust for the provider).

More suitable for Peer-to-Peer systems where the network is not the primariy enemy is to use a byzanthine attacker model. Some of the nodes may not operate faithfully and conspire against the rest. Friend or foe cannot be distinguished by the honest nodes. To

be secure against such an attacker the system needs to withstand their conspiracy and make the right decisions, despite a (limited) number of attackers.

4.2.2 Realization of the security goals

There are security goals that rely on cryptographic protocols and means. Some goals, however, are different. Availability is not related to cryptography at all. Anonymity is also different, although it also relies on cryptography as a part of practically all its solutions.

For the security goals that are based on cryptographic protocols, all of them operate on a set of assumptions that a peer believes to hold. From a logic point of view, these assumptions of the peer are axioms. Additional axioms come from the system design and structure, although they could also be seen as axioms of the peer. The purpose of a cryptographic protocol is to find true statements that can be derived from these axioms and other known true statements (either derived within a previous protocol execution or tautologies). So, within the formal system of a cryptographic protocol we cannot generate other knowledge than the one that is already existing in the logic due to the axioms. The consequence of this is that there is no hope to add knowledge from nothing where it does not exist. Section 6.4.3 provides more information on the subject.

Now from abstract to concrete, Alice and Bob want to communicate securely. Authentication is the first step. Alice may know that entity Bob and identity Bob correspond. Furthermore, she knows a key that is the key with Bob (either shared or public key). Bob can prove his identity by applying the key. Alice concludes

'This key $K_{Alice,Bob}$ was applied'
'This key $K_{Alice,Bob}$ belongs to Bob'
'Entity claims to be Bob'
→ 'Entity is Bob'

The basic protocol steps include that Bob signals to Alice which identity he claims to be, Alice challenges this (e.g. sending a new nonce) and Bob has to apply the key (e.g. encrypt the challenge) to answer the challenge. In most cases, an ephemeral (short-lived temporary) shared key is generated for subsequent use. Subsequent use may be confidential communication or access control. If Alice does not know Bob, similar proofs can be made by using a trusted party. If more intermediate parties are needed like in Webs of Trust, Alice may also have an axiom that limits the number of intermediate parties as long paths reduce the confidence into the authentication.

Alice and Bob can achieve confidentiality by encrypting their messages with the shared key. The prerequesite is that the statement that the shared key is the key for Bob and Alice is true. Alice and Bob can conclude that a message sent with this key is only readable for key owners. As a consequence, only Bob or Alice can read the cleartext. Similarly, integrity of the messages can be protected by signing a digest or adding a keyed digest (e.g. HMAC). In practise due to potential attacks the key for confidentiality and the key for integrity will usually be different (e.g. different hashes from the shared key).

Signature with key
& Key is Bob's key
& Signature is correct
→ message from right sender and unaltered
→ correct message.

The realization of the mentioned security goals in a Peer-to-Peer system depends on the realization of authentication. Integrity as well as confidentiality start from the same initial point as in the server-based setting. One may differentiate the different settings on the aspect of trust. Some may argue that one can rely less on a peer not to disclose information. However, depending on the use-case and scenario one may also put less trust into a server.

4.3 Additional Security Aspects of Peer-to-Peer systems

As we have seen already, not all entities in a system may be trustworthy. Therefore, we take into consideration that certain peers may attack the system. Nonetheless, most peers may still not do any harm and behave as reliable as trusted peers would behave. However, many security mechanisms fail once a single attacker or a cooperating group of attackers controls a large fraction of the system. Some may also fail if only one attacker is present. In this section we categorize and describe a variety of attacks against Peer-to-Peer systems. This is an extension of our own work[NWSC04] as well as the most prominent article on the topic by Sit and Morris[SiMo02].

In the following, we discuss fundamental attacks that are rather general. The rest is categorized into internal and external attacks. External attacks are divided into network as attacker and external nodes as attackers. There are some relations between these attacks and their attacker. Internal attacks are from nodes in the same Peer-to-Peer network. It is clear that these nodes can also operate as external nodes. Nodes do not have the power of the network, while the network can also use such attacks (only external node attacks) by running an entity as node.

4.3.1 Fundamental attacks

In this section we present fundamental attacks against Peer-to-Peer systems that cannot be sufficiently discussed as internal or external attack only.

Bootstrap Attacks

The situation of the bootstrapping is a very critical situation. The new peer is not yet part of the Peer-to-Peer system. It needs to find an entry-point to the Peer-to-Peer system and perform the join operation.

The first problem is to find the network. A peer that joins a system may wonder if it joins the right system. This could be threatened by a fake system that malicious nodes have set up. This is primarily an issue for the bootstrapping. However, as we will see, there are also attacks to mislead a node from the correct system into a malicious one. When a peer communicates with another peer it may want to ensure that it communicates with the correct other peer or the correct service for its request. In a data-centric view, one may say that it wants to ensure to receive and work on correct data.

The next problem is to join via a correct and honest rendezvous peer. If the new peer joins via the attacker the attacker can mislead the node into a malicious network or to communicate to a large degree via nodes of the attacker.

The defense against bootstrap attacks is not easy, in particular for open networks. Many networks may solve this issue with acceptable quality by having a set of trusted rendezvous peers that the software from the standard website of the system knows. However, fake sites like Phishing sites may threaten this measure and reduce its effectiveness.

Sybil Attack

The Sybil attack has its name from a schizophrenic person in a novel that had more than 20 personalities. The term actually dates back even further to ancient greece. Sybils were prophetesses, e.g. at the oracle in Delphi. From time to time not the human person of the Sybil but a God was speaking through her lips, which can be seen as a temporary shift in personality.

The idea in the network is similar. An attacker performs the Sybil attack[Douc02] when it joins the Peer-to-Peer network with multiple identities. The basic idea of the attack is one entity with multiple identities in the network.

A defense against the Sybil attack is not easy in open networks. There is no convincing solution. Proof-of-work tests do not work and waste resources. The basic argument is that an attacker with a powerful computer can simulate the work for multiple weak computers of other users. There are also protocol issues, i.e. that there is no limit when the attacker does not need to have the resources for its Sybil nodes ready all the time. A one-time check is not sufficient[Douc02].

Social networks and human interaction can help to mitigate the Sybil attack. One can exploit their structure to differentiate between a social graph between multiple humans and the graph of a group of sybils[YKGF06]. Nodes may bootstrap via nodes they can trust and may need to persuade them in order to join. Given the initiator and early members are trusted, then the bootstrap graph can be used to route around a sybil attacker by combining iterative routing, routing in the bootstrap graph and routing in the network graph[DLLKA05]. Binding the identity to an unforgeable external identity is another option to counter the Sybil attack, e.g. by enforcing $ID = hash(IP)$. However, one may need to allow multiple IDs per IP due to NAT and optimization[DiHa06]. While these solutions may help to fight to some degree, the only secure solution without restrictions is to use a central authority and a closed Peer-to-Peer network.

Eclipse Attack

The Eclipse attack[SNDW06, SCDR04] can directly follow an attack on the bootstrapping. In the Eclipse attack the attacker positions its nodes between the eclipsed victim and the real Peer-to-Peer network. All (or most) communication between victim and Peer-to-Peer network passes the attacker. Usually, the victim of an Eclipse attack is a node or set of nodes. There is also a notion of the Eclipse attack that eclipses items. In many Peer-to-Peer systems, this is trivial as an attacker may position itself accordingly and delete the item. For networks like KAD, however, the item eclipse is non-trivial[StENB07, Stei08, KoLR09]. This is due to the operation of the Kademlia KBR, replication and caching of items as well as no strict definition of responsibility. The latter means that any node in an interval around the item ID can be responsible for the item and it is up to the source to store and refresh the item on enough nodes.

The Eclipse attack is not only a threat for the bootstrapping process, it can be staged with the maintenance mechanisms of the network as well. The attacker then propagates its own nodes as good candidates for the outbound links of the victim to certain parts of the network as well as on links towards the victim. The Sybil attack can be used by the attacker to get enough nodes in the Peer-to-Peer network. The attacker can also introduce a chain of fake nodes to mislead lookups in case of iterative lookup processing.

The Eclipse attack can be fought by using trusted entities for the bootstrap. Churn can be beneficial in that case[CKSH$^+$06]. If a peer is in fear of being eclipsed, it may rejoin via a trusted entity. Peers need to rejoin often enough to fight the attack. Other proposals or recommendations to fight the Eclipse attack are less promising. An attacker that uses

fake routing updates may increase its in-degree for the Eclipse attack. However, this is not true when the Sybil attack is used and yet there is also the problem that honest nodes may have a high in-degree due to their position or due to optimization. Another recommendation is to prefer strict routing systems like Chord (for each routing entry only one peer is correct) over flexible routing like Pastry or Kademlia (many peers are equally-good as routing entries). This is only valid if the Sybil attack is prevented or, at least, the sybil nodes cannot obtain necessary IDs. The Sybil attack helps to circumvent the strict routes and lets the attacker position its nodes. Even more fatal, the nodes of the attacker become correct routing entries the victim must use. Flexible routing on the other hand allows the use of strategies like in Kademlia where old routing entries are preferred and will not be replaced with new nodes unless necessary. As a consequence, attacker nodes need more time to make it into the routing table of their victim. Another assumption that is frequently used when considering a distributed Peer-to-Peer attacker is that its nodes and their Sybils are located in similar geographic and network topological regions. KAD started to limit the number of nodes in a bucket from one IP subnetwork as a defense against the first item eclipse attacks. However, this measure can be circumvented with a good positioning of attacker nodes and the introduction of a chain of fake nodes once an attacker node is hit by a query. The attacker misleads the query as its fake nodes are always the best next hops. One could counter this by introducing IP diversity also into the selection of next hops similar to the Sybil-resistant routing that uses the bootstrap graph to route around attacker nodes[DLLKA05].

Weaknesses of the local machine or user

In a Peer-to-Peer system the machines of users take over the role of a server and store relevant data. Other software (also the operating system) as well as the user herself (on multiuser machines also administrator and with restrictions other users) can attack the Peer-to-Peer system.

They may observe the keyboard (logging), read the files stored on disc or the messages sent and received. With appropriate knowledge they may modify data or use exploits. Human users may use social engineering, maybe on the basis of the system, e.g. chat functionality.

Storage attacks from local attackers can be mitigated with redundancy (compare data with replicas) and encryption (cleartext data unreadable to local host). It is a reasonable assumption that local attackers do not know each other and do not cooperate. This may be different once the attacker is a worm that attacks the system.

4.3.2 Attacks from external entities

Peer-to-Peer systems do not operate in a separated environment with no contact to external entities. The peers use the Internet and, thus, the network can interfere with the system. As a Peer-to-Peer system is far from consisting of all nodes of the Internet, most nodes in the network are not part of the system. Furthermore, while these external nodes may not be directly involved with the Peer-to-Peer system, they can still launch attacks.

Attacks from the network

Peers are connected to each other via the underlying network and the lower layers in the protocol stack. The network is a rather strong attacker and it corresponds to the attacker being a network provider or anyone that can force them to comply, e.g. law enforcement, but also a selfish employee.

Here, we list some attacks that can be performed by the network. Of course, it can also operate as external node or maybe even as internal node.

- **Eavesdrop**: Observe the network's traffic to obtain information.
- **Drop messages**: The network can block the communication of the peer.
- **Modify messages**: The network can modify messages sent from and to the peer. It may not have the knowledge to do this in a way that defeats cryptographic integrity checking, but it may try to obtain the key like any attacker by other means hoping for weaknesses in the protocol or algorithms.
- **Delay messages**: The network may delay messages, so that they arrive when the network wants them to arrive.
- **Replay messages**: The network may replay messages. This could be used to attack weak protocols or to trigger reactions.
- **Interfere with the bootstrapping**: The attacker could intercept bootstrap messages, either preventing the bootstrap or offering itself as a fake rendezvous node. This attack is useful for subsequent attacks, e.g. Eclipse attack.
- **Attack privacy of node**: The network may use its means to find out about the node's Peer-to-Peer systems, its identities, and the purpose of the system. Some of the information may be found clear text in the messages, other may be infered from use of ports, message flow or even patterns in the communication.
- **Attack privacy of P2P network**: The network may find out about the purpose and operations of the Peer-to-Peer system. It may analyze message flow and communication patterns to do so. It may

In contrast to the Dolev-Yao attacker model, the attacker is not necessarily the whole network, but the fraction that is controlled by the attacker. This attacker is still extremely powerful and the attackers in the subsequent sections are weaker. Other nodes do not have the power to easily interfere with another peer's communication. There are some options, especially in local networks, like spoofing or ARP poisoning.

If trust exists some of the attacks of the network can be resolved. The basic question is how to initially setup this trust. Keying material in the application is one solution, if one assumes that the network as attacker is not powerful enough to interfere, obtain, or change the keys during download or obtain them via analysis. A web-of-trust may also help. However, the basic argument to defeat the network today is that the network will not do complex operations as it is busy with its core operation of transporting traffic and maintaining its status. For low-value-targets this may hold. The major threat today in the network is the ubiquituous logging of communication relations.

Attacks from external nodes

External nodes are hosts on the Internet that do not participate in the Peer-to-Peer system. In a closed Peer-to-Peer system they may not be able to join. In many open systems they might easily join to additionally be capable of mounting internal attacks. In addition to automated attacks, human users run the attack, guide the mechanisms, and research real-world (Social Engineering) or search the Internet.

- **Use the Peer-to-Peer network**: The external node may use the service provided by the Peer-to-Peer system without participating in it. If messages from external nodes cannot be distinguished from messages coming from external nodes, this attack is possible. In systems like BitTorrent with its tit-for-tat-driven service, the attacker

also has to give to gain and, thus, there is no direct damage[1]. In general, only nodes of the Peer-to-Peer system should be able to use it. Using it means looking up data, reading messages that are exchanged, sending routing updates, etc.

- **Pretend to be internal node**: An external node may pretend to an internal node that it is an internal node.
- **Become illegitimate member**: An external node may become a member of a closed Peer-to-Peer network by overcoming its access control.
- **Obtain internal information**: An external node may communicate with internal nodes and obtain internal information that was not meant to be known by external nodes.
- **Interfere with bootstrapping**: The external node may mislead new nodes that want to join the Peer-to-Peer system to itself. This may be done by setting up a fake website with wrong rendezvous points.
- **Exploit weakness**: The external attacker may send prepared messages to nodes that may run an instance of a erroneous version of the software. The prepared message will harm the victim in some way.
- **Use of public information**: Use public information about the Peer-to-Peer network to identify members and to obtain information. There are systems that post status information on the Internet.
- **Attack privacy of peer**: The attacker may find out about the used application by scanning the peer's ports or by sending fake requests to potential ports of the Peer-to-Peer system.
- **Attack privacy of P2P network**: The attacker may find out about other members and the purpose of the Peer-to-Peer system. This is not as easy for a node as for it is for the network, but the attacker may be successful.

Peer-to-Peer systems without explicit security mechanisms might hope that an external attacker may not be able to use this fact. The ports may be unknown, as well as the protocols of the Peer-to-Peer network and its formats. The likelihood is also low that a machine with an arbitrary IP address is part of the Peer-to-Peer network. Both issues make that the attacker cannot easily connect to a node of the Peer-to-Peer network.

Methods for authentication and authorization are the standard solution for closed networks. If they are centralized (CA or trusted party in network), one can keep external nodes out. In case of decentralized methods, there may be more attack vectors that an attacker could use to overcome them.

If one can trust the infrastructure, one may exclude spoofing and attacks where communication has to pass the attacker. Unless bootstrapping needs talkative nodes[2], a hidden or silent operational mode should be used, e.g. not replying to strange messages, no pinging, etc.

A common answer given in real Peer-to-Peer networks is to tackle the problem from the opposite side. External nodes may not matter, but all nodes are considered untrustworthy as if they were external. The operation of the network can live with this or otherwise the attackers will be recognized by the legitimate nodes. Incentive mechanisms are one answer if applicable.

[1] Unless the attacker is the music industry in illegal filesharing and the goal is to take legal actions.

[2] Spontaneous network creation and ad-hoc scenarios may use local broadcast mechanisms to find other nodes. A common scenario is a meeting where the participants connect their devices.

4.3.3 Attacks from internal nodes

We classify the internal attacks similar to [SiMo02]. The classes are Denial of Service, Storage and Retrieval, Routing and Topology. Additionally, we add the class of attacks against security mechanisms.

Denial-of-Service Attacks

Denial-of-Service (DoS) attacks are destructive attacks. The attacker is not gaining anything directly. The performance of a node or the overall system is degraded. Nodes in a Peer-to-Peer systems are more prone to Denial-of-Service attacks than the network core as peers and end-user machines are often connected to the Internet via asymmetric connections, i.e. with low upstream and high downstream bandwidth. Peer-to-Peer system have a higher need for upstream bandwidth than most client-server communication. There is one bandwidth invariant, over the complete system utilized upstream equals the utilized downstream. Compared to other possible data rates, the bandwidth of peers is usually low (16 Mbps DSL vs Gigabit Ethernet).

From the asymmetric link properties, we can see that one bit out for each bit in may cause a congested upload. Most systems, however, are not that symmetric when all traffic in the network is taken into consideration. A simple request in a DHT travels $O(logn)$ hops on average. Thus, one message causes $O(logn)$ messages in the DHT. In case of recursive routing, these messages are all sent and received by other nodes than the initiator. The network has to work a lot to process one single request. This effect in general is called amplification.

DoS attacks can also be performed distributedly by a group of attackers. This is then called a Distributed Denial-of-Service (DDoS) attack. The common DDoS scenario today is a botnet with up to millions of infected machines attacking a certain entity in the network (server, router, peer, etc.). Botnets are usually connected via IRC, but recently also Peer-to-Peer botnets were reported. So, Peer-to-Peer systems may not only be a victim, but could also be exploited to perform or organize a DDoS attack as well. With respect to defending DDoS, Peer-to-Peer systems may be an answer in cases when failures of nodes due to DDoS can be circumvented. This is not only a question of having a node up that could perform the service as well. It also needs to be found[3].

- **Overload a target node or network**: This is the common goal for DoS attacks, to overload an entitiy. There are various ways how this can be done. In the DHT example, this could be done by sending useless queries. So, the attacker requests a particular ID. In return, she receives the IP address of the node responsible for the ID. That alone is not much work and the number of requests needs to be high to have an impact. However, one could also implement the system so that data is returned. In such a case Denial of Service attacks are more efficient. In particular, an attacker may also fake its own IP address or ID, so the traffic of the reply may hit another node. This other node can be efficiently attacked by sending a lot of different queries with this node as fake source. However, if the node checks if it requested the data it can close the connection and avoid most of the traffic.

[3]Proponents of Peer-to-Peer systems may sometimes neglect the latter. It is not uncommon that the uptime of the Peer-to-Peer system (at least one node is up) is considered to be the uptime of the service. This is far from being true. First, even in normal operation requests may get lost due to churn. Second, considering it a replacement for a server farm, there are clients that try to access the service. Now, how should they connect to the unknown node that is up? No matter if they join or simply request the service, they will do this on the basis of a single or few rendezvous entities that they know a-priori or via machanisms like DNS. If they fail, the existence of other nodes does not help.

- **Fake messages**: The attacker send faked messages like wrong replies or data.
- **Modify messages**: The attacker modifies a message in a systems with recursive routing in order to sabotage the lookup or service.
- **Dummy queries**: The attacker floods the network with dummy queries.
- **Unsolicitated queries**: The attacker sends queries with other nodes as source. The reply goes to the wrong peer. In typical DHTs each query for an ID has a complexity of $O(\log n)$ messages. Since the topology of the Distributed Hash Table and the underlay are different, the cost in network traffic can be high, even with only few messages.
- **Churn**: Rapid Joins and Leaves attack a Peer-to-Peer system as a whole. Fingertables have to be updated and data has to be replicated after a node joined or left the system. If the system did not replicate or failed to replicate the data data may be lost. Even more efficient is the attack when the attacker is not a node leaving the network, but a node convincing other nodes that a particular node is leaving. In this case the attacker avoids to be burdened with routing traffic and data movement. Delayed routing updates and data movement can mitigate the effects of the attack.
- **Use peers for DDoS**: The attacker may trick other peers into communication with a victim peer. The distributed peers do not have the intention to harm the victim and are not doing something illegitimate. However, their number may overload the victim. This threat is especially important for systems where peers can perform resource-intense operations for other peers, e.g. measuring bandwidth between B and C on behalf of A.

Denial-of-Service attacks are hard to prevent. The basic idea for prevention is to force the attacker to do more work and, as consequence, reduce the amplification factor. One example is TCP where the server (= node that is contacted via TCP) does not keep full state for the connection until the handshake has finished. Thus, the attacker has to interact, which it did not do for the classic SYN flood attacks.

The basic idea to counter DoS attacks is replication and diversity. If one peer cannot fulfill its task, another one does it or already did it. In order to be able to enable various peers to serve a request, data should be distributed and one should not rely on one peer to perform long operations or the peer being required to be up until the complete task is finished. BitTorrent is a good example. The file is divided into many small chunks. Peers exchange the chunks themselves and only need the tracker to find other peers. The swarm may not immediately die if the tracker is down as peers already know other peers. As peers have many different chunks, a single failure does not hurt.

Storage and Retrieval Attacks

In a Peer-to-Peer system a query passes many intermediate nodes. A node that is responsible for storing data items can attack the data in many ways. A node that routes a query can interfere with the retrieval.

- **Fake Items**: The attacker can create fake existing items. Fake items can be used to waste storage resources, disturb the lookup, and mislead the victim.
- **Deny Item**: The attacker can deny the existence of data with a particular ID even though it exists.

- **Modify Item**: The attacker can modify items and data. She is not necessarily limited to items she is responsible for. If stored items are not protected, the attacker may modify any item in the system, e.g. by updating or storing the item.
- **Item Poisoning**: The attacker may mislead a victim to a poisoned item that contains fatal links, a virus, a worm, degraded data (e.g. noise instead of music). The goal is to harm, disturb, or catch the victim. Another goal may be to disturb the system as a whole (e.g. users are annoyed by the polution and leave).
- **Create non-existing fake data**: The attacker may invent data and store it in the system. This can be used to waste storage resources, disturb the lookup, and mislead victims.
- **Reply with wrong answer**: The attacker may answer requests with the wrong content. The request may not necessarily be for the attacker or its services and items.
- **Profile nodes**: Nodes that transport messages or provide a service can create profils about other nodes. The general use is not completely clear. In filesharing systems, query strings or requests for data chunks may indicate illegal downloads if the item is shared illegally.
- **Confidentiality of data**: Nodes that store and transport data can read the data and might also be able to interpret the content. The confidentiality of the data may then be broken.

One possible defense against such attacks is to make data useless and unalterable for other peers than the legitimate ones. The data is encrypted and integrity-protected with keys only known to the legitimate peers. This suggestion does not prevent destructive attacks, but may help with the others. The basic problem is that such a solution assumes the existence of restricted groups with pre-shared keys. The groups cannot find themselves on the basis of the Peer-to-Peer system. So, while some use-cases exist, they are limited.

Apart from the application of cryptography, replication is one solution. It does, however, not help when the attacker can easily replace the replicas or take over the replication. Write-access on data and replicas needs to be protected, e.g. by a secret or the key of the owner. The owner may be the first who stores data with the ID. The second issue is that the sybil attack needs to be prevented, so that the attacker cannot control the majority of replicas.

If the data is large in size, it cannot be replicated to a lot of replicas. Here, fingerprints may help. The owner of the data may store them at a lot of places instead or even flood them, if system size and load allow that. To avoid the storage of out-of-date items, they may be stored softstate and disappear after some time.

Some of these attacks work because knowledge about the data is lacking. Context may be useful to detect fake data and defeat the attacks. The application knows what it expects and what the data might look like. It can check if the new data makes sense and otherwise discard it. Here, the network may leave the job to the application to secure the system against the attacks.

Routing and Topology Attacks

In this section we discuss attacks on routing and topology. The attacker may take over parts of the network and interfere with the routing tables. We assume that an attacker can either select its ID at will or within some degree of freedom. This is true for most DHTs,

where the new node chooses its ID. There is a recommendation that the ID is a hash over the IP address, but this is not enforced and would cause problems with Network Address Translation (NAT). We also assume that the attacker may control more than one node. Methods to achieve this are the Sybil attack or simply cooperation between attackers.

- **Route table poisoning**: One goal of the attacker can be to poison the routing table of a peer. The poisioneous entries link to the attacker, instead of an honest peer. The attacker can use this to interfere with storage and retrieval, but also for the routing. This attack is related to the Eclipse attack.
- **Eclipse attack**: An attacker positions herself between a node (or set of nodes) and the rest of the network, so that most to all communication and information goes via the attacker. The attacker eclipses the node from the rest just as the moon eclipses the sun. See also section 4.3.1.
- **Attack routing update mechanisms**: Routing update mechanisms are not secured in typical Peer-to-Peer systems. The attacker may send fake information that changes the routing entries or triggers an update.
- **Freeriding on routing**: A lazy peer reduces its in-degree by attacking the routing update mechanisms. Optimization methods like Proximity Node Selection (PNS) can be useful to perform the attack, e.g. by delay replies.
- **Partition the network**: An attacker may try to partition the network, so that nodes from each partition cannot communicate with nodes from other partitions.
- **Attack an ID**: An attacker may be interested to attack the data at a particular ID. There can be many reasons for such an attack, most likely the attacker does not want others to be able to read the data. The easiest way to do this is to select the node ID in a way that the attacker is responsible for the data (in Chord e.g. an ID slightly lower than the ID of the node currently controlling the data). She can then delete the data.
- **Attack a node serving an ID or data**: An attacker may also attack a particular node. This can either be for Denial-of-Service purposes or for eavesdropping and traffic analysis, e.g. for proving that the node is hosting illegal data. In Chord, the attacker should position herself directly before the other node. All queries pass the predecessor of the node responsible (usually this node will reply with the IP address of the responsible node). If the attacker cannot select its ID with enough degree of freedom, it can still try to come as close as possible. In general, the closer the attacker is to the node the more likely it is that a query will pass it.
- **Attack a node requesting an ID or data**: An attacker may also attack a particular node that requests data, maybe to sabotage it or simply to eavesdrop what data the node is requesting. Since the structure of the routing table is known, the attacker can position her nodes directly on positions on the identifier space where the routing table of the node under attack has to point to. Considering a uniform distribution of requests, poisoning a long distance link is more profitable than short distance links. Example: Under this assumption roughly 50 % of the requests in Chord use the longest link in the finger table (link half way around the circular identifier space) as first hop link.
  A measure to reduce the success probability of such an attack is to avoid finger table entries with similar IP addresses (assuming that the nodes of an attacker are from one subnet, administrative domain, or region). This measure requires topologies that allow different paths.

- **Attack a particular path**: An attacker may also attack a particular path. Any position in the system that is on the path is ok for the attacker. When it has no direct contact with any of the endpoints of the path it is less likely that the attack is discovered. However, randomizing the first hop renders this attack useless as there is no definite path. Randomizing the first hop does not change the complexity of the lookup in Chord. However, some topologies of Distributed Hash Tables (e.g. Butterfly) do not allow alternative paths and randomization is not possible.
- **Black hole attack**: The attacker tries to attract other nodes to herself, so that its in-degree is high or atleast the desired victims send most of their messages on paths with the attacker. She then drops messages to disturb the network.
- **Flooding**: In case of iterative routing, the attacker returns only other attackers as potential next hops and invents IDs. The message will not reach the target as long as the attacker can invent IDs that seem closer and a good choice. Of course it can also only return other nodes that do not have expected IDs, but are only closer. That, however, may lead to suspicion. No matter which variant is taken, a flood of messages is generated.

An obvious, but only partial, solution to the some of the problems above is not to let a node decide its ID. In the Chord paper[SMKK+01] the authors propose to compute the ID of a node using a hash of its IP address. While this was not meant as a security measure it could be a solution for the security problems mentioned above. However, if we use this approach it has to be enforced by the Distributed Hash Table and the relation IP address and node ID has to be checked whenever nodes communicate. Otherwise, it would be useless and nodes could change their ID afterwards. Another drawback of this solution is that it fails when nodes are behind Network Address Translation (NAT). If the application is not aware of the NAT it will use its local IP address which is not the IP address visible for the other nodes. This is also closely related to fighting the Sybil attack. In the Sybil attack context Dinger and Hartenstein[DiHa06] proposed to allow a fixed number of nodes per IP as a compromise. The benefit of this approach is that an attacker is limited to IP addresses it can use and cannot position nodes at will. A drawback is that this limitation becomes obsolete with the large IPv6 address space once IPv6 replaces IPv4.

In general, nodes cannot be easily positioned in the Peer-to-Peer system when the system proposes and enforces constraints that can be verified by any node and that put a real limit to an attacker. As a consequence, the attacker may be limited to only few places in the system, which are usually not the places at which the attacker wants to be to run these attacks.

Another idea against such attacks is to keep the network in constant change, e.g. [CKSH+06]. Nodes may profit from trusted nodes and their routing tables when they rejoin via them. Furthermore, as an attacker needs some time to stage a successful attack, its target may have shifted before completion of the attack. However, such a method can also fail and help the attacker when the churn may help the attacker to become a majority in an interesting region. If items are the target of the attacker, the items also need to change. Another potential problem is that in such a scenario it may be sufficient that the attacker may be in the right position for a short period of time, e.g. to delete an item in case of a storage attack.

Still, there is a danger that the attacker succeeds. And while in large networks the attacker may not necessarily succeed against a particular victim, the attacker may succeed to be in such a position for a random victim. To evaluate the NodeID assignment situation we developed a simple model. We assume that the attacker cannot position itself in the

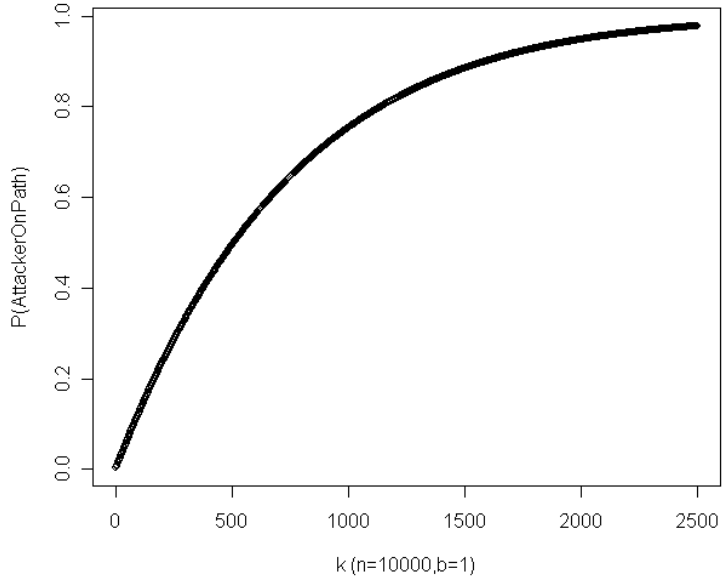


Figure 4.1: P("Attacking node on Path") with $n = 10,000$ and $b = 1$.

system at will, but is strictly limited to the position enforced by the constraint. However, we assume that the attacker controls a nodes with n nodes being in the Peer-to-Peer system. The consequence of these assumptions is that the attacking nodes are randomly distributed over the ID space. For large numbers the density of attackers within a part of the system converges to the overall density. This means we do not model it as combinations without repetitions. Each next hop is in a completely different segment, a segment with a similar density. We conclude that therefore for any next hop the probability that the next hop is an attacker is $\frac{a}{n}$. Another simplifying assumption is that we set the number of hops that a message travels to $b\,ld(n)$ hops, using the $O(log(n))$ property of path lengths in hops.

$$P(\text{"}Attacking\,node\,on\,Path\text{"}) = 1 - \left(\tfrac{n-a}{n}\right)^{b\,ld(n)}$$

Figure 4.1 shows how a stronger attacker increases her chances. Even with only $20 - 25\%$ of the nodes, it is almost certain for a communication to go via at least one attacker hop. In Figure 4.2 we kept the fraction of attackers constant and increased network size. As probabilities are fixed, the $O(log\,n)$ lookup path dominates the results.

A real attacker that has some degree of freedom to position herself will not position herself randomly, but maximize the probability to be on a path it wants to attack. A model like the one above may describe how many possible places an attacker needs to position herself on a particular path with a given probability. If the network size is small, the attacker may easily use its degree of freedom to get into the right position. Networks should not only rely on this being hard. If such attacks are important, one may need to consider closed networks or reduce the impact of such attacks.

Attacks against Security Mechanisms

Peer-to-Peer systems may use security mechanisms. Mechanisms that fit quite well with the Peer-to-Peer paradigm are distributed decisions and reputation systems. It is clear

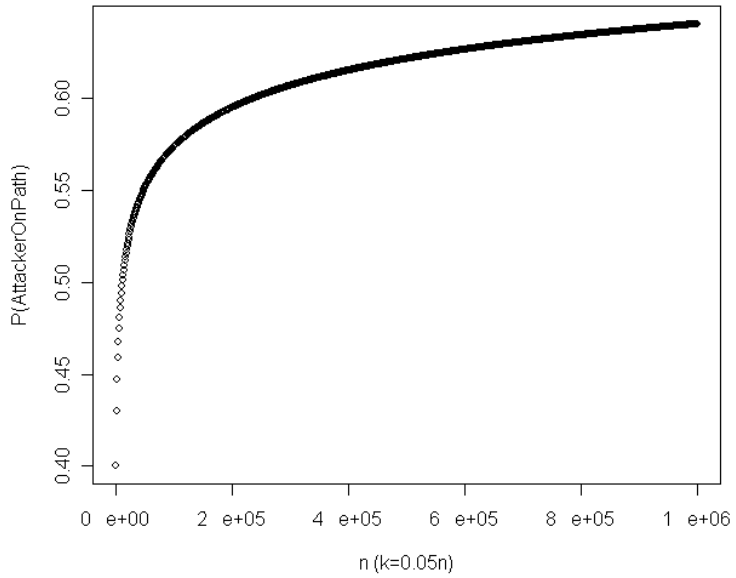


Figure 4.2: P("Attacking node on Path") with $\frac{k}{n} = 0.05$ fixed.

that such mechanisms are prone to attacks by strong attackers. Nonetheless, even systems with central authority may have weaknesses, in particular for being usable. A common example is email where an attacker only needs to know the date of birth, address, or the name of her pet to attack the account of a user.

- **Attack distributed majority voting**: Distributed decisions can be attacked and their outcome changed with enough sybil nodes or cooperation of a group of attackers.
- **Undermine incentive mechanisms and reputation systems**: If the consequences are not severe enough, nodes may simply freeride. If there are consequences, one may use them tricky enough to get the most out of it. Cheapriding is our variant of freeriding, where a node only cooperates as much as not to face consequences. Some misbehavior has to be tolerated due to failures. The attacker uses this tolerance. Within reputations systems, misbehaving nodes may whitewash by leaving and rejoining the network with a new identity. Reputation systems may also suffer from strategy changes. The basic idea is to build trust and then misuse it for a valuable transaction.
- **Wrong identification**: An attacker may use a wrong identifier or the ID of another node.
- **False accusation**: Similar to wrong IDs, the attacker may falsely accuse nodes of malicious behavior. A very good attack is one where the attacker makes another node look to be the attacker and honest nodes will falsely accuse it.

The weaknesses listed in this section usually operate on a limitation of knowledge about other peers. One is not able to distinguish honest and dishonest nodes. The more restricted a network is the more likely it will not require operations that are prone to such attacks. Closed networks may keep the attacker out, while open networks need to react to and

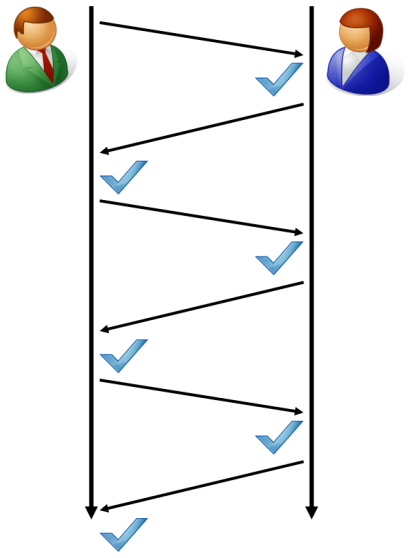


Figure 4.3: Tit-for-Tat

mitigate attacks. Yet there is a limit for achievable control and potential restrictions, it is the application itself. Online auctions, even with server, require the peers themselves to exchange goods and money. These operations are not under the control of the server and it does not know what happened and who is right in case of complaints. We conclude, some of these issues are a consequence of the application and can hardly be avoided.

4.4 Incentives as alternative to control

Trust is usually lacking in Peer-to-Peer systems as no single entity has control and is responsible. There may be trust in the software or between entities that have established trust relations. If there is no trust, one may try to establish trust on the basis of reputation or live with it. If control is lacking, providing incentives for correct behavior is another option. Incentives will not stop a determined attacker, but they can be used to protect some core functions of the system, i.e. an attacker can only achieve long interaction with a node when she sends correct data in a file distribution system, thus, supporting the system instead of attacking its operations.

This last example makes sense in systems where peers directly interact and can adapt their behaviour according to this experience. The simple and yet important basic mechanism is *Tit-for-Tat*. A peer provides a good service to another peer if the other peer does this to her as well and degrades the service when the other peer misbehaves. Figure 4.3 shows such an interaction, as both peers cooperate they continue to serve each other. Tit-for-Tat is the method that provides the security in that scenario. However, usually the use of Tit-for-Tat requires adaptation and optimization. Unknown nodes need to be able to start an interaction. One also needs to start and give before one can evaluate the behaviour of others. Moreover, new nodes may need the service of another node more than they can currently provide to the other node in return. So, some of the resources need to be provided without being sure of getting a return.

Direct incentive mechanisms can be applied when we do not trust other entities and where we want only local decisions. There needs to be mutual interaction, so that both sides can adapt the service according to the behaviour of the other. Of course, the behaviour needs to be observable and a direct evaluation needs to be possible. Peers in BitTorrent can do

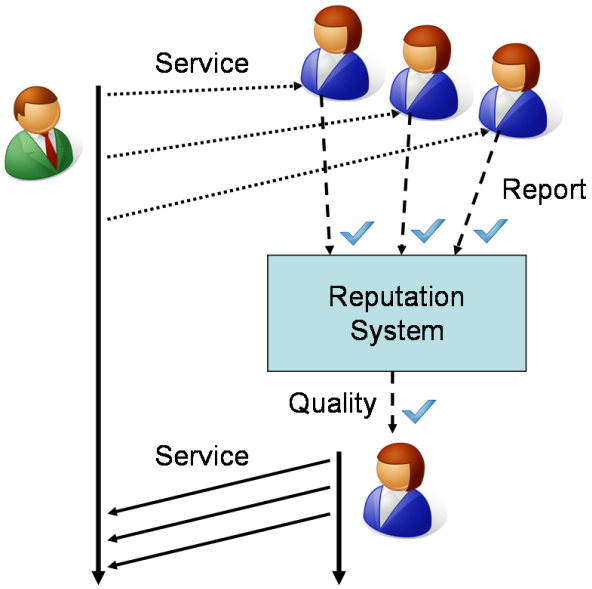


Figure 4.4: Reputation System as Incentive

Figure 4.5: An auction site like Ebay cannot observe the mutual interaction of the peers Alice and Bob.

this because they know the hash values of the chunks. If not, they would not know if data was correct before completing the whole file or even not before user evaluation.

A different approach is to use a reputation system. This can either be centralized on a server or distributed among all peers. There are various ways how they may operate. The basic idea is to determine the reputation of the other peer and then provide better service for peers with good reputation and no service or a less quality service for other peers. Figure 4.4 visualizes this concept. The reputation is computed from reports of previous interactions. The experience of other peers is taken into account, contrary to the purely local decision of Tit-for-Tat incentives. There are some issues. Securing a reputation system is not trivial. Nodes may report falsly. Traceable identities are a problem. New nodes may either have a bad start with low-quality service or whitewashing (re-join as new node with neutral reputation) for dishonest nodes is easy.

An example for a server-based reputation system is the Ebay auction site on the Internet. Users may offer items for sale. Let us call them sellers. A seller describes her item for the auction and uploads photos of the item. She does this herself and only she has access to the item. So, Ebay cannot directly control the product description for the auctions. The

buyer who won an auction needs to transfer the money and behave correctly after receiving the item. She could deny this and issue a complaint despite everything being in order. It is not uncommon that misunderstandings occur. Here again, this is not under Ebay's control. Similarly, the seller could misbehave. As money is usually tranferred first, the buyer needs to trust the seller that she will ship the item after payment. Figure 4.5 shows the dilemma of an Internet auction service, it can only see the participating entities, but it cannot observe what they do and what really happens. Thus, Ebay needs a way to control the process and enforce good behaviour. One of the keys is to use the reputation system. In the original one, buyer and seller rate each other after the transfer and the handling of potential complaints. In the current version, only the buyer can fully rate the seller. The seller can only give positive ratings. The reason for this change was that Ebay feared that buyers do not rate negatively as they might fear a negative rating in return, which may hurt their sales as seller. The simple assumption is that the trust into the seller is more important and that, in contrast to the buyer, the seller can really profit from misbehavior.

The reputation for an Ebay user is then $Sum_{positive\_ratings} - Sum_{negative\_ratings}$, together with the percentage of positive ratings[4]. The incentive behind the cooperation is that buyers will more likely bid for auctions from sellers with high reputation, in particular for items of high value. Other incentives are getting excluded from the system after enough misbehavior as well as legal consequences of severe misbehavior.

The Ebay example is a centralized approach with all reports evaluated at the server. There are also approaches to distribute the reputation system and compute the reputation in a distributed manner. The basic idea there is to exchange reputation values with neighbors and to weigh the reports of peers according to their own reputation. Maximum Likelihood Estimation [DeAb04] and Eigentrust [Ka$^+$ot03] are examples.

A reputation system as incentive mechanism to achieve cooperation among peers is a common choice for applications if the following points hold. First, there are no direct longterm mutual interactions where a simple tit-for-tat strategy could be used. Another reason might be that one may need to know a reputation (prediction of behaviour) before starting or accepting a service. Second, a server or other trusted entities cannot control all processes of the service. Only the peers can evaluate an interaction. Otherwise the server could enforce cooperation. Third, an evaluation of behaviour is possible. This means that someone, usually the client peer, needs to be able to observe misbehaviour and honest behaviour of the serving peer[5]. Fourth, the assumption that the behaviour towards other peers is a good projection for the behaviour towards a particular other peer[6]. Fifth, the experience and evaluation of other peers is trustworthy and mechanisms in the system ensure that false evaluations have no significant impact. We could also say that the reputation system needs to be reasonably secure.

There are also requirements that the reputation system needs to fulfill. Some reputation systems rank peers instead of linking them with a quality. This may be sufficient if the system needs to select the best among a set of candidates. This is not sufficient if the system needs to know if the peer is good or bad, i.e. if it will cooperate or not. For using the reputation system as incentive, we usually want the latter. We want to know the likelihood of cooperation. A more friendly bad peer is not a good choice because one

[4] As positive ratings are standard, a percentage of 90% is already rather bad, given enough transactions. The idea behind this approach is good as it signals both quality and quantity of the previous experience with one user to the another user.

[5] E.g. breaking the privacy of a peer by sending data to other colluding parties cannot be observed.

[6] Cooperation may depend on sympathy or interests and, thus, the experience of many other peers may be irrelevant. Taste is another issue. Say, a good Chinese restaurant may get a bad rating from someone who was disappointed because she does not like Chinese food.

will most likely also lose the money. Thus, reputation systems where the behaviour can be differentiated into good and bad (and intermediate classes) need to be used.

4.5 Cheapriding - an attack against reputation systems

In this section, we consider that a system of whatever kind uses a reputation system as incentive for cooperation. It does not matter if the system is server-based or decentral. The peers in the system provide services to each other. Each interaction includes costs and benefits for each participant. In an auction example, the buyer profits from the item she receives and she has the monetary cost of the payment. The seller has the cost of having and transfering the item, but she profits from the money that was paid. In filesharing, the downloader profits from obtaining the file, while the file provider has the cost of providing the file. The costs may be a mixture of resource usage and potential legal threats.

Considering the freeriding problems in Peer-to-Peer filesharing networks, a freerider does not share any files. The freerider just consumes the services without a return. With the first measures to counter freeriding employed in some systems, some freeriders offered unpopular and unproblematic files to show enough good will to be able to primarily consume. One can already call this cheapriding instead of freeriding, but literature does not differentiate these freeriders from other freeriders.

The basic idea of cheapriding is to maximize the profit within a system where a reputation system is used to ensure cooperation. Reputation systems are often agnostic to cost and benefit of operations, thus, the same reputation can be achieved with an extreme difference in profit. A cheaprider does less-profitable or costly interactions only as much as necessary in order not to get penalized.

The notion of cheapriding was inferred from the idea of weighting trust with cost in order to be able to faster detect severe strategy change attacks, which was proposed by our student Ingmar Schuster. A peer may do a lot of simple interactions in an auction system like Ebay in order to get a good rating. Once it is trustworthy enough, it changes its strategy. This means it defects (takes money, but does not send the items) and sells some high profitable goods to make enough profit. The idea to start with less-valuable goods instead of expensive ones before defecting is that it is easier to obtain and sell 10,000 USB sticks than 10,000 cars. Expensive items that are sold to defect do not need to exist. Thus, the initial investment for the attack can be small.

The basic idea of the weighted trust as counter-measure is to include cost into the reports. Considering the example, selling 10,000 USB sticks does not make one trustworthy for selling expensive goods like cars.

The cheapriding attack can be combined with this strategy-change attack. In a strategy-change attack, an attacker will behave differently over time and, thus, the current reputation of a peer may not be a good estimator for future behaviour. So, the goal of the strategy-change attack is to make the reputation scheme worthless. Cheapriding is then an optimization. Reputation is gained with cooperation on cheap interactions and expensive interactions will be consumed. If the system does not exclude misbehaving peers after frauds, the cheaprider may mix the good and bad behaviour instead of changing it from good to bad as in a strategy-change. Cheapriding is therefore related to the cost-averaging effect in finance. The cost-averaging strategy is to regularly invest a fixed amount of money instead of investing in a fixed number of stocks. The advantage of this strategy is that more stocks are bought when they are cheap and less stocks are bought when they are expensive. The effect is a lower average price per stock. The cheaprider uses the fixed price in reputation to serve cheap requests and to consume expensive ones.

4.5.1 The cheapriding attack on Ebay reputation system

There are two basic strategies for the attack on auctions. The first option is to use the strategy change and optimize with cheapriding. The second option is longterm cheapriding with a strategy to avoid the severe counter measures.

The first step for an attacker is to gain reputation. Reputation is necessary as users are more willing to bid for items from a well-reputated seller than for someone unknown. The goal is to get reputation fast. As the attacker needs to cooperate she also needs to buy and then sell the goods with which she wants to gain reputation. So, the decision of what items to sell is a compromise of low price, popularity, and knowledge of the attacker[7]. For these items the attacker will cooperate and gain reputation. It may not be important that the attacker makes profit with the items, but if the situation allows to, it should be seen as optimization and this can be used to build enough money to be able to sell more or maybe better items. As in this first variant the attacker is going for a strategy change, there will be one point in there where fake auctions with high profit will be placed. For these auctions, the items do not exist and the attacker will only take the money. As a time window the attacker can expect roughly two weeks at least. This comes from money transfer, potential delays, problems with postal service, etc. It is more likely that the problems may be detected even later. Once this is the case, the attacker account will be closed. As the attacker is likely to face prosecution, the attack needs to be highly profitable. So, within the defect phase of the attack, enough profit for being able to disappear has to be made.

We can categorize this attack variant as high profit, but only one shot. The idea in the other variant is to reduce the profit and in exchange being able to run the attack for a long time.

To be more precise, the second variant is to avoid prosecution and exclusion from Ebay as well as a bad reputation. How can this be achieved? The basic idea is to play with the tolerance of the system and preferably use a private seller account to avoid business-related guarantees to the buyers. Users may accept small fraud if their loss is insignificant and bothering would be too costly or time-consuming. The worst thing they may do is bad rating. In some cases this may not even happen if the user forgets about the incident before because it turned out to be complicated, e.g. denying money transfer, sending wrong cheaper product (one may even get a positive rating for that, although it may become expensive when the item needs to be returned and shipped again), etc. To hide the bad behaviour we have to show good behaviour and the good behaviour has to end up in 95 % of the ratings. This already shows a problem of this approach. It is most likely that the profit of being 100% honest is almost as high. Thus, it may not be worth the risk.

4.5.2 The cheapriding attack on Peer-to-Peer filesharing and distribution

Apart from the strategy with unpopular and legal files, a cheaprider can also optimize the cost. Let us consider the gain in reputation be identical for all files. Thus, sending 100 bytes is equally valuable as sending a GB.

So, the cheaprider prefers to serve small files and it will consume big files instead. If a larger file is requested it is likely to defect. The strategy can also be transferred to layered

[7]It is evident that someone who is an expert for the items sold is better in determining the right items and in presenting the items. From private experience which is not representative we can say that we have never seen a good product description and presentation by auction services that sell goods for customers. Usually it is obvious that the service employees did not know anything about the product. It is not surprising that the auction did not get a good price – if the item was sold at all.

video files and the attacker reducing the quality by simply reducing the layers it sends. Such an approach is more expensive for formats that do not support this as re-encoding is likely to be more expensive than the gain.

4.5.3 Mitigation strategies

There are some ways in which cheapriding attacks can be fought. In most scenarios they cannot be stopped completely, but their impact and likelihood can be reduced. That is why we only speak of mitigation.

The attack can be fought within the reputation system and apart from the reputation system. First, we will discuss what the reputation system could do. The basic idea here is to include the cost and profit in the reports and trust evaluation. There are many possible ways to combine cost and trust. One may calculate trust for different price segments or general categories. So, trust for USB sticks will not imply trust for cars. One can also observe the behaviour for different categories and detect cheapriding and related attacks. In the weighted trust example a weighted sum of cost and trust is used as well as an inversely weighted cost. With these two parameters strategies can be categorized and depending on the system one can classify certain areas as fraud (cheapriding) and other areas as good behaviour. Of course, there are still limits within a cheaprider can operate and optimize. In particular for the auction example, the buyer may be stupid enough to do the transfer. Such a risky behaviour is also necessary to build reputation as also a car seller needs to start her business first before becoming a well-reputated seller. Moreover, false positives or false negative are a problem as fraud and honest behaviour may overlap and is therefore not distinguishable for certain (hopefully less profitable) cheapriding strategies.

There are some further issues. As the example already indicates, the reputation software cannot stop the actual user or software to do risky transactions. Furthermore, it also depends on their input. Often, cost or profit cannot easily be determined. So, there is an inadequacy an attacker can use. Another issue is the timeliness of feedback. A corrupt file may not be known as currupt before the user has seen it. In the auction, the transaction of money, shipping, and overall processing of an order may allow weeks before the fraud is recognized as such. This may be better for networked applications where the software can directly check the correctness and recognize cooperation. If evaluation has to wait, the attacker has a long time window for profit with a strategy change attack. For long-term cheapriding the cheaprider can do more transactions before being detected due to the delay in evaluation.

Cheapriding can also be fought outside of the reputation system. One solution could be based on micropayments. The basic idea is to divide the service into small accounted chunks that can be immediately evaluated and paid. The loss in case one side defects is only one such micropayment, not the overall price for the complete service. Assumptions are that the service can be completed by others or that a partial service is already sufficient and only reduces quality. The reputation system helps to avoid such peers that often defect and it therefore helps to minimize the loss of honest peers.

A completely different approach is to make the software trustworthy and reduce the attack vectors for the user or attack software. This is close to the trust model of Skype. Software and protocol are closed-source and the software uses anti-debugging tricks to avoid analysis. One may use less extreme measures, but they might be broken if the application is not sufficiently protected. The basic idea here to ensure that the core functionalities are unaffected by the cheaprider. A simpler example than the Skype example is that most filesharing software requires a minimum upload bandwidth that cannot be reduced to $0kbps$, so that all downloaders also share the downloaded chunks of the file.

Finally, the last solution we present is to introduce severe consequences in case a fraud is detected. This is the standard solution in commercial scenarios like Internet auctions. The cheapriding attack cannot continue because the account of the attacker gets closed and prosecution might bring the attacker into jail (depending on the severity of the crime). This reaction is good against long-term attackers and also mitigates strategy change attacks. The basic problem is that a limited percentage of fraud needs to be accepted due to false reports, accidents, and misunderstandings. The consequence is either that there is still some room for cheapriding or that severe consequences are taken on false reports. False positives are a big problem and may prevent such counter-measures.

4.6 Relation to Architecture and Design

From a security perspective a closed architecture and the strict enforcement of security are design decisions of choice. It is best to make a system most secure. But there are other issues like scalability and usability that may prevent such solutions. Systems do not exist for their own sake, they need to be used and useful for the intended set of users. Most Peer-to-Peer systems want to be popular and attract a lot of users. This may conflict with strict security decisions.

The assumption that Peer-to-Peer systems are a trusted web of peers is not realistic. Security problems have to be solved before running a critical application on top. For an application it is necessary to determine the security it requires. The realization of security in a Peer-to-Peer network is not easy and usually it will partially break the Peer-to-Peer idea.

More details on security are presented in the next chapters. Performance is discussed in Chapter 5, Authentication and Authorization in Chapter 6. The last two chapters are about the security goal anonymity. Chapter 7 provides a more-detailed overview. Chapter 8 is about the MORE system and its analysis.

5. Performance of Security Mechanisms

5.1 Introduction

The performance of a secure system depends on the performance of the general system, the security algorithms, and their protocol embedding in the system.

Most communication on the Internet today is not secured. For non-personalized content like static websites, there may be little need for security. One might speculate whether this is a question of performance - true from provider perspective - or immanent to the system - in many cases it is not clear what security is, say the download of a website from a known webserver is not very confidential even if encrypted. In open systems like the Internet there is also no well-known key to protect the site.

In this chapter we focus on the performance aspect. The computational power is still increasing, although now more into multicore directions and less increase for the individual performance of a single core. Nonetheless, it is very valuable to notice that even small handheld devices are more powerful than computers few years ago and probably more powerful than computer pioneers ever thought that computers will be.

As a consequence, cryptographic operations become cheaper and more realistic for more and more applications. The increased usage of wireless communication is another driving factor for the use of crytography, as in particular wireless communication is a threat to confidentiality.

Only if too many applications produce too much of an increase of operations, the effect of better computers will be destroyed. Ofcourse, cryptographic algorithms and secure systems also need to withstand stronger adversaries.

On the current Internet, there is no overall security mechanism. Security can be on the basis of systems, e.g. IPSec tunnels set up by administrators, or done on application level by the applications themselves.

The compromise between security on the one hand and performance on the other is by the term Quality of Security Services (QoSS). QoSS allows to vary the level of protection ('how secure') within a given range to improve performance on the systems or for the communication, e.g. CPU load or throughput. We will relate measurement results and QoSS later in this chapter.

5.2 Performance of building blocks for cryptographic protocols

As a first step, we briefly address the performance of cryptographic functions. The primary purpose is to create a rough understanding of the complexity as particular values are system-dependent.

5.2.1 Symmetric cryptography

In this section, we look at symmetric encryption and decryption. Symmetric ciphers operate on a shared key that both parties know. The size of this shared key varies from 96 bits to 256 bits in today's applications. The attacks against good ciphers are close to the $2^{keysize-1}$ of a brute force attacker.

The values in this section were derived from experiments with the libgcrypt of the gnupg software project. For the ciphers in the table encrpytion and decryption did not need significantly different times. Table 5.1 shows the results. Twofish is slightly faster than AES. 3DES significantly slower. The different encryption modes hardly differ in their performance, except for counter mode (CTR). CTR was significantly slower. This is surprising as it is expected to be fast as blocks do not depend one previously encrypted blocks. So, in theory it should be slightly slower than ECB and faster than the rest. In many recent implementations Twofish outperforms AES, although in the AES competition performance was important for selecting the Rijndael cipher over other ciphers including Twofish.

Cipher	ECB	CBC	CFB	OFB	CTR
3DES	0.100 ms	0.104 ms	0.103 ms	0.101 ms	0.152 ms
AES	0.041 ms	0.046 ms	0.043 ms	0.042 ms	0.093 ms
Twofish	0.034 ms	0.039 ms	0.039	0.038 ms	0.089 ms

Table 5.1: Symmetric Ciphers - time for processing 1,000 bytes.

Cipher	Throughput
3DES	78.8 Mbps
AES	178 Mbps
Twofish	210 Mbps

Table 5.2: Symmetric Ciphers - corresponding throughput rates for CBC.

A general observation is that symmetric encryption and decryption is fast and that modern ciphers like Twofish or AES combine a good performance with security. IPSec measurements even showed a better performance. The difference is caused by different libraries and hardware.

5.2.2 Asymmetric cryptography

Asymmetric cryptography allows two or more parties to communicate securely without a shared key. The public key of each party is well-known and used by the other parties. The private key provides a trapdoor to being able to invert the public key operation. Depending on the algorithm used for asymmetric cryptography the key size can vary a lot. It is not directly related to the cryptographic strength of the cipher.

Again, we used the libgcrypt of the gnupg project to get the results.

As the keys relate to an instance of some algorithmic problem, the generation of keys is more complicated and they are not purely random numbers. For the generation of a RSA

Cipher	Sign	Verify
RSA 1024	5 ms	0.2 ms
RSA 2048	37 ms	0.8 ms
ECDSA 192	6.5 ms	10 ms
ECDSA 256	10 ms	23 ms
ECDSA 512	54 ms	96 ms

Table 5.3: Asymmetric Ciphers

key we get $150ms$ for a 1024 bit key and $1s$ for a 2048 bit key. For the elliptic curve ECDSA we could generate keys within $40ms$ (192 bits) to $187ms$ (512 bits).

Table 5.3 lists the performance of RSA and elliptic curves. The difference between sign and verify (encrypt and decrypt) for RSA comes from the size difference of the exponent and key pairs could be switched to achieve the inverse results.

Asymmetric cryptography is significantly slower than the symmetric cryptography. Even the 0.2 ms for the RSA 1024 verify is 16 times slower than a comparable symmetric operation. Yet, this still ignores the long time (25 times longer) for the signature. Nonetheless, asymmetric cryptography has its use-cases. Most important, no prior key is needed and other entities can prove interaction with a second party to a third party. This is necessary for contracts with digital signatures, where seller and buyer may need to prove the contract to a judge.

5.2.3 Identity-based Cryptography

Identity-based cryptography (IBC)[Sham85] allows arbitrary strings as public keys. As the name identity-based suggests, the identity is used as public key. This is elegant as the name can be human-readable and without certificate or communication with a TTP, each entity can prove their identity. So, IBC is the inverse approach to cryptographic identifiers where a key is the identity and the identity is, thus, not human-readable. Today there are a variety of proposal on how to realize and use IBC efficiently[Mao04], in particular the Weil pairing of Boneh and Frankin[BoFr01].

IBC has one drawback. It is realized with a central entity (Key Generation Center KGC) that has the secret master key to compute the corresponding private keys for each entity from their name.

For our measurement we used the PBC (Pairing-based Crpytography)[1] library from Stanford University. We set the ID and key size to 1024 bits. The generation of the private key from the name took 38.5 ms. This is done in the KGC once for each entity. It only needs to be repeated if the masterkey is changed. For the communication between two entities, they need two steps. The generation of the ID from the name took 27.5 ms. To calculate the shared key from the ID needs 5 ms.

This process does not yet include any encryption. We only derive a shared key that can subsequently be used for symmetric cryptography to encrypt and decrypt. Both sides need to perform the operations to determine their key. So, while IBC still lacks non-experimental libraries for widespread use, it can be compared to public key cryptography with respect to performance.

5.3 Performance of complete protocols

In this section we evaluate the performance of a complete network security protocol.

[1]The Pairing-Based Cryptography Library http://crypto.stanford.edu/pbc/

5.3.1 Latency of IPSec processing

In this section we measured the latency of IPSec in the Linux kernel.

Measurement Methodology

In this section we present measurements of the processing delay of IPSec. To be more precise, the delay a packet receives from entering to leaving the IPSec layer. These measurement points were set using hooks in the Linux IP stack. Unless otherwise stated the results presented are from outgoing packets and were measured from IP_Local_Out to IP_Post_Routing hooks. The time measurements were done using the Pentium CPU time-stamp-counter (TSC) register. All measurements were performed using Intel Pentium 4 2,60 GHz computers with Linux 2.6.9 and its native IPSec and all unnecessary services terminated. For comparison we also used Linux 2.4 with StrongS/WAN[Stro].

The measurement software stored the results in an array during the evaluation and purged them to disk afterwards. For the measurement, single ICMP packets with different packet sizes were sent using the command ping. 10.000 samples were taken for each combination of the algorithms and outliers due to system specific distortions, say interrupts, were manually removed using the statistics software R [fSta]. The impact of the outlier removal on the mean is at most 2,7 %, but usually well below. Due to the low variation and the large number of samples the confidence intervals are narrow.

One limitation of this paper is that we solely look at the latency of sending a packet. However, there might be small differences for the results of packet reception caused by different performance of encrypting and decrypting a packet, impact of caching, etc.

Authentication Header

The Authentication Header Protocol is the IPSec protocol for message authentication and data integrity, usually achieved using cryptographic hash function in the HMAC construct. Table 5.4 presents our measurements with AH using the native IPSec of Linux 2.6.9. As expected SHA-1 is slower than MD5.

Algorithm	Mode	Cycles	Time
MD5	Transport	23665	9.10 μs
MD5	Tunnel	24591	9.46 μs
SHA-1	Transport	63628	24.5 μs
SHA-1	Tunnel	65916	25.3 μs

Table 5.4: AH - delay for packet with 1400 B payload between IP_Local_Out to IP_Post_Routing

The processing of AH with HMAC-MD5 needs approximately 23,500 cycles in Transport Mode. This corresponds to a data rate of 1.2 Gbps (ignoring other overhead and internal limitations). The processing of SHA-1 takes approximately 64,000 cycles. The corresponding data rate would be 460 Mbps.

Encryption with Encapsulated Security Payload

The Encapsulated Security Payload is the IPSec protocol for confidentiality, usually provided by symmetric encryption, e.g. with block ciphers in CBC-mode as in all our measurements. Table 5.5 presents our measurements with ESP using the native IPSec of Linux 2.6.9. The AES is the fastest encryption algorithm in these measurements. Its delay for processing a segment with 1,400 bytes payload length is about 65,000 cycles, which corresponds to an assumed data rate of 440 Mbps (ignoring other overhead and internal limitations). The slowest algorithm is 3DES with a delay of 395,000 cycles and corresponding data rate of 74 Mbps.

Algorithm	Mode	Cycles	Time
AES-128	Transport	65629	25.2 μs
AES-128	Tunnel	66620	25.6 μs
AES-192	Transport	70976	27.3 μs
AES-192	Tunnel	72927	28.0 μs
Blowfish-128	Transport	112603	43.3 μs
Blowfish-128	Tunnel	116292	44.7 μs
3DES	Transport	394956	152 μs
3DES	Tunnel	398989	153 μs

Table 5.5: ESP - delay for packet with 1400 B payload between IP_Local_Out to IP_Post_Routing

Combining ESP and AH

A realistic scenario combines both ESP and AH to achieve a maximum of security. Table 5.6 lists some results. A typical combination is ESP with AES-128 and AH with SHA-1. The processing of IPSec in this case takes about 131,000 cycles, which corresponds to a data rate of 222 Mbps (ignoring other overhead). Using ESP with 3DES in this combination increases the processing overhead to 451,000 cycles (data rate 64.5 Mbps), which is more than factor 3! The Tunnel Mode that is used for scenarios like Virtual Private Networks. It is approximately 2,000 to 6,000 cycles slower than the Transport Mode.

ESP	AH	Tunnel Mode	Transport Mode
AES-128	SHA-1	133481	131012
3DES	SHA-1	456558	450979

Table 5.6: ESP+AH - delay in clock cycles for packet with 1400 B payload between IP_Local_Out to IP_Post_Routing

Packet Size

All the results presented so far were for large packets that are close to the size limit imposed by the standard Ethernet. This is one typical packet size, but smaller packets are also common, e.g. for ACKs or real-time traffic. Table 5.7 presents the delay measurements for packets with the sizes 64 B, 300 B, 700 B, and 1,400 B.

ESP	AH	Mode	64 B	300 B	700 B	1400 B
AES-128	-	Transport	5907	16890	35069	65628
AES-128	-	Tunnel	6607	17565	35233	66620
AES-128	SHA-1	Transport	24587	43944	74956	131012
AES-128	SHA-1	Tunnel	25084	44946	76666	133481
-	SHA-1	Transport	16037	25365	38773	63627
-	SHA-1	Tunnel	18433	25264	41031	65915

Table 5.7: Native IPSec/Linux 2.6.9, Delay in clock cycles for packets with different payload size between IP_Local_Out to IP_Post_Routing

Figure 5.1 visualizes the delays for different packet sizes for the measurements with IPSec in Transport Mode. The linear regression is also plotted and shows that a linear model

$$Latency_{avg}(Size) = a * Size + b$$

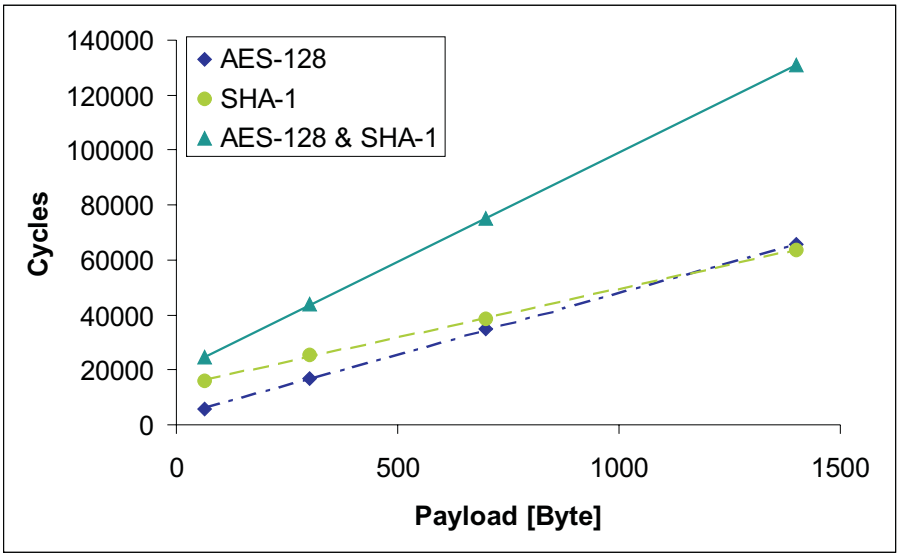


Figure 5.1: IPSec with kernel 2.6.9 in Transport Mode Delay

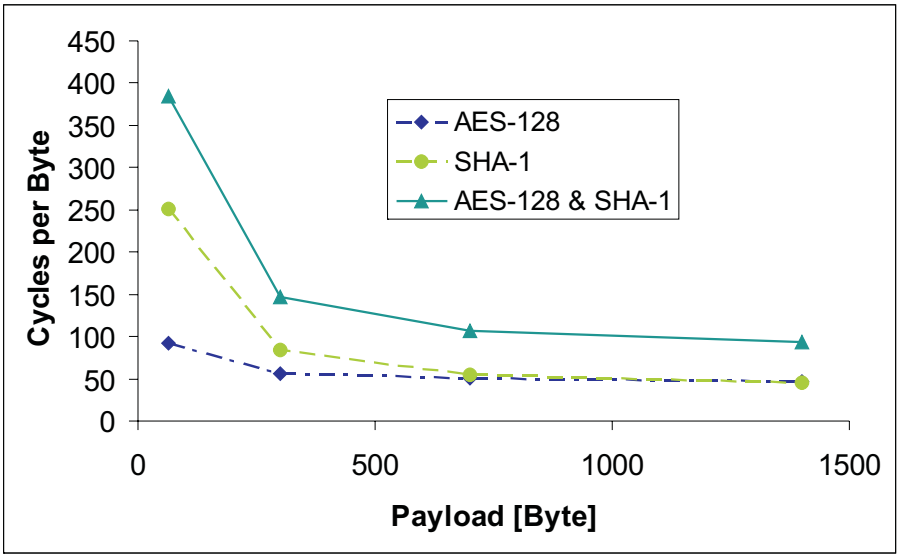


Figure 5.2: IPSec with kernel 2.6.9 in Transport Mode Delay per Byte

is already sufficient. a gives the number of cycles per byte and b the constant overhead in cycles. For the case with AES-128 and SHA-1 in Transport Mode linear regression gives $a = 79.5 \frac{cycles}{Byte}$ and $b = 19,700\, cycles$.

Figure 5.2 gives another insight on the data. As expected, the overhead per byte is larger for small packets. For packets with a payload of 300 B and more the overhead per byte is already close to the overhead per byte for large packets. Thus, the constant term of the processing delay is mainly important for small packets, e.g. packets in audio streams with small payload sizes.

Comparison of Linux 2.6.9 and StrongS/WAN

When we started the measurements the use of Linux 2.4 was still common and IPSec was not yet part of the kernel. We used StrongS/WAN which is based on the FreeS/WAN project which stopped their development when a different IPSec implementation was to be integrated in the 2.6 kernel. Table 5.8 shows the results.

ESP	AH	Mode	64 B	300 B	700 B	1400 B
AES-128	-	Transport	20918	26294	37465	49585
AES-128	-	Tunnel	25964	31293	39816	54726
AES-128	SHA-1	Transport	44518	58224	75235	114530
AES-128	SHA-1	Tunnel	48847	62353	82495	115117
-	SHA-1	Transport	32186	40336	50160	69861
-	SHA-1	Tunnel	42665	44470	60486	80080

Table 5.8: StrongS/WAN/Linux 2.4 - IPSec-related Delay in clock cycles for packets with different payload size

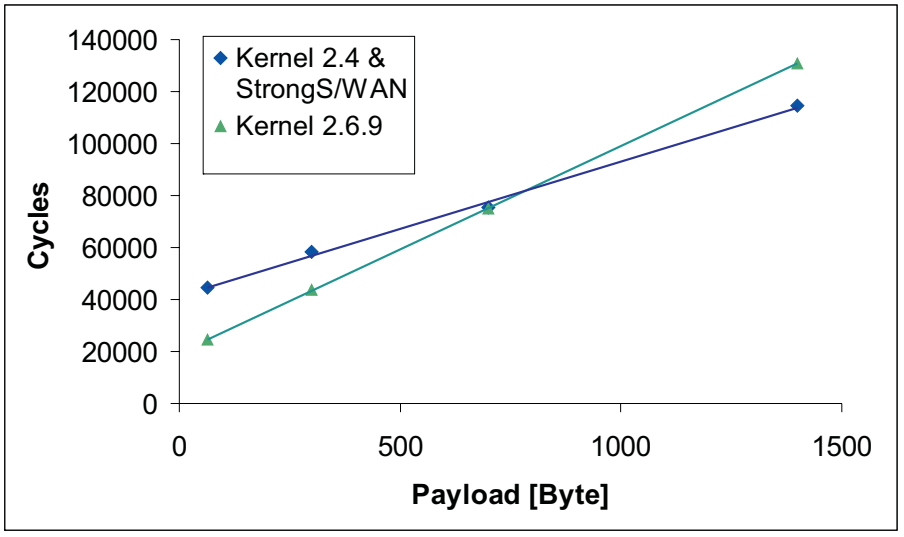


Figure 5.3: Comparing the measurements for StrongS/WAN with the measurements with Linux 2.6.9

Even more interesting is the comparison of these results with the results from the previous sections where we used the native IPSec implementation of Linux with kernel 2.6.9. Figure 5.3 shows the comparison of the results. StrongS/WAN performs better for large packets and the native IPSec of Linux 2.6.9 is better for small packets, e.g. 64 bytes packet size.

One might speculate that while the algorithms are better optimized in StrongS/WAN, native IPsec is naturally better integrated into the kernel and has less overhead itself.

5.3.2 Throughput for IPSec

In this section we measured the throughput.

Measurement Methodology

In this section we present throughput measurements mainly for IPSec with various configurations. All measurements were performed using Intel Pentium 4 2,60 GHz computers with Fedora Linux and its native IPSec implementation with all unnecessary services terminated. The computers were directly connected via Gigabit Ethernet. Iperf was used as a tool to perform throughput measurements. The values are mean values taken from at least 3 Iperf runs (9 runs for encryption only and 18 runs for hash function only) with each run having 30 seconds duration.

At first, we performed the measurements with the same kernel (Fedora Linux 2.6.9) where we performed the latency measurements within the IP stack. Just recently we repeated the measurements with the new 2.6.13 kernel. The new version has an improved DES implementation among other improvements in the kernel. However, the general performance increase was suprising to us, but our tests with another 2.6.9 kernel variant showed results similar results to our 2.6.9 reference distribution. However, there is a variation between installations with the different distributions.

Unless otherwise stated we given values are from measurements with the kernel 2.6.9 in a standard Fedora installation.

Authentication Header

Table 5.9 and table 5.10 provide an overview of the throughput measurements with AH. AH with MD5 hardly reduced the throughput of IP datagrams. It is interesting that the newer 2.6.9 kernel has a lower IP and MD5 throughput, but all other IPSec combination profit from its use. The use of SHA-1, the most commonly used cryptographic hash function, leads to a throughput of roughly 300 Mbps (kernel 2.6.9) and 350 Mbps (kernel 2.6.13). The use of the most secure hash function in our measurements, SHA-2-256, reduced the performance of AH to 186 Mbps.

Algorithm	kernel 2.6.9	kernel 2.6.13
IP	825 Mbps	817 Mbps
AH with MD5	610 Mbps	599 Mbps
AH with SHA-1	298 Mbps	349 Mbps
AH with SHA-2-256	186 Mbps	189 Mbps

Table 5.9: AH performance in Transport Mode

Algorithm	kernel 2.6.9	kernel 2.6.13
IP	825 Mbps	817 Mbps
AH with MD5	595 Mbps	580 Mbps
AH with SHA-1	294 Mbps	343 Mbps
AH with SHA-2-256	183 Mbps	186 Mbps

Table 5.10: AH performance in Tunnel Mode

Encryption with Encapsulated Security Payload

The Encapsulated Security Payload is the IPSec protocol for confidentiality, usually achieved using symmetric encryption, e.g. with block ciphers in CBC-mode. Tables 5.11 and 5.12

present the results. The performance of all the algorithms dramatically increased when switching from kernel 2.6.9 to 2.6.13. The throughput of ESP with AES-128 increased from 314 Mbps (kernel 2.6.9) to 456 Mbps (kernel 2.6.13).

Blowfish is designed to have a performance independent of the key size. This can also be seen in the results.

With the 2.6.13 kernel, Fast Ethernet can almost be saturated using ESP with 3DES.

Algorithm	kernel 2.6.9	kernel 2.6.13
IP	825 Mbps	817 Mbps
ESP with AES-128	314 Mbps	456 Mbps
ESP with AES-192	295 Mbps	419 Mbps
ESP with Blowfish-128	192 Mbps	336 Mbps
ESP with Blowfish-192	192 Mbps	337 Mbps
ESP with DES	132 Mbps	212 Mbps
ESP with 3DES	66 Mbps	97 Mbps

Table 5.11: ESP performance in Transport Mode

Algorithm	kernel 2.6.9	kernel 2.6.13
IP	825 Mbps	817 Mbps
ESP with AES-128	306 Mbps	441 Mbps
ESP with AES-192	287 Mbps	404 Mbps
ESP with Blowfish-128	191 Mbps	325 Mbps
ESP with Blowfish-192	189 Mbps	324 Mbps
ESP with DES	127 Mbps	206 Mbps
ESP with 3DES	64 Mbps	95 Mbps

Table 5.12: ESP performance in Tunnel Mode

Combining ESP and AH

True security requires both encryption and message authentication. Thus, it is necessary to combine both. We skip the values of combinations with DES as is insecure due to its small key size (56 bits relevant, key itself is 64 bit).

	MD5	SHA-1	SHA-2-256
AES-128	231 Mbps	166 Mbps	123 Mbps
AES-192	225 Mbps	160 Mbps	120 Mbps
Blowfish-128	164 Mbps	127 Mbps	99 Mbps
Blowfish-192	163 Mbps	127 Mbps	99 Mbps
3DES	60 Mbps	56 Mbps	50 Mbps

Table 5.13: ESP+AH performance in Transport Mode, kernel 2.6.9

Tables 5.13 and 5.14 present the results. The fastest reasonable IPSec combination is ESP with AES-128 and AH with SHA-1. Its throughput is 166 Mbps (kernel 2.6.9) or 223 Mbps (kernel 2.6.13) in Transport Mode. Comparing this section with the results of the ESP and AH sections we additionally note that message authentication with the hash functions is increasingly dominating the performance, especially considering that SHA-2-256 is the only hash function in this study that is not yet heavily under attack.

	MD5	SHA-1	SHA-2-256
AES-128	309 Mbps	223 Mbps	148 Mbps
AES-192	289 Mbps	213 Mbps	142 Mbps
Blowfish-128	244 Mbps	190 Mbps	129 Mbps
Blowfish-192	244 Mbps	190 Mbps	130 Mbps
3DES	88 Mbps	81 Mbps	69 Mbps

Table 5.14: ESP+AH performance in Transport Mode, kernel 2.6.13

AH vs ESP

With IPSec it is not necessary to use ESP and AH for achieving confidentiality and message authentication as ESP supports both. Many experts recommend the use of ESP and AH, because AH also authenticates non-volatile fields of its IP header.

AH adds 12 B extra overhead and additionally authenticates the outer IP header. So, the combination of ESP and AH should be slower. However, the extra 12 bytes AH header are small compared to payloads of more than 1400 B.

Algorithms	ESP	ESP & AH
AES-128/SHA-1	167 Mbps	166 Mbps
AES-128/SHA-2-256	125 Mbps	123 Mbps
Blowfish-128/SHA-1	126 Mbps	127 Mbps
Blowfish-128/SHA-2-256	99 Mbps	99 Mbps
3DES/SHA-1	56 Mbps	56 Mbps
3DES/SHA-2-256	51 Mbps	50 Mbps

Table 5.15: ESP vs ESP+AH, kernel 2.6.9

Comparing the results presented in Table 5.15 there is no significant impact on the performance whether ESP and AH or ESP with authentication is used. Thus, the recommendation is to use ESP and AH as it provides better authentication with no or negligible extra-cost. An exception to this recommendation are small packets and the additional overhead of 12 B for AH might me unacceptable.

Comparison with SSL-based Tunnels

Finally, we compare IPSec tunnels with tunnels based on SSL. Besides the native IPSec of Linux 2.6 we use the SSL tools stunnel [mStp] and OpenVPN [Open].

Algorithms	stunnel	OpenVPN
null	-	519 Mbps
AES-128 & SHA-1	180 Mbps	117 Mbps
Blowfish-128 & SHA-1	-	149 Mbps

Table 5.16: Performance of SSL tunnels

The performance values for the SSL-based tunnels are given in Table 5.16. IPSec with MD5 achieves a better performance than OpenVPN without encryption and authentication.

Using OpenVPN with AES is slow compared to Stunnel. However, the Blowfish implementation for SSL is rather efficient. Thus, if Blowfish is preferred as encryption algorithm OpenVPN is an interesting solution.

In general, the SSL tunnel tools partially outperform IPSec of kernel 2.6.9, but are usually slower than IPSec of kernel 2.6.13.

5.3.3 Discussion of IPSec Measurements

In this section, we finally discuss our results and put them in a general context.

Observations from the measurements

Our results exposed some interesting characteristics of the behavior of IPSec. The average processing latency increases linearly for the evaluated algorithms which is not surprising. However, there is a significant overhead for small packets below 300 bytes. We experienced that the behavior of the security services depends a lot on the IPSec implementation and also the linux kernel. The tested encryption algorithms of Linux kernel 2.6.13 showed remarkably better performance values than the ones of kernel 2.6.9. There are also differences between the StrongS/WAN implementation and the IPSec of the kernel 2.6.9. StrongS/WAN under kernel 2.4 performs better for small packets but cannot keep up with the current kernel for packets bigger then 700 bytes. These observations indicate that a careful implementation of the security services and the network stack have a large impact on the performance.

Hash functions

Recently, NIST started a contest for a new hash function and the first round of the competition is about to be finished soon. This was mainly initiated by the fact that MD5 is broken and SHA-1 has been heavily under attack. However, these attacks are mainly important for the security of digital signatures. Authentication in Internet communication is still considered secure.
Our results from the IPSec measurements add one more argument to the discussion about hash functions and the necessity to find a good new one that is secure and fast. ESP-AES performs better than AH-HMAC-SHA1 and ESP-Blowfish better than AH-HMAC-SHA-2-256. Thus, message authentication has turned into the bigger performance bottleneck than encryption. This has been different for the last decades and was unanticipated when we started the evaluation. Thus, the deployment a new secure and fast cryptographic hash function seems to be desireable.

Performance impact of security services

Security services operate on a rate out of reach for standard computers in the recent years. The performance of cryptographic algorithms and hash functions is no obstacle anymore for their wide scale deployment. Even the slowest combination (50 Mbps, 3DES/SHA-2-256) exceeds the connection speed of typical home users by far (e.g. current DSL in Germany at up to 16 Mbps).
However, performance remains an issue. Particularly servers with high bandwidth connectivity, say VPN gateways demand for highly efficient algorithms. Moreover, servers which do not exclusively serve as security gateway but provide other functionalities must limit the impact of security services. Resource-constrained devices, on the other hand, do not possess nearly as much resources as our test systems. Security is only one issue that may influence the performance. Even when its load is not the dominating bottleneck it is not necessary to do it inefficiently and waste energy.
The bandwidth of Gigabit Ethernet networks still exceeds the capacity of our test system to secure traffic. The expected evolution of network technology in fixed and wireless networks toward higher bandwidths may increase the gap.

Quality of Security Service

Most QoSS aware systems require a rating of the available security services. We will show exemplarily for ESAF [KMNC05] how the discussed algorithms can be classified.

Algorithm	Confidentiality	Performance
AES-128	8	8
AES-192	9	7
DES	2	4
3DES	9	2
Blowfish-128	9	6
Blowfish-192	10	6

Algorithm	Authentication	Performance
HMAC-MD5	2	9
HMAC-SHA-1	5	6
HMAC-SHA-2-256	8	3

Table 5.17: Possible values for ESAF policies

It is important to keep in mind that such ratings are subjective in nature and can only represent some usage scenarios. Hence the ESAF considers only locally available ratings to determine which protocols might be acceptable. There is no exchange of the ratings during the negotiation of the communication context for a connection.
Our results from the presented measurement study are useful to derive the performance levels of encryption and authentication services as shown in Table 5.17. The other scalar value expresses a rating about the assumed security for each algorithm. This rating is subjective in nature as well and can have a large impact on the choices the framework makes.
Let us consider an example. An application wants to establish a communication link with some security requirements. One requirement is that the minimal authentication security level is at least 4. The HMAC-SHA-1 would be chosen in our example, because its security rating is sufficient and it is faster than HMAC-SHA-2-256. If the HMAC-MD5 would possess a better security rating, say 4, it would be selected due to the better performance value.

5.4 Relation to Architecture and Design

In this chapter we presented the results of some measurement studies we undertook. We looked at symmetric cryptography, public key cryptography, and also tested Identity-based cryptography. As cryptography is only one part of the performance of a cryptographic protocol, we measured the performance of IPSec implementations.

Performance has been a major reason for not implementing security and encryption some years ago. It may still be an issue today if an application has special requirements or a lot of public key cryptography has to be used. Identity-based cryptography can be compared to public key cryptography in both its use-cases as in its performance. It is not widely deployed as of yet and most libraries do not support it. Our only recommendation for designing applications is that cryptographic protocols and services should be used where it makes sense and that designers should avoid a heavy use of asymmetric cryptography.

6. Authentication and Authorization for Peer-to-Peer systems

6.1 Introduction

In the last chapter we have already mentioned that the Peer-to-Peer paradigm with its reduced control has additional problems with security. This is also true for authentication, as most systems are agnostic to identities and open to anyone.

Authentication is not a question of open system or closed system. In a closed system authentication and authorization are required. In an open system, the system may not restrict the user group, but will use authentication to keep longterm identities and to prove the current identity for the time of a session.

The classic Peer-to-Peer filesharing and distribution systems operate differently. They do not want to secure the peers, but to secure the file. In BitTorrent, the initiator of a torrent provides a .torrent file with cryptographic hashes for the complete file and for its chunks. The file transfer for the .torrent file is out-of-band. For other filesharing systems, similar measures may be taken. However, the peer that downloads the file is not secure against the provider of the file and may download noise or even a virus instead of the desired song.

6.2 Authentication and Authorization

In our work we do not limit Peer-to-Peer networks to identityless filesharing networks. If one needs an example for other Peer-to-Peer systems, messaging systems like Skype or Voice-over-IP are good candidates. We consider peers to have an identity. This identity may be related to the ID used as address on Peer-to-Peer network layer, but we do not necessarily assume this.

Authentication provides the means to prove an identity. For Peer-to-Peer systems authentication and access control are closely related. In particular, Peer-to-Peer-alike methods often seem to aim at both at the same time. This is in contrast to classic systems where one tries to separate the two operations. A common example for a Peer-to-Peer setting is that a new node wants to join and needs to convince a group of existing nodes of its identity and the legitimacy to join the network. If this succeeds, it may enter the network. As one can see, the identity alone may not be sufficient to decide the node's request as

its name may be arbitrary and access rights are not centrally coordinated. In a friend-to-friend network a new node has to convince the other nodes that he really is the friend and not someone with the same name.

The following list displays situations when authentication is necessary. It may not be complete, but includes the most important ones.

- A node wants to join the Peer-to-Peer system.
- Peers yet unknown to each other want to communicate.
- Peers that know each other want to communicate.
- Determine identity of a peer.
- Establishment of a secure channel between two peers.

As we mentioned Skype, we will briefly sketch their authentication. Skype uses authentication servers that are contacted when users log in. This is a centralized approach that guarantees that user accounts and their names are secured. Another security feature of Skype is their authentication of the Skype software. Skype clients recognize each other on the basis of private keys in the software. The keys are protected by Skype's anti-debugging tricks and encryption of most parts of the binary[BiFa06].

6.3 Standard and non-Standard Solutions

One standard solution for authentication is a Public Key Infrastructure (PKI). It is a centralized concept where a hierarchy of trusted authorities provides certificates for authentication. Another common PKI concept is a Web-of-Trust.

As Peer-to-Peer settings are similar to joint operations among a group of entities, secret sharing and multi-party computation methods are also discussed as candidates that combine authentication and access control for the system.

In the following sections, we briefly discuss these solutions, how they may be applied and what they do not provide. The latter gives reasons why any of these solutions is not satisfying.

6.3.1 Centralized Authority

The first solution is not very compatible with the Peer-to-Peer paradigm. A server acts as authority for the network. The server is either a Certificate Authority (CA) or a Trusted Third Party (TTP) that controls the authentication and access control for the system, e.g. the CA does this by not giving certificates to unauthorized entities. The approach hardly differs from the client/server case. Security is ensured by the central authority. Unlike many other approaches it can provide the requested security, which makes it the preferable choice for security if it is possible. Looking beyond authentication as a security goal, Peer-to-Peer networks with such a solution may still face security problems as critical operations are not computed on a trusted machine.

This centralized approach may contradict the Peer-to-Peer paradigm, but only as much as Napster also violates this. Other proposals for network optimization are supervised networks[RiSc04] where a central node coordinates the network. A PKI for Peer-to-Peer systems is, thus, a possible solution[BeEM04, Wölf05].

From a conceptual point of view we can implement this centralized approach either with a hierarchy of CAs or a group of TTPs. CAs do not actively participate in the protocol. An

exception is the checking of the certificates for revocation, which is often considered to be optional (although the philosophy changes towards checking them more often). Without revocation checks, they are more scaleable and also more robust against Denial-of-Service attacks. TTPs operate on most up-to-date information and systems on their basis do not have a revocation problem. The security is more clearly enforced and symmetric cryptography may be sufficient. Furthermore, an active TTP can provide additional features like implementing a reputation system or being a superviser in a supervised Peer-to-Peer network (additionally performing network management and optimization).

The centralized PKI is our solution of choice if it can be applied and a more decentralized or open setting is not necessary. Such a necessity may be due to robustness issues or perhaps an anarchic nature of the network. Even though a distribution to multiple TTPs/CAs may increase robustness, they still remain high-value targets and almost single-points-of-failures. There is also the issue of an attacker gaining credentials by attacking a TTP/CA. It might be preferable not to have all information available on one node.

For many systems there may not be a centralized authority that all peers respect and that has the knowledge to be able to provide the service. For a Peer-to-Peer system with a global scope, a central authority may have problems to check identities of users and machines from other parts of the world. There are many use-cases where this cannot be guaranteed or the goal of authentication is adapted accordingly. Centralized PKIs are also under criticism from the security perspective, e.g. [Gutm02, ElSc00]. On a large-scale the operation of certificate authority cannot be done with the care necessary to really deserve the trust that cryptographic protocols and system designers put into their operation. The power that a PKI can have may also not be desirable in some contexts. Many application scenarios like control over personal devices or files should not be delegated to a global authority. Security can become a security threat if not applied in a responsible way[Staj02].

6.3.2 Decentralization with a Flat PKI

A flat PKI is a PKI without a hierachy and with no single central CA. The most common concept for flat PKIs is called Web-of-Trust[ReSt99]. In a Web-of-Trust peers are connected (at least virtually) with other peers that they personally trust. They may also rate their trust into one link, but that is an optional concept. If the graph of the Web-of-Trust contains only one connected component all nodes can authenticate each other via paths in the graph where each link represents a trusted relation. The basic assumption of the approach is that trust is transitive. Given this transitivity, authentication between any two nodes A and B can be achieved as, say, A trusts C, C trusts D, D trusts E, and E knows B.

There are some issues. First, it is clear that finding a path in this social trust graph to achieve authentication is expensive. Routing in this graph does not scale and one needs to scan the graph like the common single-source-shortest-path algorithms[1].

The second and even more important criticism is that trust is not transitive. If one believes that A is A, then one may still not trust A's ability to judge others. Even if that is the case, what about the nodes that A knows? Typical Web-of-Trust concepts like PGP limit the length of the paths that are trusted and compute trust ratings that are lower the longer the path and the less trusted the links that were used. However, the issue remains that this approach makes nodes to authorities despite the fact that the nodes that want to authenticate each other do not necessarily trust these nodes.

This dilemma can also be seen as a trade-off. If one uses only short paths that are more trustworthy, each node can only authenticate with a small number of other nodes. If on

[1] Like the algorithm of Dijkstra

the other hand long paths are chosen, nodes may be able to authenticate but are less sure that this authentication is correct. This dilemma as well as the complexity of finding the corresponding trust paths allow to question the scalability of Web-of-Trust. This is relieved if communication is primarily along social links like in email where PGP shows that Web-of-Trust can work. However, one may speculate that theses issues are among the reasons why email encryption and authentication are not very user-friendly and hardly used.

6.3.3 Other Decentralized Solutions

The flat PKI assumes that there may be some way to find a trusted path between peers. Other decentralized solutions follow a different approach. They assume that an individual may not identify an entity and decide about it alone, but that a group shall do this operation. They are usually proposed by cryptographers doing research on secret sharing and multiparty computation. The basic idea is that a new node wishing to join the network has to apply to all members of the current network. If they all agree, the joint cryptographic operation will succeed and the new node receives its identity and keying material. Subsequently, the network has to rekey and adapt to the new size. Such proposals have been made for pure cryptosystems as well as for settings with ad-hoc networks[SaTY03, NaTY03, Pede91].

Secret sharing[Sham79, CGMA85, Feld87, RaBO89] can be verifiable, but that requires a centralized key distribution or the use of broadcasts. Verifiable means that nodes can detect wrong values and discard them. Pure decentralized secret sharing does not achieve this property. This is related to a general problem that all these approaches have. They often do not allow that attackers or unreliable peers are in the system. Furthermore, they are usually expensive and might involve many broadcast operations, in particular to circumvent the nonverifiability in decentral cases. From the practical point-of-view they do not seem too realistic, in particular they lack the argument why the whole network or at least a subgroup should have information about the new peer that a single peer does not have. Each peer in the group has to decide, but on what facts? The main argument for using it is that trust is distributed and attackers need to persuade more peers.

6.3.4 Decentralization without multiparty interaction

In the Web-of-Trust case as well as in the solutions in the last section, for the authentication of a peer A to a peer B there are many peers involved and their information is used to come to a conclusion. This section is about ideas to operate without third parties. Peer A and B authenticate without further help. Of course, there comes a limit with these approaches. There are cases of uncertainty that allow an attacker to infiltrate the system.

The basic problem is authentication the first time when two entities meet. A and B do not know each other. In the Resurrecting Duckling or Baby Duck model [StAn99] A and B hope that at the moment of their first interaction no attacker is present. They exchange keys and subsequently can be sure they meet the same entity again when they see the corresponding key. If the attacker was present at their first contact, the attacker can stage a man-in-the-middle attack. This attack works as long as the attacker is present. If not, the peers will try to authenticate with wrong keys and detect an error[2]. The baby duck model is not academic, e.g. it is used in SSH[YlLo06] where the server certificate in local environments is unknown to the users and learned during the first contact.

Zfone[Zimm07] is an approach that extends the baby duck idea. It assumes that users communicate via a voice channel (authentication for Voice-over-IP) and that the attacker

[2]Not necessarily that there was an attack, although for typical use-cases like SSH this is quite likely.

cannot interfere with the voice conversation. To avoid the man-in-the-middle the users need to read short authentication strings that are derived from a Diffie-Hellman exchange. The numbers would differ in case of attack and the attack would be detected. There are still ways to attack if the users are unknown to each other. An attacker could be learned as authentic if he interacts convincing enough. Zfone's early adaptation of RTP called zRTP also contained a weakness[GuSh07]. The basic idea of Zfone is nonetheless very good and is useful to mitigate first-contact problems (in particular the problems on protocol layer).

6.4 Fundamental Issues

In this section, we briefly discuss the fundamental problem underlying authentication. If one looks closer at the previous sections, one may notice that in the decentralized cases one tries to gain information from nothing and agree on something without existing information.

6.4.1 Identity

Authentication is often simplified to the use of some key to ensure that everything is fine. This ignores that authentication is used to connect something else to the key and, in addition, to prove the knowledge of the key to show that this connection is valid for the current communication.

Usually, we say that an identity is connected with the key. However, there is least one further mapping that is usually used implicitly when applying authentication. Authentication proves the relation of the corresponding key with a representation of an identity. This identity can be seen as a string. We may call this identity, although itself may not satisfy the true semantics meant by the term identity in most scenarios. In most cases when humans are involved, identity refers to identities outside of the technical system. This may be the real human being or his real-world name or further data. As a consequence, there is a mapping of these real-world entities to the identities in the system. This mapping is beyond the control of the system, but its components and users need to believe in it. Their decisions for subsequent security goals and also their understanding of authentication are related to their belief into the assumed real-world entity. This assumption may be wrong when another real-world identity is behind the presented identity string.

This means that we either have to trust the authority to operate perfectly and that we have meaningful identifies – from an identity we can immediately tell who it is. Otherwise, we need to learn that the given string representation is in reality a particular entity and after some time we accept this representation as his pseudonym. In the latter case, the authority only protects identities from subsequent impersonation, but does not provide and certify meaningful identities.

6.4.2 Defining Authentication

Most people (at least with computer security expertise) have an intuitive understanding of authentication. It is related to proving the identity of an entity (entity authentication) or the origin of a message (message authentication), see [MeOV97]. Someone proves with these means of authentication that he was involved and this shows that either a message was sent or an entity owned by him.

While this understanding may seem enough, we believe that for the architecture and design of systems that use authentication and security protocols one needs be aware that defining authentication and detecting when authentication was really achieved in a system is more challenging. It usually involves that parties agree on the values of certain parameters. Thus, during the steps of the protocols involved all parties conclude the same for a certain

set of important parameters. There may be other parameters where they may not agree. Injective agreement of Lowe[Lowe97, Lowe99] is an example for a more formal definition of authentication. The benefit of such definitions is that formal methods[Crem06, AVIS07] may be used to understand protocol and system behavior.

We stated all this as it shows that authentication cannot be taken lightly and simply using some authentication protocol may not lead to a secure authentication for the system. In particular authentication and identities may have different meanings on different protocol layers and for different applications.

6.4.3 Limits for Authentication Mechanisms

As we have already seen, authentication is related to agreeing on or even proving something. Peers may agree on authentication results, but within the agreement one may still be prone to errors and false results. Secure authentication does not rely on speculation, but on facts and subsequent proofs. Peers have an initial set of facts (axioms) they believe in. The peers perform some operations and as a result new facts are proven. So, the authentication of two peers is a process that uses initial information together with deduction to create the then newly proven facts that their corresponding identities are their correct identities. If no common facts exist that relate to the two peers, the authentication can only succeed if the peer's identities are tautologies (true no matter what axioms are chosen) in the system. For Peer-to-Peer authentication the lack of information is a rather common case, say peer Alice knows noone in China and peer Cheng does not know Alice. This lack of information is common in networked systems where possibly anyone could participate while everyone can only have enough information about a small fraction of potential participants. In today's networked applications it is common to transfer the necessary information out-of-band, e.g. personally exchange Skype IDs or mail addresses.

A general analysis of the problem was provided by Boyd[Boyd93] in 1993. Authentication is related to the problem of a secure channel. For a secure channel between two entities, the two entities possess keying material that they can use for authentication or confidentiality. An informal description of the basic results is given in the following list:

- A secure channel between two entities can only be established if a secure channel already exists[3].
- If both entities A and B have a secure channel with a third entity (they trust), then this context can be used to establish a secure channel between A and B.

The consequence is that secure authentication can only be achieved if a context exists. Networks have to provide the means for any two participants to authenticate. The most straight-forward way is to have a central authority that knows all entities and can provide the necessary context for all combination of entities A and B. We can also conclude that all approaches that try to establish a secure channel from nothing can only establish an insecure channel.

Please note that relying on out-of-band information can again be seen as a context. This context, however, may not be available at the time when it is first required. Moreover, there is a risk that the information that was gained is wrong. As we already mentioned when discussing the centralized PKI case, their operation is not as riskfree as security theory assumes. Context in PKIs is also created on the basis of real-world interaction or real-world context. In the end, their real-world operation raises similar issues and introduces the risk of false information. As a consequence, authentication and subsequent security measures always include some kind of risk assessment.

[3] in the sense that there is keying material from the old channel to relate the new channel to the old one

6.5 Exploiting social clustering for authentication

We learned that decentralized approaches for authentication suffer from drawbacks that make them infeasable. Centralized authentication contradicts the Peer-to-Peer paradigm and has problems of its own. It provides secure authentication because it simply defines all the problems away with its assumption of an omnipotent and well-known authority. At first there seems to be no way out of this dilemma as a trust anchor seems to be missing in all decentralized approaches. However, this neglects the physical world, which can be a substitute for trust anchors in the virtual world. The most accessible part of the physical world for networked systems is human relation. In the next subsection we briefly discuss social networks and their implications. As further consequence, we introduce in the subsequent section how we want to exploit these properties.

6.5.1 Social networks

Computer-based social networks have been a hype in recent years. Facebook or in Germany StudiVZ are prominent examples. Their basic idea is that users create their profiles with information about them. Friends can become virtual friends as well and interests can be used to find other potential friends or collaborators. When meeting someone, social network IDs may also make it easier to exchange IDs in a human-friendly way without explicitly writing an address down and exchanging paper. There are a variety of such social network sites. However, we are not interested in them in particular. Social networks are networks that are formed on the basis of human-level relations instead of network relations.

A common denominator for most social networks is that their structure follows a Small-World graph[Milg67, BaAl99, WaSt98]. In theory, a graph is a Small-World graph if two properties hold. First, the average path length is small (= only slightly larger than for random graphs). Second, the nodes in the graph are clustered. This means that neighbors of a node in the graph are likely to be neighbors as well. Many friends of a person are also friends. There may be subgroups of friends that form cliques. Friendship relations are not the only example for such graphs. Companies, departments, or other small subgroups may form cliques (in the sense everyone knows each other). The graph may still be highly-meshed for larger groups or a complete company. Another example would be based on families or clubs.

We therefore conclude that clusters are a natural phenomenon and that within a cluster there is a small level of trust as it is more or less a friend-to-friend network with small diameter. We now propose what we call Domain-based Authentication as our way of exploiting social network structures.

6.5.2 Domain-based Peer-to-Peer and Authentication

We have seen that human relations and networks that are generated by such human interaction follow small-world graph structures. In these graphs peers that know each other on human level cluster in certain areas of the network. These clusters can be seen as subnetworks and we call them domain in the rest of this section. Each domain can be a Peer-to-Peer network of its own and is connected to the overall network. Previous work by Mislove and Druschel[MiDr04] proposed a domain-based Peer-to-Peer approach to introduce administrative control in Peer-to-Peer networks.

The idea of Domain-based Peer-to-Peer and authentication is that peers belong to domains that are related to their social clusters. These smaller graphs can be more easily administered and security is based on local real-world knowledge and interaction. Each domain is responsible for itself and could use any organizational form it likes. As interface

to the rest of the Peer-to-Peer network, it provides a domain authentication server (DAS) that operates as responsible entity for the domain. The domain could distribute the server operation, but from the outside it looks as one server.

The DAS manages its domains and its members. It has a public/private key pair to represent the domain. It acts as a trusted third party for the domain in the protocols for authentication and authorization. To increase scalability, it manages relations to other domains. For domains it has already seen, the DAS knows the public key and also stores data on the experience and the way of the contact. There may be some domains that were preconfigured by the administrators of the domain, e.g. friendly domains. These domains may be a-priori trusted or distrusted. Other domains are only known from interaction in the network. Here, the authentication starts with untrusted results. Experience may either be good and increase trust or be bad and reduce trust. The management of trust between domains instead of peers is more scalable than the one with peers only. Furthermore, it puts pressure onto domains to only include and support legitimate honest peers.

To deal with this open nature of this domain-based Peer-to-Peer approach, we introduce a form of risk assessment into the authentication process. The system requires that within each run of an authentication protocol, the DAS tells its peer its information about the other domain. We call this the *Risk Assessment Token*. This token does not include a single trust value, instead information about how the domain knows the other domain is included as well as derived aggregated values on previous experience.

The risk assessment enables the peers to decide on the situation instead of letting the server decide. This may seem contrary to common centralization of security. However, as the authentication infrastructure and domains should be used beyond a single Peer-to-Peer use-case, different applications may use the same network and their different requirements may lead to different assessments of security.

As example, in a game we might be more interested in the fair-play of the user and the experience than into the real-world authentication. On the other hand, for messaging we want authentication, but if it is Voice-over-IP we may use methods like Zfone to improve security and can thus more easily allow yet unknown peers. For business transaction, authentication and trust must be ensured. As a consequence, the DAS helps the Peer-to-Peer application with information and it decides on authorization.

6.6 Trust - Confidence and Reputation

In traditional protocols there is always a trusted party that helps all entities to ensure their security. Identity Federation operates on the assumption that there is a circle of trust. The idea is similar to a web-of-trust. Customers of another company are also accepted with their IDs from the other company. The basic assumption here is that all domains trust each other or that there is a central point of trust.

Trust has two notions in this context. Both need to be achieved for authentication and authorization of entities. The first aspect is the trust into the entity. This can also be called reputation. The trusted entities need to have a high reputation and all necessary entities need to believe this. Other entities need a reputation to allow a subsequent authorization. A high reputation indicates that this peer is well-behaving and good-natured. Reputation is built from behavior in the past and usually high reputation comes from many interactions over a not too short period of time. The other aspect is that the authentication was not fraudulent. If an entity is not sure about an authentication, then it cannot achieve other goals like even the building of trust between two entities or domains.

We have, thus, seen that trust in the context of authentication is manifold. Furthermore, here in the Peer-to-Peer setting, identity may not only be a question of authentication,

but is coupled with the question of authorization. Initially, two entities or domains cannot trust each other and anyone could provide fake information. To overcome this state we need to build up trust. Trust can only be built on experience if it is not there a-priori. This is the major reason for this coupling of authentication and authorization as atfirst authentication is not trusted enough to be secure enough. On the other hand, one needs a set of insecure interactions to establish trust into other entities.

6.7 Establishment of Trust

Trust is usually generated by experience. Trust is also context-dependent. In some cases it is sufficient to meet for the first time and present plausible data. The baby duck authentication model uses this assumption. In real-life we might trust a police officer to be one due to his uniform. This is quite similar. However, in critical cases we might not trust the uniform alone, but need more appropriate means. In electronic terms this would be a certificate or in the real-world example an ID card. For real friendship and its associated trust one will not gain full trust and friendship immediately the first time when one meets[4].

Trust can be in an entity or groups of entities that are similar enough to be evaluated together. A social cluster or domain can be such a group. As groups are larger, there is more interaction that can be used to built trust between groups than between individuals. Nonetheless, we propose to include individuals in the statistics as authorization may need individual information.

For the trust establishment, besides collecting and aggregating feedback, one needs to decide when to trust. When being in contact for the first time, one cannot have trust already. This communication is then rather untrusted, but one might argue that it should include a chance for gaining trust. Someone who misbehaved may be distrusted and considered worse. Although one should not overdo the impact of this difference as whitewashing with a new identity may make the difference obsolete. Trust is gained from positive experience and the number of experiences should have an impact on trust. Trust should rise with the number of positive contacts. Different feedbacks should be weighted as their impact may be different (e.g. the trust set by an administrator outweighs the feedback of one arbitrary contact by far). Naturally, one would expect trust to rise faster at first with the first experiences and then an improvement will become harder the better the trust already is.

Trust is usually normalized to a range of $[0, 1]$ where 0 is no trust or even distrust and 1 would be full trust. An alternative would be to expand it to $[-1, 1]$ to make 0 a neutral value and describe distrust as negative value and trust as positive. The mapping from the weighted number of contacts to this fixed range can be done by various functions. Sigmoids are a good choice. Sigmoids are functions that output any input to values in a fixed range like $(-1, 1)$ where the output increases with the input and the derivative increases to some maximum (e.g. at 0) and from then on decreases. This leads to forms that look like an 'S'. Examples for sigmoids are $f(x) = tanh(a * x)$ and $f(x) = 1 - e^{-a*x}$. Figure 6.1 shows a sigmoid with the positive and negative experience summed up and the sum being related to a trust value by the function. The scale of the experience axis is arbitrary. The figure shows a possible set of regions within the sigmoid, from distrust for all values lower than for unknown entities, untrusted as initial form of trust, some trust for entities with already more than only few positive experiences, and trusted for entities with a lot of positive experience. The particular scale, a in the given formulas, depends on the

[4]To a certain degree, humans may decide already within the first few seconds, but this is usually the wish to build trust and friendship and not the final establishment of it.

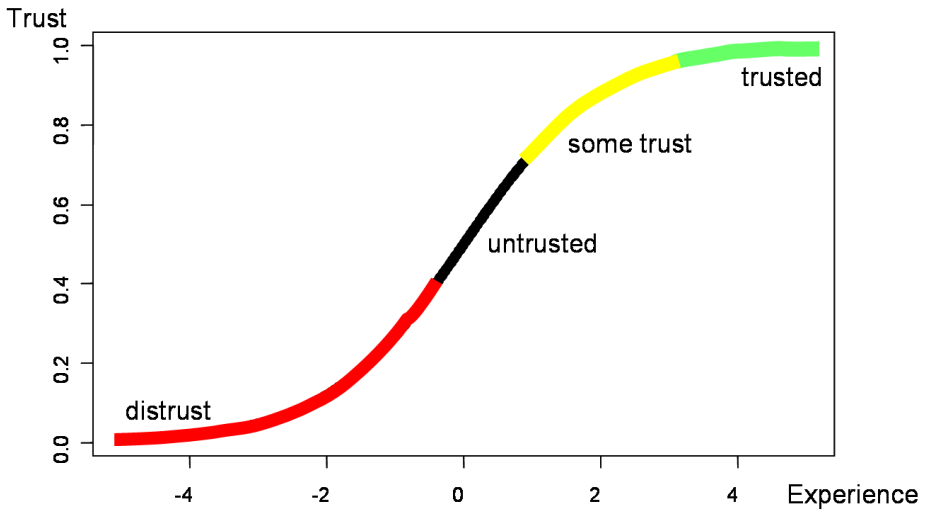


Figure 6.1: A sigmoide function that connects trust and the intensity of experience (number of possitive contacts, ...).

system and its use-cases. The feedback should be weighted and except for administrator settings, trust should not be built from only one or few contacts.

In the following sections, we describe a method that avoids to directly build a trust value. However, individual applications will be informed about various parameters and can then weight them according to their needs and then decide on the basis of achieved trust value. Similar mechanisms also seem to be reasonable for aggregation as simple peers or arbitrary applications should not be confronted with raw data that might be specific for the local environment or may need to be adapted when the usage changes.

6.8 Protocols

In this section, we introduce two authentication protocols for the Domain-based authentication, including the transfer of a risk assessment token from domain servers to their peers. We also describe a protocol to provide feedback back from the peer to its corresponding domain server.

6.8.1 Risk Assessment Token

The authentication protocols in a Peer-to-Peer setting without global trusted entities need to deal with cases where fully secure authentication is not possible due to limitations of trust. Usually, the entities and the domains may not yet know each other and no context can be used. Even if they know each other from a previous interaction, they may not be sure about it.

In the subsequent authentication protocols the domain authentication servers will send their peers at a certain step in the authentication process a risk assessment token where they inform the peers about the state of the authentication and provide valuable information also for subsequent authorization.

A risk assessment token should include the following information

- **Date:** This field is just to indicate the freshness of the information. Within the protocols this is not necessary as cryptographic protocols have to ensure freshness themselves.

- **Routing:** This field includes information about the security of the lookup or contact between the domains. This may be IP or Multipath or that the DAS did not lookup, e.g. because it was contacted by the other DAS.
- **Identities:** Specify if the peer and/or domain IDs are cryptographic IDs or not.
- **Trust based on:**
  - **A-priori:** A human authority (administrator, peer) manually configured the trust.
  - **Web-of-Trust:** The other domain is known via other trusted domains. This also requires to state a distance in the trust graph for the risk assessment.
  - **Interaction:** The trust was established via many previous contacts.
  - **VoIP-secured:** Zfone mechanism used
  - ...
- **Statistics for domain:** This field is used to inform about previous contacts that describe the experience.
  - Number of Contacts
  - Positive Feedback
  - Negative Feedback
  - ...
- **Statistics for peer:** This field is used to inform about previous contacts that describe the experience. This is similar to the domain description

To avoid information overload and potential privacy issues the reports will hand out information less fine-grained. A potential example would be to reduce the numeric fields to fields with the values: no / low / middle / high / sufficient / perfect. In terms of trust one might say: distrust, untrusted, some trust, trusted.

The Risk Assessment Token allows to build trust and do statistics on the more scalable domain level. However, the final decision remains with the peer and its requirements. The token and systems that use it are thus not limited to one particular Peer-to-Peer system, but can be seen as a generic support for multiple systems.

6.8.2 Notation

There are some slightly different ways to describe crpyptographic protocols. Besides the formal notation which we will now briefly describe, it is important to notice that operations beyond the sending of messages are usually required and need to be described in the accompanying text. Some of this may be implicitly clear, some operations are not. The following notation can be found in [BoMa03].

The identity of a principal is denoted by a capital letter. An arrow indicates the process of one principal sending a message to another. A symmetric key is denoted by the letter K with an index indicating between which principals the key is shared. We denote a public key as PK_X, where X indicates the principal to which the key belongs. Encryption with a symmetric key is written as $\{m\}_K$. Encryption with a public key is denoted by $E_X(m)$. A signature with a private key is denoted by $Sig_X(t)$: token t, signed with the private key of agent X. Nonces are denoted by a capital N with an index.

All encrypted parts of messages are also integrity-protected without explicitly adding this to the notation.

6.8.3 PDP-A - the authentication protocol of Holz

The first authentication protocol was developed by Ralph Holz under our guidance in his diploma thesis[Holz07]. It was presented in [HNHC08].

Prerequesits

The peers and their Domain Authentication Servers (DAS) need to know each other and within their domain they have shared a symmetric key and know their respective public keys. This process of exchanging keying information and relations in a domain is considered out-of-band from the view of the authentication protocol. The domain (represented by the DAS) needs to be careful in choosing and accepting its peers. Appropriate protocols have to be used.

So, a peer X and its DAS S_X share the secret symmetric key K_{XS_X}. S_X uses the cryptographically secure hash function h to calculate $h(X.PK_X)$ and signs this with its private key. $Sig_{S_X}(h(X, PK_X))$ will be referred to as a *Public Key Token*.

Let the peers be A and B. Before they use the authentication protocol PDP-A to prove their identities and to assess the risk, they also need to exchange keys. This is not done securely, but simply a setup for the subsequent authentication protocol. B initiates the communication.

$$B \rightarrow A: \ (B, A, PK_B, Sig_{S_B}(h(B, PK_B)), N) \qquad \text{(Key Exchange Query)}$$

$$A \rightarrow B: \ (A, B, PK_A, Sig_{S_A}(h(A, PK_A)), N) \qquad \text{(Key Exchange Response)}$$

The key exchange is necessary so that all relevant keys are known. The interaction with the DAS of each domain of the peers will prove that the keys are the right ones. Each peer will prove by knowing the corresponding private key that it is the principal with the provided identity.

Protocol Specification

The protocols follows the guidelines by Abadi and Needham [AbNe96] wherever possible. Every message is encrypted. The number of potential attack vectors is reduced for such protocols. As we will later note, this version of the protocol was found secure by an uptodate model checker.

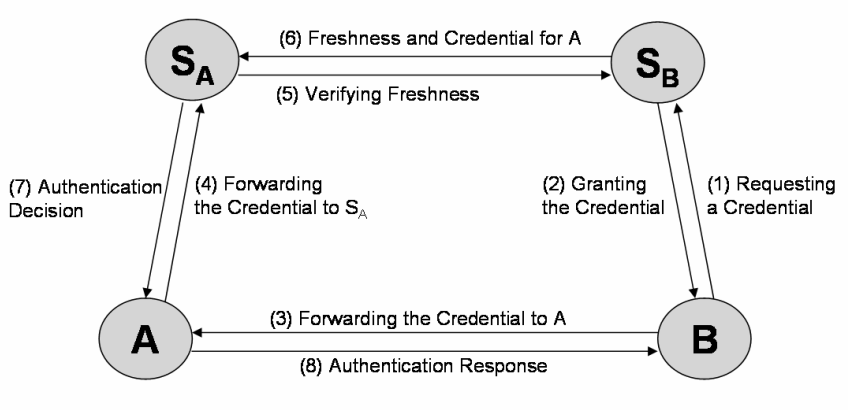


Figure 6.2: The message flow of PDP-A (protocol of Holz).

In the following, we descibe each message and its fields. Figure 6.2 shows the basic structure of the protocol. Peers cross domain borders first between the client peers. Both Domain Authentication Servers will communicate after that and final results are shared between the peers to verify the authentication and establish a key.

Step 1: Requesting a *Credential*

B is the initiator of the authentication run. Its domains is D_B. The first step remains within the domain. B speaks with its Domain Authentication Server S_B to acquire a fresh credential for the communication with A from domain D_A. The complete message is encrypted and integrity-protected with the shared key of B and S_B. Sender and receiver (B and S_B) are explicitly stated in the first fields of the message. This will be continued throughout the whole protocol run to avoid potential attacks. B then states who is to be contacted, which is A. In the message A is defined by its name A, its public key PK_A and its Public Key Token $Sig_{S_A}(h(A, PK_A))$. Finally, B provides its nonce to detect freshness of the protocol run later-on.

$$B \rightarrow S_B : \{B, S_B, A, PK_A, Sig_{S_A}(h(A, PK_A)), N_B\}_{K_{BS_B}} \quad \text{(Message 1)}$$

Step 2: Granting the *Credential*

The Domain Authentication Server S_B replies with a freshly-signed credential. This message also include the Risk Assessment Token for the domain D_B about the domain D_A and the peer A. If the domain D_A is yet unknown, S_B may need to determine the public key of D_A and therefore lookup the domain and key. Other possible cases include a-priori knowledge and trust or a certain level of trust based on experience. Since the network is considered to be multi-purpose, S_B provides this information to B and the software decides if this is in consent with the security requirements of the application. Trust establishment also needs to allow untrusted cases in order to built it.

The complete message is protected with the shared key K_{BS_B}. It includes sender and receiver as first fields. With the third field, S_B signs the communication request of B. The hash includes B and A, PK_B (the public key of B) and the fresh nonce N_{S_B}. We consider this third field the credential for B to establish the session in consent with S_B. As fourth field, N_B is returned to B and provides freshness. N_{S_B} needs to be sent as well. The last field finally contains the Risk Assessment Token.

$$S_B \rightarrow B : \{S_B, B, Sig_{S_B}(h(B, A, PK_B, N_{S_B})), N_B, N_{S_B}, risk_{D_A}\}_{K_{BS_B}} \quad \text{(Message 2)}$$

Step 3: Forwarding the *Credential* to A

With the last message the peer B received the Risk Assessment Token for the desired communication. B may now decide to stop the authentication run if this does not satisfy its security constraints. For the rest of the protocol we now assume that B accepts the situation and continues to authenticate with A.

This message is now secured with the public key of A that B knows from the previous key exchange (see Prerequesits, Section 6.8.3). The first fields are sender and receiver of the message. B informs A about its Domain Authentication Server S_B, and forwards the credential $Sig_{S_B}(h(B, A, PK_B, N_{S_B}))$ with the necessary information, the nonces N_B and N_{S_B}.

$$B \rightarrow A : E_A(B, A, S_B, Sig_{S_B}(h(B, A, PK_B, N_{S_B})), N_B, N_{S_B}) \quad \text{(Message 3)}$$

Step 4: Forwarding the *Credential* to S_A

A now asks its Domain Authentication Server S_A to evaluate the request by B from the other domain D_B. It expects S_B to either know the public key of S_A or to acquire it. The basic operation of this message is to forward the request of B to S_A.

The message is protected by the shared key K_{AS_A} of A and S_A. The first two fields are sender and receiver of the message. The next fields are about B and its domain, respectively the Domain Authentication Server S_B, the public key PK_B of B as well as the credential signed by S_B. The final fields are the necessary nonces, N_{S_B} for the communication of S_A and S_B and nonce N_A for the return message to A.

$$A \rightarrow S_A: \ \{A, S_A, B, S_B, PK_B, Sig_{S_B}(h(B, A, PK_B, N_{S_B})), N_{S_B}, N_A\}_{K_{AS_A}} \quad \text{(Message 4)}$$

Step 5: Verifying Freshness

S_A can now check the credential presented by B and issued by S_B. It therefore needs to know the public key PK_{S_B} of S_B. If it does not know it, it will acquire it with a lookup in the global Peer-to-Peer network (or Internet). The protocol is agnostic to this lookup process. This information will, however, be used in the Risk Assessment Token from S_A to A.

The interaction of S_A with S_B is necessary to ensure freshness. Furthermore, S_B will provide a credential for A to authenticate to B in the end.

The message is secured with the public key of S_B. The first two fields are sender and receiver. The next two fields are initiator B and responder A. Freshness and legitimacy of the request is shown with nonce N_{S_B}. Finally, S_A adds its nonce N_{S_A}.

$$S_A \rightarrow S_B: \ E_{S_B}(S_A, S_B, B, A, N_{S_B}, N_{S_A}) \quad \text{(Message 5)}$$

Step 6: Freshness and *Credential* for A

S_B recognizes the authentication run with N_{S_B} and can check the validity of the run by also checking if B is initiator and A is responder. Otherwise, the request would be invalid and has to be be ingored.

This message is secured with the public key PK_{S_A}. The first fields are sender and receiver. S_B signs a credential for A that is based on A and nonce N_{S_B}. The final field is nonce N_{S_A} that signals the freshness of this return message to S_A.

$$S_B \rightarrow S_A: \ E_{S_A}(S_B, S_A, Sig_{S_B}(h(A, N_{S_B})), N_{S_A}) \quad \text{(Message 6)}$$

Step 7: Authentication Decision

S_A can now finally fully verify the credential of B and the credential for A.

This message is protected by the shared key K_{AS_A}. The first fields are sender and receiver. The third field is the credential for A by S_B. Nonce N_A is included to ensure freshness for A. The final field is the risk Assessment Token from S_A for A. It includes the evaluation of S_A for the communication.

$$S_A \rightarrow A: \ \{S_A, A, Sig_{S_B}(h(A, N_{S_B})), N_A, risk_{D_B}\}_{K_{AS_A}} \quad \text{(Message 7)}$$

Step 8: Authentication Response

A evaluates the Risk Assessment Token from the last message. If the indicated trust meets its requirements, it will continue with the protocol. Otherwise, it will stop the protocol run. The information in the previous message also authenticates B as its server S_B signed the corresponding information and nonces. A generates the new session key K_{BA} between A and B. This key will now be sent to B together with information to authenticate A to B.

The message is secured with the public key PK_B of B. This key is now verified to be correct to the degree specified in the Risk Assessment Token. If there is full trust between S_A and S_B, there is full trust into this key. The first two fields are sender and receiver of the message. The third field is the credential for A that was issued by S_B. With the knowledge of nonce N_B A shows the knowledge of its private key as well as being a message from the current protocol run. The last field is then the new session key.

$$A \rightarrow B: \; E_B(A, B, Sig_{S_B}(h(A, N_{S_B})), N_B, K_{BA}) \qquad \text{(Message 8)}$$

B can verify the authentication of A as it applied its private key in Message 3 for nonce N_B. The signature of S_B ensures the support of its Domain Authentication Server S_B its cooperation with S_A.

Perfect Forward Secrecy

The basic form of PDP-A (protocol of Holz) is not forward secure. Perfect Forward Secrecy allows the establishment of a session key that is not exposed even if keys used in the authentication protocol are broken. This is usually solved with a Diffie-Hellman Key Exchange. The exchange can be done after the protocol and is then independent of the protocol. The other option is to include it. Message 8 needs to be modified in that case. It then includes the Diffie-Hellman values instead of the key K_{AB}. B needs to reply with its Diffie-Hellman values in a ninth protocol message. Both entities can now determine the key from the shared values.

$$A \rightarrow B: \; E_B(A, B, Sig_{S_B}(h(A, N_{S_B})), N_B, g_A, p, DH_A) \qquad \text{(Message 8)}$$
$$B \rightarrow A: \; E_A(B, A, N_B, DH_B) \qquad \text{(Message 9)}$$

Analysis

PDP-A was analyzed and designed using the AVISPA[AVIS07] model checker. Its final version satisfies authentication according the definition of Lowe (injective agreement) without weaknesses according to the formal model checking. Model checking, of course, operates under perfect assumptions. In this case, that the Domain Authentication Servers already know and trust each other. Any other assumption would lead to a situation where the formal methods would detect the lack of trust into the keys and therefore consider the authentication to fail. This risk is tackled with the risk analysis token that informs the participating peers and allows them to do a risk analysis and decide if this is ok for the application and its current usage.

6.8.4 Reducing the number of messages for authentication – a protocol with 6 messages (Niedermayer)

PDP-A, the protocol of Holz, needs nine messages for the authentication run and this does not include the exchange of public keys beforehand. A basic principle for cryptographic protocols is to send, receive, and compare nonces. This is used to ensure freshness and to determine the protocol outcome. Each entity needs to send out a nonce and subsequently receive its nonce from another entity. For a four-party protocol this corresponds to at least seven messages. Each entity needs to send at least one message. Thus, the sending out of nonces requires four messages. In the first message, there can be no receiver for a nonce that was already sent. This is also true for all messages where the receiver is contacted the first time in the protocol run. There can be no nonce from this entity yet. Except for the initiator all entities need to be contacted. As a consequence, at least three messages cannot return a nonce to their receiver. All entities have to see their nonce later-on. So, we need at least four messages where the receiver receives a nonce it previously created. The last of the first four messages that sent out nonces can also be used to return a nonce. Seven messages is, thus, the lowest number for four parties. Other protocol requirements may increase this number and it may not necessarily be achievable for all such protocols.

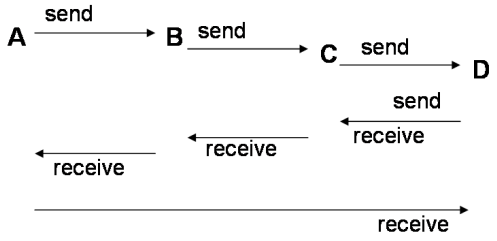


Figure 6.3: A protocol flow for four parties with seven message and freshness ensured for all parties.

These assumptions are based on the use of nonces. Timestamps are sometimes used to replace nonces. Timestamps are a way to eliminate the necessity for all parties to send and also later-on receive a message with its fresh nonce, because freshness may also be ensured by the timestamp. However, at least some parties will have to receive messages after they sent a message in order to see that the protocol run had an effect or succeeded.

In the following we describe a protocol that excludes the authentication servers from being able to decide freshness and only ensures freshness for the authenticating parties. With this restriction we can reduce the number of protocol steps to six.

Prerequesits

The prerequesits are similar to the one of PDP-A (cf. Section 6.8.3). An important difference is that the public keys need not be exchanged beforehand. This is achieved within the protocol. The design is not as clean as the design of PDP-A. An example is the use of message parts that are encrypted and cannot be read by principals that receive and forward it. There are other cryptographic protocols that do this, but this is not in consent with design recommendations.

Protocol Specification

In the following, we descibe each message and its fields. Figure 6.4 shows the basic structure of the protocol. Peers cross domain borders directly in the first message. The message flow is then a circle via the Domain Authentication Servers and a final reply from A concludes the protocol.

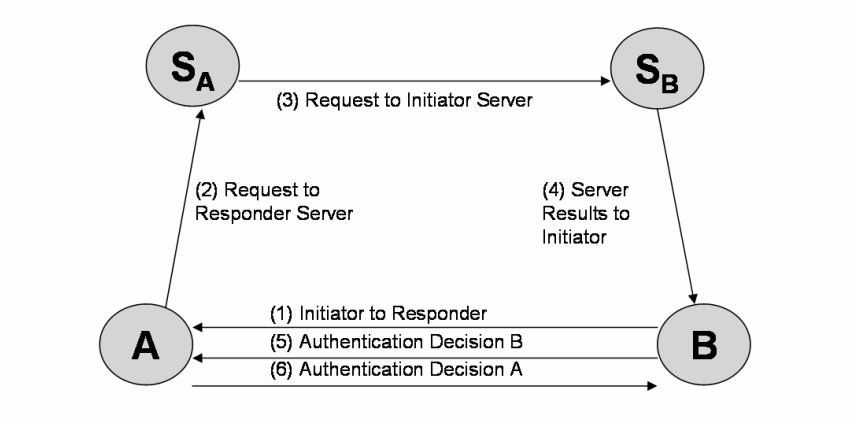


Figure 6.4: The message flow of the message-reduced protocol (Niedermayer).

Step 1: Initiator to Responder

Principal B initiates the protocol. B informs A about its desire to communicate. The message contains two parts. The first is the public information for A about the communication. The second part is an encrypted segment that contains information for S_B and is encrypted with the key K_{BS_B} shared between B and its Domain Authentication Server S_B. The nonce N_B is only in the protected part and A will find out about it later-on and not now.

$$B \rightarrow A: \ B, A, PK_B, D_B, \{B, D_B, A, D_A, N_B\}_{K_{BS_B}} \qquad \text{(Message 1)}$$

The effect is that A now knows that B from domain D_B wants to communicate with A and that B's public key is PK_B.

Step 2: Request to responder server

The responser A now informs its Domain Authentication Server S_A about the request from B from domain D_B. It also forwards the unreadable encrypted part of Message 1, which is B's message to its Domain Authentication Server S_B.

The message is secured with the shared key K_{AS_A} of A and S_A. The first fields are sender and receiver. The next three fields include the information that identify B. The next field is the forwarded part. Finally, A adds its nonce N_A, which is will expect to see later-on in the protocol.

$$A \rightarrow S_A: \ \{A, S_A, B, PK_B, D_B, \{B, D_B, A, D_A, N_B\}_{K_{BS_B}}, N_A\}_{K_{AS_A}} \qquad \text{(Message 2)}$$

Step 3: Request to initiator server

S_A now needs to find out about S_B and its key. It may either know it already or need to look it up via some external mechanism. It also needs to add the Risk Assessment Token for A as it will not be involved in any further message of the protocol.

The overall message is secured with the public key of S_B. S_A signs the message, which we understand here as a hash of the message encoded with its private key in addition to the

content of the message. The first fields are the sender and receiver of the message. The next fields are data of B and A to identify the desired contact. The next two fields are the public keys of A and S_A. The latter is necessary to verify the signature. The next field is the message of B to S_B, which is secured with their shared key. Then follows the message of S_A to A that authorizes the communication and provides the Risk Assessment Token for A. This part is secured with the shared key of S_A and A. The freshness for both encrypted parts is provided by the nonces that B and A can use to check for freshness.

$$S_A \rightarrow S_B: \; E_{S_B}(Sig_{S_A}(S_A, S_B, B, D_B, A, D_A, PK_A, PK_{S_A}, \{B, D_B, A, D_A, N_B\}_{K_{BS_B}}, \{B, A, PK_B, PK_A, risk_{S_A}, N_A\}_{K_{AS_A}}, N_A)) \quad \text{(Message 3)}$$

S_B is the entity to decrypt the message part that was entered by B in the first message. As a consequence it now knows N_B and can be sure that such an information was requested by B. It cannot check freshness of the request, which later-on B will be able to do with the nonce. It can, however, verify the request of A via S_A and can discard the request if it was not from a node supported by S_A.

Step 4: Server Results to Initiator

S_B may not already know S_A and might want to look up its identity and key via the external lookup mechanism. The basic information is already in the message though[5]. S_B checks its information on the other domain and peer and will use this to create the Risk Assessment Token for B.

The message is protected with the shared key K_{BS_B} of S_B and B. The first fields are sender and receiver. The next is the responder A and its public key PK_A. The next field is a part that is forwarded and sent from S_A to reach A. Then follows the Risk Assessment Token for B. The nonce N_B finally provides the freshness for B.

$$S_B \rightarrow B: \; \{S_B, B, A, PK_A, \{B, A, PK_B, PK_A, risk_{S_A}, N_A\}_{K_{AS_A}}, risk_{S_B}, N_B, N_A\}_{K_{BS_B}} \quad \text{(Message 4)}$$

Step 5: Authentication Decision B

B received the information from its Domain Authentication Server S_B in the last message. It can identify the current run with the nonce N_B that it sent in its encrypted part to S_B of the first message. B evaluates the Risk Assessment Token and may now discard the authentication run if the trust is not satisfying its requirements. B also learned A's public key PK_A. It will now use this public key to send the nonces to A and also forward S_A's information A. B can also create a fresh key K_{AB} to share with A.

The message is secured with the public key PK_A of A and signed with the private key PK_B of B. The first two fields are sender and receiver. Then follows the message part from S_A to A. Both nonces are sent in order to verify the run. The final field is the proposal for a shared key K_{AB}

$$B \rightarrow A: \; E_A(Sig_B(B, A, \{B, A, PK_B, PK_A, risk_{S_A}, N_A\}_{K_{AS_A}}, N_B, N_A, K_{AB})) \quad \text{(Message 5)}$$

With the message, A can now consider B to be authenticated.

[5]The idea of making a lookup instead of simply using the presented key is that an attacker may be able to send a fake message with a wrong key, but it might not be able to also interfere with the lookup.

Step 6: Authentication Decision A

A received the information from its Domain Authentication Server S_A in the last message. It can identify the current run with the nonce N_A. A evaluates the Risk Assessment Token and may now discard the authentication run if the trust does not satisfy its requirements. It will now create a shared key and sent it with the nonces to B.

The message is secured with the new session key K_{AB} proposed by B. The fields are sender, reciever and the reciever's nonce N_B.

$$A \rightarrow B: \{B, A, N_B\}_{K_{AB}} \quad \text{(Message 6)}$$

B knows key K_{AB} and can now accept the authentication of A.

Analysis

Except for parts of the first message, all messages are completely encrypted. This gives an attacker fewer chances to misuse information from message fields or using parts of messages in replays. The protocol does not prevent the servers from accepting replays or messages with wrong nonces. It is not easy for an attacker to do this and it would involve interfering with cryptographic operations. However, even if an attacker can generate a valid message that is accepted by a server, the operation of the server is not critical. All decisions are done by the peers A and B. Peers A and B detect protocol runs on the basis of their nonces and can thus discard all replay messages.

B recognizes A as authenticated because A knew N_B and therefore also the private key to its public key PK_A. S_A asserts the relation A and its public key PK_A within its message to B in the protocol run and includes a risk assessment for the relation. A recognizes B as authenticated because B it knew N_A. Therefore it had to cooperate with S_A and S_B to determine the value. This combination also assures the relation of B and PK_B as far as indicated in the risk assessment.

Let us look at the freshness for the messages of S_A and S_B. Both principals A and B are assured of the freshness of the message part from their Domain Authentication Server to them as it used the respective fresh nonce (N_A or N_B). B can detect the freshness for S_A as N_B was signed by S_A together with its fresh message to S_B. Only if this was done, S_B would have accepted the message and be able to determine N_B. With the trust into its Domain Authentication Server, it can be sure that S_A participated in the protocol for domain D_A. A can detect the freshness for S_B because its Domain Authentication Server S_A would have only sent this nonce N_A to S_B. B can only have learned the nonce via S_B.

With respect to forward secrecy, the protocol as described is not forward secure. K_{AB} is currently created by B. This could, however, be also changed to a Diffie-Hellman exchange by adapting the last two messages like in PDP-A and then forward secrecy could be achieved. As the Diffie-Hellman exchange can always be done afterwards, we did not include it in the described protocol.

6.8.5 Communication between DAS and its peers

The concept of trust establishment and risk assessment needs feedback from the peers after the contact was established or even after connection teardown. The kinds of possible feedback are rather application-dependent.

Depending on the authentication protocol that was used, the corresponding nonces (N_B, plus N_{S_B} for PDP-A) that both B and its DAS S_B know need to be used to validate the feedback and to connect it to the correct protocol run.

An example message flow is the following. The messages are secured with the shared key K_{BS_B} of B and S_B. In the first message, the peer writes all information necessary to identify the session. The second message is simply an acknowledge message of the server.

$$B \rightarrow S_B : \{B, S_B, A, D_A, nonces, FEEDBACK, time\}_{K_{BS_B}} \quad \text{(Message 1)}$$

$$S_B \rightarrow B : \{S_B, B, A, D_A, nonces, ACK, h(FEEDBACK), time\}_{K_{BS_B}} \quad \text{(Message 2)}$$

The server will now store this message and aggregate the feedback of this message with the aggregation from previous feedback messages. After some time, the server may remove the message from its store due to storage constraints.

A general problem is the question of how much feedback is anticipated. A contact may seem well-behaving and authentic at first, but after some time misbehaviour or fraud is discovered. If the first good feedback for a session is still there it could be replaced with the bad one, including an adaptation of the aggregation. If too much time has passed or nonces are not known anymore, reporting without connecting the report to a particular run may also be an option that a domain may implement. In that case, the server should challenge the peer in order to avoid replays, which corresponds to at least three messages instead of two.

$$B \rightarrow S_B : \{B, S_B, N_B\}_{K_{BS_B}} \quad \text{(Message 1)}$$

$$S_B \rightarrow B : \{S_B, B, OK, N_B, N_{S_B}\}_{K_{BS_B}} \quad \text{(Message 2)}$$

$$B \rightarrow S_B : \{S_B, B, A, D_A, N_{S_B}, FEEDBACK, time\}_{K_{BS_B}} \quad \text{(Message 3)}$$

$$B \rightarrow S_B : \{S_B, B, A, D_A, N_{S_B}, ACK, h(FEEDBACK), time\}_{K_{BS_B}} \quad \text{(Message 4 (optional))}$$

6.9 Adding Cryptographic Identifiers

Cryptographic identifiers are identifiers that are based on a public key that only the corresponding identity knows. It can therefore prove its identity by applying the private key. The most common construct is not to use the public key directly, but to use a hash function to construct the ID: $ID_{entity} = hash(public-key_{entity})$. Cryptographic identifiers are not human-readable and do not allow additional constraints[6] coded into identifiers. So, identifers cannot have network-specific semantics.

As the concept of the trust-rated authentication is to learn identities and subsequently associate names, the identifiers themselves do not need to be human-readable. Therefore, cryptographic identifiers are an interesting option.

With a cryptographic identifier users can directly claim their their ID in the domain, but they cannot show they belong to the domain. With a cryptographic identifier for domains, the public key of the respective domain authentication server is well-known and responsibility for the domains does not have to be learned. The nodes can directly verify

[6] Additional constraints may be keywords for a content like in content-addressable networks.

this. However, nodes still do not know anything about the domain and how much they can trust it.

Similar to trust, names and cryptographic identifiers can be learned. User feedback may add attributes like human-readable names to domain and user identities. Over time we therefore not only establish trust relations, but also connect names and identifiers. Further attributes like 'friend' or 'neighbor' can be learned, but are more interesting for the authorization than for the authentication.

Cryptograhic identifiers themselves cannot be revoked and they can be generated by anyone. Thus, they do not put a limit to accessing the network and still allow the Sybil attack. One may modify the construction to include a shared secret that all legitimate entities know: $ID_{entity} = hash(shared-secret, public-key_{entity})$ or a similar HMAC construction. However, the shared secret cannot be changed cheaply. This may be necessary when a peer has to be excluded. Thus, such an approach is only feasible for few limited use cases where the assumption holds that the secret will remain secret and legitimate entities may not become illegitimate during the lifetime of the network.

6.10 Relation to Architecture and Design

Authentication is a prerequisite for most security goals. Implementing authentication in a secure way usually conflicts with the idea of the Peer-to-Peer paradigm. Peer-to-Peer applications nonetheless have to deal with security and authentication.

In this chapter we presented an approach that deals with the frequent case where no fully respected central authority exists and peers need to establish knowledge and trust over time. Authorization (Access Control) can be based on being authenticated, on the names, on trust and reputation into the owner of a name that was learned by oneself and others.

Figure 6.5 summarizes the decision process for basic authentication methods with respect to the given scenario. The first step is, of course, to identify the situation and the requirements.

The first question (Sec Q1) is about the authentication that is needed. In many cases, systems only need to recognize the same user again. In particular, there needs to be no need to identify a yet unseen user. If all this is the case, it can be sufficent to use cryptographic IDs and authenticate on basis of their private keys.

The second question (Sec Q2) is about the reliability of the users and the size of the user group. If users all know each other and if they are reliable, then user interaction can be used to authenticate a user for the first time and then build a Web-of-Trust. Still, one may also add or be interested in the concepts asked in the next questions.

The third question (Sec Q3) is about the existence of substructures in the user group, social clusters. The question aims to identify if the approach presented in this chapter can be used. Except for the clustering, it also needs that the system is able to handle untrusted cases where authentication is not yet completely secure and authorization may be able to deal with reduced levels of trust. Also in this case, one may still be interested in the next methods and could add them for further security.

The fourth question (Sec Q4) is about the existance of a single or few central elements, that are trusted and that are able to know all entities that will use the system. If this is the case, these central entities can operate as Trusted Third Parties (TTPs) or Certificate Authorities (CAs) within the system. As trusted parties they may also operate as mediators in case of conflicts or as global reputation system. This is also the case when initial authentication has to be delegated to less trusted local entities as with the PDP approach.

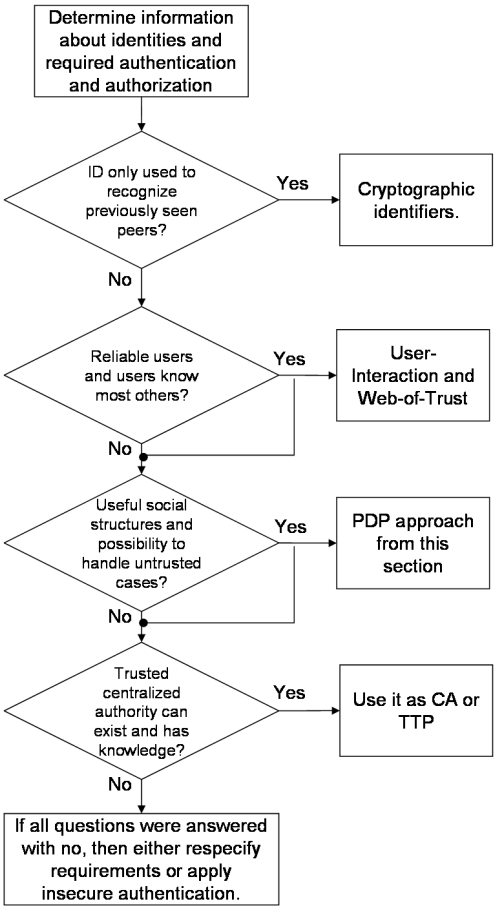


Figure 6.5: Flow diagram for deciding on using which kind of means for authentication.

If none of these solutions fits to the requirements, one may have to rewrite the requirements and change the concept of the system. Another option is to use purely decentralized methods that do not achieve high levels of security, but might still be good enough for small networks or if the system is a target of low value.

Identity is an essential part of the problem of authentication. Applications are very important for the definition of identities and usually their operation limits the degree of freedom one has. One may not use cryptographic identifiers if they already have a semantic or they need to be understood initially. However, their operation also may provide the chance to find appropriate solutions. Our solution of Domain-based Authentication with trust establishment also needs support from the application. The method requires the application to interact and to help build the trust as well as the ability to cope with yet untrusted cases.

7. Anonymity and Peer-to-Peer systems

7.1 Introduction

What is anonymity? One may try to give popular answers, e.g. 'Being there, but not being seen.', 'Under a different name', 'no link to real name, face, or address', etc.

Imagine a system that is perfectly secure (access control, no rag, see Chapter 4) with all of its services working fine. Is this yet fully satisfying? The system stores information about all users. Let us consider, a user wants to perform an action, e.g. establish a communication session. The user will eventually give away some information to the system to enable this operation. For atleast a short period of time, some of this information will be stored in the network. It is quite likely that another user may find some of this information being public enough to be accessible for him or, more generally, for at least some users for some time. But in an anonymous system we want to avoid being associated with information.

Now, in real systems today, the threats against anonymity are increasing. The Internet itself is not anonymous. It is pseudonymous at best, as IP addresses are not really anonymous. Some of them are fixed, others may change, but they give away some information (e.g. user in Germany near Munich). The new data retention law that affects commercial providers from Internet to mobile telecommunication areas is among the threats. However, legislation is only one danger, there are many more illegitimate threats, e.g. by other humans or organizations. Companies may attack critics. Some may not like freedom of speech and threaten others not to use their fundamental human right.

So, anonymity and security is for important communications? Not quite! The best reason for anonymity and security is no reason. Only with wide usage anonymity and to some degree also security[1] can be achieved.

It is not inherent to life, that we easily give away information. In everyday life we have some kind of choice. We can go to a big shop where nobody knows us, we may go to the shop in the neighborhood where we might be known. By using cash nothing about the transaction will be stored and we are hard to trace. We can also use the Internet. Here, nobody will see our face, but the payment and the transfer of goods needs the exchange of data. So, our name or maybe even address have to be given away and will be stored. The

[1] Security can also work among two entities. However, if it is not widespread or common, then one or both of these entities may not be able to use security. Email is the best example. PGP and other software exists, but they are hardly used.

transaction becomes trackable. Of course it depends on what one wants. Why should one care that a person whom one will never meet knows about the transfer and one's name? So, in many cases we have choice. Of course, in some we do not have it. Surveillance is an example.

Technically, anonymity for a client is to request a service without someone else knowing. For a server, anonymity is to provide the service without the identifying IP address and real location. For a peer, anonymity includes both, as a peer is an entity with a client and a server role.

This can be extended to application-level goals and entities. The major motivation for anonymity is to avoid being attacked for doing something. Therefore, anonymity is also strongly related to resistance against censorship. A file or data or a service can be provided without fear and it cannot be removed or stopped. Of course, in reality the systems are not perfect and an attacker exceeding a certain limit in strength may be able to break the system.

7.2 Anonymity

The basic idea of anonymity is to escape from an attacker by leaving the attacker's domain for at least some part of the communication process. Figure 7.1 illustrates this. An anonymity system enables an entity to use resources distant to itself and its attacker. Geographic or topological *escape* is the primary method used in anonymity systems to defeat potential attackers. Free speech on the Internet is one example. To escape the legislation of one country, a user uses webspace or runs a server in a different country with different legislation[2]. So, the basic principle here is to go somewhere where it is possible and as basic Internet communication knows no national boundaries, use it there. Censorship tries the opposite. The basic idea is to get control of the leaks and to make more countries censor something. So, their goal is to make the going away impossible or awkward. Systems like the Great Firewall of China also block traffic with problematic content or traffic from/to a problematic server. This is not perfect, but the basic effect is to keep non-experts away from the services and information.

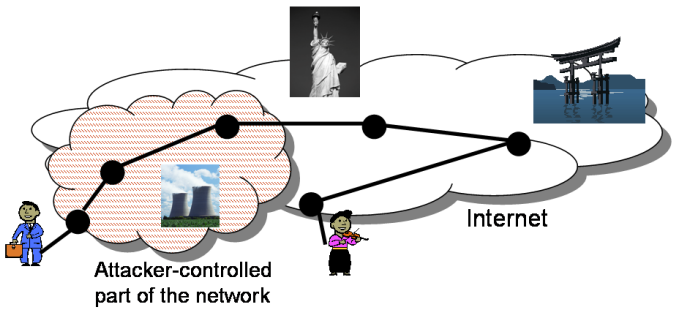


Figure 7.1: Geographic Escape - The communication leaves the attacker-controlled part of the network with intermediate hops hiding the source and destination addresses.

Escape does not necessarily mean geographic escape. A server that is not under attacker control may also help to provide enough anonymity for certain scenarios. Figure 7.2 displays the escape on the basis of trusted nodes within an untrusted attacker network.

[2]It also needs to hide to some degree that this user is the provider as providing a service is often different from using its result with respect to interest of potential attackers as well as legislation. Say, writing something and maybe only accidentally reading something. A user may fear prosecution also for publishing something abroad if this can be proven.

The basic concept in this case is to go to some point of trust and let this point proxy the request. The assumption is that the attacker cannot look into the computers and that they operate in a way that the message flow is not obviously revealing the communication. Anonymity becomes harder when sessions with more than only a few messages need to be protected. All properties (data rate over time, amount of data, message flow, ...) of a particular communication session can help an attacker to identify the same session in other parts of the network and maybe reveal sender and receiver.

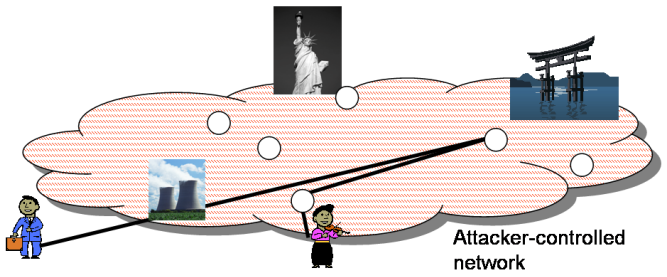


Figure 7.2: Escape via trusted nodes. Once the message reaches them, the attacker cannot easily connect the corresponding incoming and outgoing messages. The attacker, thus, loses the ability to (easily) trace the path from source to destination.

From an abstract point of view, in an anonymous system the participating entities virtually form a set, in which they hide. They may hide behind the set to achieve sender anonymity or receiver anonymity. They may also hide in the set to hide relations like who is communicating with whom. This is called unlinkability. The system performs operations in a way that any entity within or outside of the system cannot decide for a message or stream who initiates and who receives. It therefore has to streamline the communication and its properties so that they become indistinguishable from other parallel communication sessions.

Some of these entities need to serve the system. They provide the operations that make it look like any member of the anonymity set could have caused a message or stream. This could be serving as rerouters for the escape or performing other operations. The message flow can be confused with mixing, i.e. delaying messages and sending them out in a randomized order. One may further hide the message flow in constant cover traffic or add dummy messages or flows. Most of these operations also involve cryptographic operations as message content needs to be hidden from observers as well as other system entities.

One may close the anonymity introduction with some warning remarks. Even though cryptography is used, anonymous systems cannot stop information that is sent in cleartext through the system. This information may be intentionally in there for the receiver. If no illegitimate entity can read it, this is not a problem. However, a common use-case of anonymous systems is to access something outside of the system. This happens via a node that provides the service to exit the system and that performs the outside request. If messages are encrypted and authenticated outside of the anonymous system, this is not a problem[3], but if not, the exit node is in the role of a successful man-in-the-middle attacker. Successful phishing attacks have been performed to demonstrate this threat[Heis07]. Unencrypted logins to websites should not be combined with the use of

[3]Except for the general problem that Certificate Authorities may not operate as perfect in reality as in theory and most of all, users ignoring warnings and clicking them away in their browser or other software. In such cases, a man-in-the-middle can succeed despite the use of appropriate security protocols.

anonymous systems that offer anonymous access to the web. It is a common security problem that users of a security or anonymity service are not aware of how to use it correctly. A recent study on SSL warnings showed that users do not behave right when errors occur[SEAA+09]. As a consequence, the security assumptions of a system do not hold anymore, since in the SSL example invalid certificates are accepted and the protocol proceeds. This is also a threat to secure connections via anonymity systems as such attacks by an exit node may trigger a similar user behavior.

7.3 Onion and Garlic Routing

In this section, we present and discuss two methods for the geographic as well as topological escape. They are Onion Routing and Garlic Routing. Onion Routing is the classic concept that is widely used. Garlic Routing is an extention that to our knowledge is yet only used by the Peer-to-Peer anonymization system I2P[anon], which we will discuss in Section 7.4.

7.3.1 Onion Routing

Variants of Onion Routing can be found in most of today's anonymous systems, e.g. Tor[DiMS04]. In its basic form, Onion Routing simply stands for layers of encryption that intermediate routers peal off by decrypting one layer per hop, see Figure 7.3. It is clear that any system with multiple hops needs such methods to make a message untraceable[4], i.e. look different on each hop.

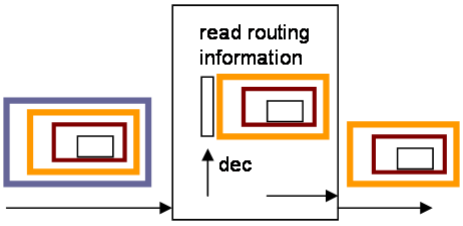


Figure 7.3: An Onion Router processes a message. It decrypts its layer. It will see a readable header and an encrypted body. It will send the body to the next hop according to header and its state.

The differences between all these approaches are in the key management and usage as well as in the way messages are processed. A mix cascade also uses onion encryption, but there is no path setup and messages are randomly delayed. This is usually not the case for pure Onion Routing systems like Tor. There are also differences in how peers get to know other peers and the rerouters.

A path, also called tunnel, can be built in various ways. One could select a series of rerouters and within one message built the tunnel. The node therefore onion encrypts a tunnel create message with data segments for all rerouters on the path. For each rerouter the message needs to provide the key establishment (e.g. shared key encrypted with public key of rerouter) and information about the next rerouter on the path. Besides establishing cryptographic keys, a connection context and path ID may also be established. Furthermore, connections (e.g. TCP) may also be established in the underlay.

The other variant is the telescoping path setup. In that case there is no separation of setup phase and usage of the path. A normal message passing a rerouter will include

[4]There are exceptions. Crowds[Mich98] is an example that follows the iterative exposure approach where each hop can see the cleartext, but does not know if the predecessor is the originator. Messages look different on each link due to different hop-to-hop encryption with SSL.

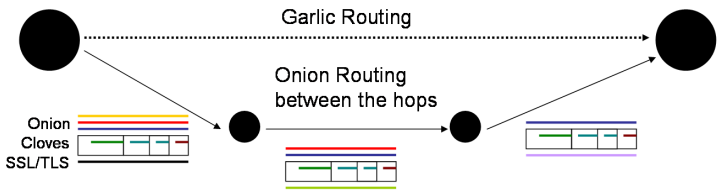


Figure 7.4: Garlic Routing may be combined with Onion Routing. The Onion encryption is used for the transfer to the next garlic hop. Each hop removes one layer of encryption, some even send these messages on SSL connections for further security. The garlic hop will decrypt the last layer of encryption and, in the example, see the four garlic cloves. It has to be able to read their headers and process them accordingly. However, content may be further encrypted when it is content for other nodes.

data to be forwarded along the path. Additionally, there are messages for the rerouter where the initiator (source) signals to cut or extend the current path. The path extention message will then include also a message segment for the new next hop. This will be used to establish the context for the next hop with the source and the path. The telescoping path approach also allows negotiation, although this is, to our knowledge, hardly used. In principle a multi-message protocol could be used between the new node and the initiator. Another option that e.g. Tor uses is to make a path a leaky pipe. At intermediate routers the packet can also signal to exit the path there.

There are also versions of Onion Routing that do not establish a context, see Chapter 8.

7.3.2 Garlic Routing

The I2P website refers to Michael Freedman as the first one who mentioned the term 'Garlic routing'. He mentioned it in his chapter about future work for Freehaven[DiFM00] in the master's thesis of Roger Dingledine[Ding00].

Garlic Routing is an extension of Onion Routing. In the Onion Routing each hop towards a destination peals off a layer of encryption. Finally, at the end of the path, the receiver decrypts the message. In case of Garlic Routing, there is not one message, but multiple message fragments (cloves of a garlic) that may be independent. Fragments from multiple garlic messages that ended at the same hop may also be reassembled to a new garlic message to the next garlic hop.

Garlic Routing may use Onion Routing between the garlic hops. Figure 7.4 illstrates the combination of Onion Routing and Garlic Routing. At each of the garlic hops a set of messages appears and each clove message is processed independently by the hop. The processing is following a description in the header of each of these clove messages. This may, as already mentioned, include to send the message together with other messages as new cloves of a new garlic to their shared next hop. So, the common denominator is that each hop assembles fragments as cloves in a garlic message. The fragments contain a description for the next hop and the garlic is sent via Onion Routing, although the Onion Routing can be considered as optional. The process is repeated for all clove messages that did not yet reach their final destination.

Another use-case for Garlic Routing or, better, garlic encryption is to send multiple messages with different receivers within one message that passes all the receiving nodes. Each node can only read the data that was meant for it. Figure 7.5 shows an example for the cloves and layers of encryption. The advantage is that this sending of an anonymous message to multiple nodes can be done and that confidentiality is still achieved.

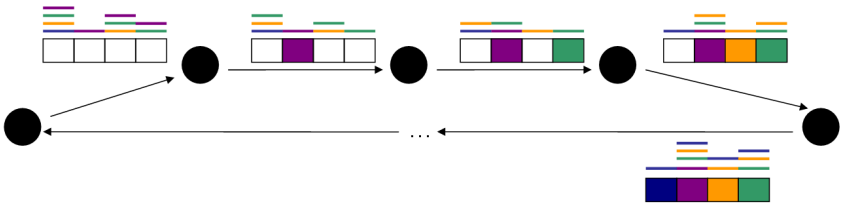


Figure 7.5: If nodes on a path need to be able to read different cloves, encryption needs to be adapted to become cleartext when the node decrypts the arriving message. The node might rewrite and reencrypt its fragment with new information. It also needs to do this in order to hide which of the cloves was its clove. The last node may return the message to the sender is some appropriate way.

Contrary to Onion Routing Garlic Routing is unidirectional. It is a one-sender many-destinations concept. There is no direct way back due to the independence of the cloves. So, it does not operate on connection-oriented paths and there is no need to setup these paths[5]. Of course, return paths are needed to reply and need to be established in some way that also uses the Garlic Routing.

A disadvantage about being unidirectional is that bidirectional connections are usually used to establish a symmetric session key and then use it for better performance and less overhead. So, the expensive asymmetric encryption and decryption operations are only performed when setting up a path. Basic garlic routing uses them per message, although key caching mechnisms can relief the problem. Another issue is that in pure Garlic Routing the reply blocks with information for a return path can directly lead to the sender if their encryption is broken. There are ways to deal with these problems, and I2P has its solutions for them.

As fragments are uncoupled from their overall message, transport protocols may have a problem with Garlic Routing. The order of arrival may be extremely mixed up and protocols either need large buffers or they will lose fragments. As this also limits the amout of data e.g. TCP would dare to send, Garlic Routing may further limit its performance due to interaction with congestion control.

The benefit of Garlic Routing is that it further confuses the traffic. Messages can be split up and sent on different paths. This is annoying for traffic analyis. Furthermore, it allows to add redundancy and path-diversity as means to improve reliability and robustness.

7.4 Case Study: I2P

The Invisible Internet Project (I2P)[anon] is not an academic anonymization system. It was developed for the practical use by volunteers who wanted to be anonymous on the Internet. It is therefore interesting what solutions they chose in their engineering with both anonymity and practical usage in mind. Like most anonymous systems today it is far from being in a final release stage and will need further years of development and redesign. While the developers do not want a lot of publicity due to the incomplete state, there have already been filesharing clients using the system. I2P currently has 700 - 1000 nodes in the network. It is, thus, one of the larger anonymity networks. A difficulty with I2P is that a lot of documentation is out-of-date and that each new version may also include changes in the concepts.

[5]With the exception that the Onion Routing between the garlic hops may be connection-oriented.

In some sense I2P is an anonymous anonymity system as it is developed by an anonymous group of developers. These anonymous persons currently only communicate via the methods I2P implements. So, nobody knows who they are and they themselves do not. While this may fit quite well with the idea of an anonymization project, it also caused a lot of trouble when the head of the project 'jrandom'[6] left in late 2007 with no way to contact him. When the server broke down, there was no way to get it up again. The consequence was that all had to be merged to a new website, now i2p2.de[7]. As the website was also used for bootstrapping, connecting to the network was not possible for new peers until a new release was published on the new website.

Originally, the goal of its developers was not a new system, but simply the proposal by jrandom to use the Java Message Service as transport layer for Freenet[CSWH01]. Things went differently and I2P spawned off as new project from Freenet. jrandom was the main developer of the system, but other contributors joined in. They took over the system early 2008 after jrandom left.

7.4.1 Introducing I2P

In this section we introduce I2P and provide the necessary details for understanding and analyzing its operation.

7.4.2 Basics

I2P is a Peer-to-Peer anonymization system. All peers, thus, contribute to the service. Originally, I2P was intended as pure P2P network. In the meantime, however, the introduction of floodfill peers resulted in a hybrid superpeer architecture.

The basic idea of I2P is to establish multiple inbound and outbound tunnels for a peer to communicate with the rest of the system. A peer builds these tunnels with the currently best performing peers it knows. The performance evaluation is based on the local experience of a peer and not globally shared between the peers. As all peers contribute to the tunneling and operate as rerouters there is a lot of other traffic passing the peer. This makes traffic analysis more difficult.

The software is based on Java with minimum requirement Java version 1.5. As a Peer-to-Peer system it operates primarily decentralized. However, it bootstraps via server and it caches peers it has seen. The basic idea is that peers join the system and register as routers. The routers are stored in the network database of I2P, which is stored in the Peer-to-Peer network and is currently also available via the I2P webserver. Applications also store their keys and inbound path information in the network database.

Contrary to many of the prominent anonymous systems I2P is designed for internal services only and it does not deal on system-level with exiting the system. There is an I2P application that provides the exit to the Internet as service.

7.4.3 Garlic Routing in I2P

I2P does currently not as a default operate the Garlic Routing the way we introduced it. It is not standard for I2P to use the garlic hops as re-routers. Usually, only data fragments for the receiver of the garlic are sent in one garlic message. The main concern is that the delay would be too large if this operational mode was used. Latency is already high without such additional cost. However, this use of Garlic Routing is in the discussion for

[6] Pseudonym of the major I2P developper

[7] For the original domain the whois only refers to a provider in Argentina without a reference to jrandom. I2P today with its website in Germany has a provider in Ismaning near Munich and is registered on a person living near Ulm.

future versions of I2P that implement non-trivial delays to further obscure traffic patterns in this combination. The idea is to use these methods for delay-tolerant communication with stricter anonymity requirements.

I2P uses Garlic Routing for tunnel setup, for publishing data in the network database, and for checking with a message in an extra clove if message delivery succeeded. The latter is usually not done per packet, but for a set of packets. The size of the cloves in the Garlic is flexible and the sender is allowed to add random padding.

Once the garlic is created it is fragmented to smaller segments with fixed size of 1040 Bytes. Each of these segments is sent from A to B via the outbound tunnels of A to inbound tunnels of B.

7.4.4 Tunnelling

A peer in I2P builds a variety of tunnels. The number is not fixed, but three per type of tunnel is a common number. Tunnels in I2P are unidirectional.

There are **outbound tunnels** to hide behind a cascade of nodes when sending a message. There are **inbound tunnels** to hide one's identity when receiving a message. Figure 7.6 shows the communication from A to B, i.e. for one-hop garlic routing which is usually used by I2P. Finally, there are exploratory tunnels to find out about other peers in the network and check their connection. This is important for the peer selection algorithm described in section 7.4.5.

The tunnel length is variable. Length 0 for a tunnel consisting only of the peer. The typical length of a tunnel is 3 nodes. Given 3 nodes per tunnel, the end-to-end communication of two peers in I2P takes 6 intermediate hops. The maximum length for a tunnel is currently 8 as the tunnel creation request is limited to 8 cloves.

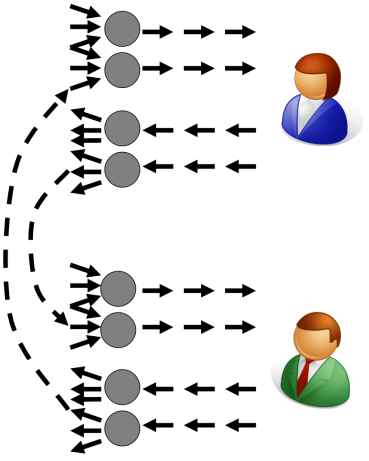


Figure 7.6: Nodes build inbound and outbound tunnels. The outbound tunnels can be addressed and accessed via the IP address of first node and the tunnel ID. If nodes communicate they virtually interconnect their tunnels.

In the tunnels I2P applies a principle called strict ordering. A peer locally orders the peers it knows. This ordering is used for all tunnels. Within a tunnel, peers are not ordered randomly, but according to this local ordering. This means that in tunnels of a peer C A is always in a tunnel before B if A is before B in C's ordering. An effect of this ordering is that it distorts the statistical distribution of predecessors and successors seen by a node.

This is meant to fight attacks like the predecessor attack where an attacker uses such information to determine sender or receiver for long-lived streams or services.

Within a tunnel, I2P basically uses Onion Routing. The encrpytion applied within a tunnel is therefore onion encryption. Additionally, hop-by-hop communication is secured via TLS.

Asymetric cryptography is used to establish a shared key. This is used for encryption with AES256 and for HMAC with MD5[8].

Replay protection is not mentioned for the tunneling and network level. This is a bit problematic as it may be used to reconstruct tunnels. However, tunnels are rather short-lived in I2P (10 minutes), so they seem to consider this as enough protection. There is a replay protection on transport layer of I2P, that works end-to-end.

7.4.5 Peer Selection and Tunnel Selection

Tunnels in I2P are built randomly, but not all peers are used for all kinds of tunnels. Exploratory tunnels are used to explore other peers. Peers are evaluated according to the tunnels they were in and how they reacted to tunnel requests.

I2P calculates four values to represent the quality of a peer. They are speed, capacity, integration, and failing. Speed is a projection of possible roundtrip messages per minute that the client may send through the peer. Capacity is a projection how many tunnel requests of the client the peer may accept within the next hour. The integration is not relevant for the peer selection, and evaluates how much the peer knows about other peers. It is planned to be used for floodfill peer selection in future versions. Failing is an attribute that signals that a peer seems to be overloaded and fails to provide the expected service as relay.

Failing peers are discarded for building tunnels. I2P considers capable nodes as nodes that have a capacity higher than the median of all known peers. Similarly, I2P considers some peers as fast. If they are fast and capable, I2P will use them for building inbound and outbound tunnels.

In our testbed in Tübingen we played with I2P and experienced a strange problem. After setting up a group of peers, they only operated for a short period of time. Then, in most cases they refused to build tunnels for a long time, roughly an hour. After some time they will slowly restart their activity. We assume that this is an artefact of this selection process that becomes a problem when only few peers are in the network. The reason seems to be that peers have to discard tunnels and, thus, all tunnel peers are considered to be failing, all peers in the network end up with knowing only failing peers. Thus, they cannot build tunnels anymore until some timeout makes them explore the other peers again.

So, while this peer selection is meant to improve performance, it is not fully stable. Another issue is that this evaluation of peers may create an asymmetry in peer usage that could be used for traffic analysis or to lead victims to well-performing attacker peers. As most peers are only contributing a fraction of their real bandwidth to I2P, an attacker with a few machines that contribute most of their bandwidth to I2P can become the top candidates for other peers to select them as tunnels. Of course, if all peers use them, they cannot handle the traffic. However, the may only show a good performance against some target peers. The target peer is a selected peer from the overall I2P network. It is not possible

[8]While MD5 is considered to be broken, this primarily affects signatures and is not yet completely relevant for short-term message integrity. However, algorithms could be changed in the future and before the first real official release of I2P. For router identities SHA256 is used, so the signatures within I2P do not seem to be affected.

to select a hidden service or communication partner directly. There is also the limitation that tunnel end-points are not obvious as the next hop may also be a relay, unless the length exceeds the current limit. To conclude, while it may not possible to detect the peer directly with this attack, the attacker can use it for a simplified confirmation attack once the correct target peer is found.

7.4.6 Identities in I2P

As I2P is a P2P approach the peers participate as routers. As router they take part in tunnels and forward messages to provide the onion routing functionality needed to hide senders and receivers. The identity of a router is known. Identity in that case means key, IP address and port and this is necessary as peers need to be able to contact the router. A router publishes its information in the network database. The information consists of its router identity (ElGamal encryption key, DSA signature key, certificate), the IP address, time of publication, its signature of the router information. Currently, there is also a text field with statistics for debugging that needs to be removed for version 1.0.

If a peer would use its router identity for anonymous communication, it would not be anonymous. It therefore has another identity, with a public/private key pair. This is used when the peer uses I2P, i.e. builds tunnels and sends messages. If a peer wants to provide a service, this services needs yet another identity that is different from the peer identity and used for all communication of the service. The service builds its outbound and inbound tunnels and registers itself at the network database.

This registration is called LeaseSet and a lease is valid for 10 minutes. A LeaseSet includes the identity of the service, expiration date, and a couple of leases, i.e. inbound tunnels to the service given by the gateway router and tunnel ID.

7.4.7 Network Database and Structure

I2P stores the public data for the network within a network database. It contains the information about the current routers and the current lease sets. Originally, I2P tried to use a DHT for the database, but they changed it to the hybrid unstructured Peer-to-Peer network on the basis of their Floodfill Algorithm. We first introduce the Floodfill Algorithm and then the reasons for the abolishment of the DHT.

The Floodfill Algorithm

I2P is currently using a hybrid P2P approach to implement the network database. In I2P a small subset of the peers are so-called **Floodfill Peers**. Each floodfill peer maintains a full network database. Changes are flooded among the floodfill peers.

Other peers update their information by asking a floodfill peer. When a peer inserts or updates information in the network database, it simply contacts a floodfill peer. The floodfill peer will acknowledge this information immediately and flood it to its other floodfill peers. As a consequence, a peer has an immediate response and does not have to ask many peers. It will, however, perform a test request in order to check if the answer has been spread to other floodfill peers.

The floodfill peers are not assumed to be trusted though. The floodfill peers are selected per configuration. In the future they will be selected automatically as well. Once the number of floodfill peers drops below a threshold, high-capacity peers will become floodfill peers. There is also the discussion to use the integration value for the floodfill peer selection. The integration value is determined in the peer selection as a measure of what fraction of the network the peer knows. There is currently no use for it, but this will change then.

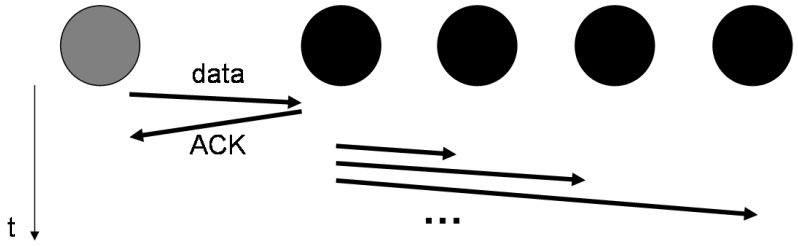


Figure 7.7: With the floodfill algorithm, the storage task is transfered from the client to the floodfill peer. The client gets an immediate response. The floodfill peer will then publish the data to the other floodfill peers.

Nonetheless, the Floodfill Algorithm seems to be an intermediate solution. It has its advantages like being faster than the old KAD algorithm, avoiding that information is unevenly distributed which could be used to break anonymity. On the negative side, there is the power a floodfill peer has, in particular during the bootstrap phase. Thus, there are also attack vectors on the basis of the Floodfill Algorithm. Another issue is the question of scalability. In particular as all the peers need to know everything and everything is flooded to all floodfill peers. The I2P developers currently have no plan to change it, although it is an ad-hoc solution. Considering advantages and disadvantages, one may think of a hybrid solution with the floodfill peers organizing themselves in a DHT.

I2P's Abolishment of their DHT-based Network Database

Originally, I2P used a Distributed Hash Table (DHT) as network database. The particular DHT was a Kademlia derivative, similar to the KAD in filesharing networks. The authors of I2P decided to eliminate the DHT as it was one of the performance bottlenecks. There is also discussion if the limited view of a DHT peer and the distributed knowledge is a possible threat to anonymity systems.

Within the DHT approach, peers all have their IDs in the Peer-to-Peer network. Data is stored at a place in ID space that is usually derived with a hash function from its name (and in I2P also the current date, hash(name,date)). For the storage of the data, a group of peers within a subsection of the ID space that includes the corresponding ID is used. The peers are found via lookup and data is stored on the peers. For the request it is necessary to get into this subsection (given by the first bits) and then look there for peers with the requested knowledge.

The main drawback of the KAD lookup is the large number of requests for the lookup and the storage. Figure 7.8 illustrates the operation. Comparing this figure with the figure for the floodfill algorithm, it is obvious that many more messages are required on client side. A lot of messages have to be sent and this takes some time to finally find the appropriate data and to get the confirmation. The I2P authors report that a node needed to contact 40-60 nodes to write data to the network database and to ensure enough replication of the data. In comparison to the floodfill approach a peer only contacts one floodfill peer that immediately answers or acknowledges the new data. The rest of the storage operation is then executed by the floodfill peers.

It is clear that a distributed DHT-alike approach may scale better in future versions. Currently, I2P has no good answer for the future of its network database. Maybe a hybrid solution with KAD among the floodfill peers may scale.

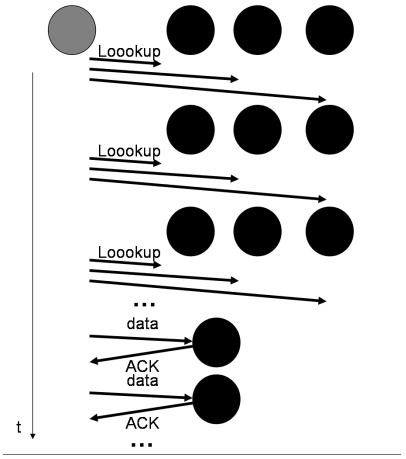


Figure 7.8: The DHT variant involves lookup and storage. First, many nodes have to be contacted for the routing towards the target ID. Second, nodes close to the ID need to be contacted to store the data. The node itself is responsible for replication.

7.4.8 Naming and Name Resolution in I2P

I2P uses a naming compared to the Internet. URIs like (www.example.i2p) stand for I2P sites. I2P calls eepsites. The name resolution has to translate the name to the hash values in the network database.

A big problem is that a system like I2P cannot use DNS as it avoids to implement the necessary control structures and also single points of failure if it is centralized on a server.

Currently, I2P uses a simple approach that is, however, far from being satisfying. It is based on host.txt files that provide name hash combinations. Each eepsite can provide its own hosts.txt file and each user has her own hosts.txt file for her own sites and for her own names for other sites. All known host files are merged to an address book. Addresses cannot be reserved globally, and duplicates may exist. It, thus, depends on the situated view of a user which one is known and preferred.

7.4.9 Experiments with I2P

In this section, we briefly discuss our experiments with I2P, including potential attacks. The experiments were with I2P version 0.6.30 and 0.6.31 (first version without jrandom).

Peer Selection Issues

As we already discussed in Section 7.4.5, we experienced stability problems when the network consisted of only few peers. We assume that I2P rates all peers as failing and refuses to build tunnels for some time. The peer selection algorithm is, thus, instable and the network only recovers as peers try again after timeout.

Another issue is that an attacker might be able to use the strategies in her favor. The attacker can do this by providing a small set of highly-capable peers. As the peer selection algorithm is highly selective, the attacker can highly profit from the approach. Attacker nodes can be preferred way more than in cases when the peer selection is purely random or simply relative to available bandwidth. This is, to some degree, countered by I2P by only using routers from different IP ranges within a tunnel. The primary purpose of the peer selection is performance. However, the basic defense idea of the peer selection is that in a large network, the attacker cannot compete with the amount of resources provided by other nodes. This only partially holds as the attacker may focus on a few nodes and use her resources to attack them.

Attacking I2P

Here, we describe some general attacks against anonymity systems and how they apply to I2P. When applicable, we also described their possible mitigation.

It is clear that I2P does not defend against a global observer. I2P may mitigate some of the attacks by a global observer due to the rerouting of messages at each peer. Thus, each peer hides its traffic within the rerouted traffic. This is one argument of I2P for the use of the Peer-to-Peer approach. However, the peer selection algorithm will not use slow peers as rerouters. Thus, many peers will not profit from this argument and not hide their traffic in this way.

Attacks against Bootstrapping

Bootstrapping in I2P means to contact a floodfill router that is fixed in the I2P code. Before the change the router was dev.i2p.net, now it is netdb.i2p2.de. This node is considered to be trusted as it is under the control of the I2P developers, just like their webserver where one downloads the program. The authors also publish a SHA-1 hash of the program to verify it. As the verification has to be done manually, it is hardly practiced in reality.

An attacker could either try to change this code and introduce her own floodfill peer. In that case the user has to download the program from the wrong source, the attacker. The other option is to take over webserver or floodfill peer. However, this attack would probably be recognized and only work temporarily. A local attacker could interfere with the communication and answer instead of the correct floodfill peer. In any case, if the attacker succeeds and controls the floodfill peer used for the bootstrapping of a user, the attacker can mislead the user and fully control its communication.

Attacks against the Network Database

The floodfill algorithm moves load from the peers to the floodfill peers. Any change is only a simple message for a peer, but it amplifies to multiple messages among the floodfill peers. Denial-of-Service attacks against the floodfill peers are, thus, easy. Peers can also become floodfill peers themselves. This has the effect that the number of messages between the floodfill peers increases linearly with the number of floodfill peers.

Floodfill peers may also alter the network database. This could include to remove the RouterInfo of a peer. As the peer checks its RouterInfo after storage, the attacker peers need to store the data until this request and they need to hope that the peer will use them. The Sybil attack may increase the likelihood of using attacker floodfill peers.

Floodfill peers may also answer requests in a wrong way, in particular denying knowledge. They cannot alter the entries themselves as all entries are signed and cryptographic identifiers are used.

Peer-to-Peer related Attacks

As a Peer-to-Peer system I2P is prone to Peer-to-Peer-related attacks, in particular to the Sybil attack. The Sybil attack means that an attacker adds multiple nodes into the network. In I2P an attacker may use the Sybil attack to add multiple relays, in order to be able to perform attacks. Given multiple nodes, a victim may select some of them within her paths. As a consequence, the attacker may use this to determine source or receiver. Here, the one node per IP range limit for tunnels may help to reduce the attacker nodes in a path. Another argument by I2P might be that the attacker needs to commit the resources to the network and, thus, has to provide per Sybil node the ressources of a good high-performing node to be successful in executing real attacks.

Another common Peer-to-Peer attack is the Eclipse attack. The attacker may trap a peer or group of peers in a subnetwork connected only or primarily with attacker nodes. As a floodfill peer, an attacker mislead new peers. Once a node is in the network, it has many connections (the number of known peers for a non-floodfill peer is above 200), so that the attack becomes hard.

Tagging Attacks

I2P is vulnerable to tagging attacks. While there is reply protection, the packets are not protected against changes. The message authentication codes only operate end-to-end on the higher layer transport protocols. Malicious peers may, thus, tag messages in order to check if the appear at a colluding malicious peer. This is one way to determine if malicious peers share a tunnel and to determine if they both see one and the same packet. Tagging itself is not a primary problem itself, but it is useful to setup other attacks, i.e. for an attacker to be highly certain to be the last node and, thus, see the receiver. I2P mitigates this attack by only using IP-diverse paths, i.e. not allowing nodes from the same IP range. This assumes that the attacker is limited to local IP addresses.

Predecessor Attack

In I2P peers know which tunnel they are in. It is quite common for anonymity systems that there is some piece of information that can be used to correlate different tunnels or messages from one user. In I2P peers know the tunnel ID (plus tunnel verification key, change key, etc.) and can at least scan the network database for the lease with the tunnel ID. Thus, being on the tunnel to a particular hidden service like an eepsite can be identified. In I2P tunnels are limited to 10 minutes and newest versions allow to further limit their lifetime also to a number of messages or maximum data volume. Thus, tunnels change often and there is a chance to get enough data for the statistics. Given a peer knows various paths it was part of that correspond to paths of an anonymous identity. Then it can make statistics about predecessor and successor it has seen in the tunnels. If all is random, the victim is more likely to be seen than any other peer. In the example of I2P, the attack against an eepsite is a successor attack.

I2P fights the predecessor and successor attack in various ways. The first way is that paths are of random length and that even in 1-hop or even 0-hop tunnels information about sender or receiver cannot be easily inferred. One needs to use statistics for predecessor or successor attacks. The second way is the strict ordering. A peer orders the nodes within a tunnel in a peer-specific way. The peer creates a random order among all nodes it uses and this order is preserved within tunnels, say C is always before A when the peer's order is 'CEBDAF'. As a consequence, the statistics are distorted by this ordering. As the order is unknown, peers cannot use their positioning in the order to adapt their statistics, e.g. peers in the first positions are highly likely to see the corresponding peer quite often (predecessor attack) while peers in later positions are good for successor attacks. The ordering can even be counterproductive. The first node in the order (C in the order example above) sees the sender 100 % of the time and the last node in the order (F in the order example above) always sees the receiver. If it can observe many paths, they can identify receiver or sender. Similarly, the second node will only see two different predecessors with the sender being more likely if many nodes are used. As I2P is rather selective and only uses few nodes according to performance and rating, it is not unlikely that peers later in the order only see a predecessor and that they cannot be sure that 100 % means sender or receiver. In the next chapter for MORE the analysis of using ordering against the predecessor attack is clearer. Ordering in case of MORE increases the efficiency of the predecessor attack.

The peer selection is also a measure to counter this attack. It reduces the number of peers that are used for the tunnels as only the best[9] fast and high-capacity peers are used. This makes it harder to bypass the distortion of the statistics imposed by the strict ordering. Fast and highly capable peers are seen more often and peers are less likely to take a position in a tunnel that is far from their position in the ordering. However, the peer selection also introduces the problem that the few peers may be more likely attacker peers and, thus, increases the problem of these statistical attacks.

Timing Attacks

Timing attacks use the message flow within a connection to determine whether one is close to the sender or receiver. Furthermore, timing attacks may be used to determine protocols as higher layer protocols may use typical timing patterns. I2P's major defense are unidirectional paths. As messages travel along different paths, replies cannot easily be seen by a limited attacker. This attack is more important for strong attackers, against which I2P does not primarily aim to defend itself. I2P plans to implement non-trivial delays in the future for critical delay-tolerant tasks. This may also help if the attacker is in the position to see both message and reply.

Attacks against the Naming

Naming is always crucial and yet always difficult. I2P uses names like in DNS. This is not the problematic point. The problem is how to map them securely to I2P identifiers. As I2P is decentralized it cannot provide a centralized solution like DNS. I2P only protects the inverse relation that is not as relevant. An eepsite may sign its name and this may be published in host.txt files. However, other sites may also use the name and I2P does not provide a solution to deal with multiple sites with similar names. It resorts to a local view so that local information and information from frequently visited sites is more relevant. As a consequence, there is no real protection, atleast for initial contact.

I2P tries to help with higher layer solutions. The I2P forum is one way to publish eepsites. Only a few trusted members are allowed to post and, thus, serve as anchor of trust into the name and ID relations posted in the forum. I2P mail can be used to post eepsites to anonymous friends in the I2P network. One can also use the addressbooks (hosts.txt) of friendly eepsites. While these methods work in some way, they do not finally solve the problem. I2P services are meant to be untrusted and one should always distrust other eepsites as a consequence of this naming problem.

We want to note that this is a fundamental problem and that a good solution for a decentralized system like I2P is not yet foreseeable, in particular as anonymity and robustness also need to be taken into consideration.

Harvesting Attacks

Harvesting attacks simply try to collect as much information as possible. At the moment, I2P is providing a lot of information through the network database (also more than necessary due to debugging), which is known by all floodfill peers completely and by all other peers partially. Before the crash of the i2p.net website, the network database could be downloaded by anyone on the web. The database includes e.g. the IP addresses of all peers.

However, there are also limits. Due to NAT and firewall problems, I2P is about to introduce the concept of restricted routes. Peers behind NAT or firewall may use a tunnel as entry-point and only publish the tunnel as entry-point in their RouterInfo. Afterwards, they may communicate directly, but their IP address is not stored in the network database.

[9] best according to a local view

There are many ways how harvesting can be used to stage further attacks. We practically also used harvesting for our intersection attack experiment described in the next section.

Intersection Attack

The intersection attack is a common attack against anonymizers. The basic idea is to compare the set of currently existing nodes in a network with the victim, i.e. a hidden service or a user. If only one node exists, the attack already succeeds. If there are more nodes, we gain the information, which nodes exist and one of them has to be the victim. The process is repeated over time. If the victim is online, all offline nodes are excluded from the set. If the victim is offline, one may either also remove the nodes that are up or leave this. One argument for leaving this is that in particular a service may not always be running despite of the node being up.

Intersection attacks are very powerful and operate on long-term basis. Usually, enough tests have to be performed in order to identify the victim. There are also problems. Victims may change their identifier over time. In case of I2P we may consider the router identity and the IP address the relevant identifier. The IP address may change when the user reconnects to the Internet after being offline for some time. The router identity seems to be long-lived.

In our intersection attack experiment, we tested the convergence of the intersection attack for different sites. This is primarily a test how fast the sets can be reduced from a large to a small number. To perform the intersection attack, we checked the eepsites and all known peers twice a day if they are online or not. At certain points in time, we thus connected to all known peers and to the eepsites. We did an active checking as database information may not be accurate. The positive test determines the set of peers that are online when an eepsite is online. The negative test determines the set of peers that are online when an eepsite is offline. We used three eepsites for the experiment: A is mostly online, B is often online, and eepsite C is mostly offline. Overall we found roughly 700 peers via the database of a floodfill router. The experiment ran for half a month, which means 30 intersections. On the basis of the positive test, we could reduce the set of site A to 58, for site B to 280 and for site C to 379. If we combine positive and negative set, this can be reduced to 38 for eepsite A, 142 for B, and 229 for C.

The experiment cannot be generalized. One can, however, draw first conclusions. The overall churn allows eepsites that are offline most of the time to hide themselves better than expected, in particular as hardly positive tests with them online can be produced. Outages for eepsites that are online most of the time seem to be more severe as they only hide in the set of highly available rerouters. In the long run or with more tests per day, the intersection attack would have reduced the set further and succeeded.

This method with positive and negative tests can be defeated when a service is run on multiple hosts or if the hosts frequently change their identifier. The little uptime of eepsite C could be a hint that it could be run by a dial-in user. However, router identities seem to be long-term in I2P, so that this problem can be circumvented.

Performance

For a performance evaluation, we installed I2P on computers in a testbed that was separated from the Internet. It contained 8 PCs that were interconnected via a manageable switch and all of them were connected to a management system for the organization of the measurements. The number of nodes could be increased by using I2P on different ports within one machine.

We already discussed issues of the tunnel create process in I2P. Within one-pass a tunnel is established. All intermediate nodes have to agree, otherwise the tunnel is discarded. Even if the first node does not accept the tunnel all nodes will be asked due to the garlic encryption and keeping the anonymity. If tunnels are long, the process to finally create a tunnel can require some tries and, thus, require some time, in particular if timeouts have to detect the failure. Tunnels are an important concept, and we observed that each peer participated in about 100 tunnels at a time. The bandwidth consumption without any traffic is up to 4 kB/s.

The datarate that I2P uses can be configured. We installed an eepsite with a file with 47 MB in size. Given a limit of 64kB/s (512 kbps) we observed 10 kB/s (80 kbps) as traffic. A larger limit of 1,024 kB/s (8,192 kbps) results in a real rate of 30 to 35 kB/s (240-280 kbps). Table 7.1 shows the data rate. Traffic control puts a limit onto individual data rates and higher rates are not fully utilized. We observed that the data rate is extremely variable over time. It is reasonable to assume that congestion control and frequent restarts of the congestion window may cause these rates.

Parallel requests	Rate per Request in kbps	Total Rate in kbps
1	240-280	240-280
2	224-240	448-480
4	184-208	736-832

Table 7.1: Data rate for the delivery of a 47 MB file on an eepsite (limit set to 8,192 kbps).

The data rate the system produces is one important performance factor. Another issue is the overhead of the system, i.e. the amount of bytes transferred in comparison to the real data. Table 7.2 shows the overhead for the transfer in our testbed. Roughly 2.3 times more data than the actual file size is sent. For the Internet, this is slightly higher. We experienced up to 3 times more data at sender and receiver.

Peer	Bytes received in MB	Bytes sent in MB
1	78	79
2	65	66
3	46	47
4	146	141
Receiver	116	8
Sender	64	176

Table 7.2: Amount of data sent over the network for a 47 MB file. The data is from an examplary test run. The sender was also used in a tunnel, thus the larger amount received and sent bytes.

7.5 Relation to Architecture and Design

Anonymous systems need the participation and cooperation of users and no single entity can achieve anonymity on its own. It is therefore necessary that a system interconnects multiple users and other contributers. A Peer-to-Peer solution seems to be a quite natural choice for such a setting. However, this partially conflicts with the security also needed within anonymous systems. Thus, to achieve anonymity in a system, the designers need to have a solution for basic security mechanisms, and in addition need to find ways how nodes in the system can interact for the anonymity goal. Similar to security, anonymity is not defined on the basis of a single property. The goal anonymity contains a variety of options of who should be anonymous where and in what way.

What is also complicated about anonymous system is that the attacks against such systems are not only internally. Many attacks are like side-channel attacks, the timing attacks for instance. This means that even a theoretically secure system may leak information in practise. These effects and other side-channels are hard to foresee and handle. It is, thus, relevant to be realistic and decent within the goals when adding anonymity or pseudonymity to the architecture of a system. This should be kept in mind when working through the steps for adding anonymity:

1. state the anonymity goals
2. state trust and distrust in entities of the system
3. state possible attack vectors for the system against these goals
4. integrate anonymity methods into the system according the goals

Existing systems provide insights and ideas for solving the arising issues of how to achieve the goals. The I2P system implements some unique methods like the Garlic encryption. It is also a Peer-to-Peer system, although hybrid as it relies on its webserver for tasks like bootstrapping and on superpeer-like floodfill peers for the network database. I2P also showed that DHTs are not necessarily the Peer-to-Peer approach of choice, in particular for an anonymity system. Their choice to remove KAD and use the floodfill algorithm is nonetheless no final answer. I2P is not a system that primarily wants to operate as anonymous frontend for the web, unlike e.g. Tor that at least is primarily used in that way. Therefore it focuses on internal anonymous services. However, anonymous services are harder to secure and we showed that there are ways to trace them and for the example case ways to mitigate the attack.

8. Dynamic Multipath Onion Routing

8.1 Introduction

Dynamic Multipath Onion Routing [LPWN+07] is a proposal for Onion Routing with hidden services that was created when most of its contributors were at Tübingen University. The inventor and main contributor is Olaf Landsiedel. Nonetheless, other contribtutors including the author provided valuable input. This chapter provides system description as much as necessary and apart from that focuses on our contributions. Previous versions were called SARA and Core[LaNW05, LRNW+05].

It all started when Olaf Landsiedel asked the security experts in our group a strange cryptographic question about RSA. However, the desired operation was not possible as it actually needed commutativity for encryption algorithms, which is similar to blind signatures, yet not completely. After we found out that this way will not work, Olaf Landsiedel told us about his idea that he wanted an anonymous system where sender and receiver hide behind paths. The receiver needs to send the encrypted receiver path to the sender and the sender needs to couple the two. Here, this operation would have been necessary. The problem was how the receiver could already encrypt the data the sender would send beforehand, so that the intermediate hops could read their part of the header and the Onion Encryption gets removed correctly hop-by-hop. After some thought we found a different way to achieve the goal. This is part of our contributions to the system.

The most exciting aspect of MORE in comparison to the early SARA concept and other current anonymizers is the idea to send each packet on a different path, which is meant to fight dangerous pattern attacks. In the next section, we will introduce MORE and afterwards we will analyze the proposal.

As there is no real MORE system, except for a first demo implementation that only supported a subset of methods and features, one cannot decide which design decisions were taken when they are not clear from the articles. In many of these cases there was yet no architectural decision as MORE was not developed to a level where all of them would have to be taken into account and been realized. If necessary, we try to fill the gap by taking the most reasonable ones and sketching problems if less reasonable ones were taken[1].

[1] For many fundamental issues like bootstrapping with the sensitive communication to the service directory, integrity protection, ... there is no specification.

8.2 MORE - Dynamic Multipath Onion Routing

Onion routing is the basic paradigm found in almost all anonymous systems. Usually, a path from sender to receiver is established and the intermediate nodes reencrypt and resend the message to the next hop. Pattern attacks were well-known before MORE was developed, but shortly after the first publication of the idea, practical pattern-attacks [MuDa05] against Tor[DiMS04, (EFF09] have been published. Pattern attacks usually operate along a path. A message flow tends to have a specific pattern (variation of datarate over time). Additionally, attackers may eventually add a pattern themselves, e.g. by delaying and batching messages. If similar patterns are seen on different routers the attacker may gain information about the path from sender to receiver. Other similar attacks target hidden services directly [OvSy06, Murd06].

Thus, MORE combines two wishes. First, the wish to have sender and receiver anonymity. For this goal the receiver also needs to hide behind a path. Second, it aims to defeat dangerous pattern attacks. MORE operates on the assumption that fixed paths cause patterns and are therefore problematic. The consequence MORE takes is to avoid fixed paths completely.

As being a Peer-to-Peer system MORE also follows the Peer-to-Peer paradigm for the rerouting. Each peer is not only using the system, but also operating as rerouter. Thus, the traffic at each peer includes its own traffic together with traffic for it as rerouter. This hides patterns of sender and receiver to some degree, in particular if their rate is small. Due to the multihop rerouting, the rerouted traffic is larger on average over all nodes. For active nodes, however, their own traffic may exceed the rerouted traffic.

8.2.1 One-Time Paths and Path Concatenation

The core concept of MORE is the concatenation of one-time paths. All nodes need to be able to create such paths with good enough randomness and there needs to be a way to combine and use them.

To achieve this goal nodes need to have knowledge about other existing nodes that they can use as relays for the rerouting. Ideally, a node knows all existing relay nodes. In theory, MORE assumes that this is the case. In reality, a node will only approximate this situation and might also operate on old knowledge. The necessary information for each relay is its locator in the network and its public key. In the current proposal of MORE, the public key is a public Diffie-Hellman value that is used as public key[2]. The only difference is that Diffie-Hellman is used to compute the shared key and that it is not chosen by the sender.

Given this knowledge, sender and receiver can each select a random path. It is assumed to be of random length. Potential receivers publish their path in some way under a pseudonym, which is also related with a different public/private key pair. A sender selects one of those paths for initial contact and receiver paths are included in all the subsequent messages. Path selection strategies are discussed in Section 8.2.3.

The basic idea for the concatenation is that header and body are separated. The receiver path that is published contains the headers for the receiver path hops and some padding to hide the exact length of the path. The first header is encrypted with the key that the receiver shares with the corresponding first node in the receiver path. Then follows the header for the next hop and this continues until the header for the last node in the path

[2]There is no deeper meaning to MORE's using of Diffie-Hellman instead of standard public key cryptography. It was used as the cryptographic library in the test implementation only supported elliptic curve cryptography (ECC) for Diffie-Hellman. Due to the short key length (200-300 bits instead of 2000-4000 for RSA with similar strength), ECC is more suitable as it avoids a lot of overhead.

is reached. The header includes the IP address of the next hop and other information for the packet processing, e.g. the receiver Diffie-Hellman value to identify and determine the symmetric key for the layer.

The headers along the sender path contain the same information, except for sender Diffie-Hellman value instead of receiver Diffie-Hellman value. Figure 8.1 describes the idea of path concatenation.

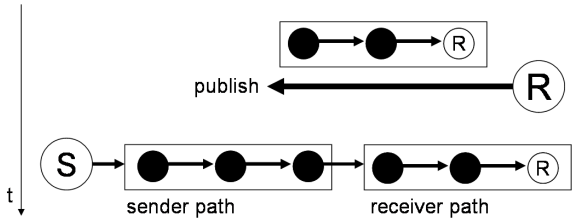


Figure 8.1: Path Concatenation – initially, the receiver published a path to herself. The sender uses this path and concatenates it with her own sender path.

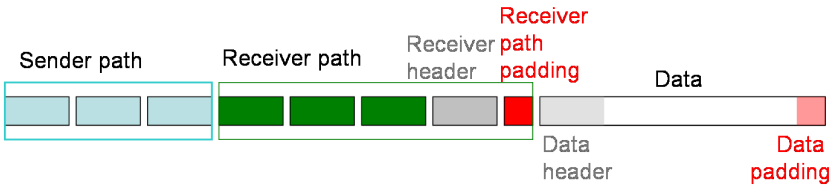


Figure 8.2: The basic structure of a MORE packet. There is a sender path header, a receiver path header, and data with header and padding.

A packet basically consists of headers for sender and receiver path and the data segment. The packet may not necessarily correspond to one IP datagram on lower layers, although according to our information the demo implementation tried to stay with the MTU. A fixed packet size is, however, recommended. Here, we may also deviate from the demo that did not specify an encrypted part within a relay's own header. Figure 8.2 visualizes the structure. For the sender path, there is a header for each hop. The receiver path consists of the same headers for different hops and padding. Unlike in the figure, the padding should be larger than a header in order to be able to hide the length of the receiver path. The header basically consists of a public part and an encrypted part.

Onion Header:

- Unencrypted
  - this-IP:Port (needed by predecessor on path to send packet)
  - this-DH (Public key of sender or receiver)
- Encrypted (This part is decrypted with the Diffie-Hellman (DH) result.)
  - Order Receiver / Relay
  - Path-ID (The receiver needs to know the ID of the receiver path to decrypt the encryption from the receiver path hops.)

  - Padding

Within the header we only mentioned encryption. Message integrity cannot be protected hop-by-hop as MORE cannot code integrity values into the header of the receiver hops. MORE could, however, protect the individual header of a hop. However, this cannot stop tagging attacks as bits of subsequent headers cannot be checked. Yet, this is the major reason for hop-by-hop integrity checks. As a consequence, we consider integrity checks in the rerouter header as optional. The data itself will be protected. MORE can provide end-to-end integrity for data.

In addition to the headers for the onion encryption, there is also a header for the data segment. It contains an unencrypted part to determine the cryptographic keys. Again, the Diffie-Hellman result is calculated and used to decrypt the rest of the data segment. This is the end-to-end encryption from sender to receiver.

Data Header:

- Unencrypted
  - sender-DH
- Encrypted (decrypt with DH-result)
  - Payload index (to order the messages and their content)
  - Length of Payload
  - Message Digest Code (end-to-end integrity check)
  - Payload
  - Padding

Now, we can look at the processing of MORE messages. First, the sender needs to create the initial message. This message is shown in Figure 8.3. The sender selects the nodes for the sender path and selects one of the yet unused receiver paths it possesses. For the message, the sender creates the data segment with data, data header, and data padding. The length of the padding is determined by the size of data plus receiver path, and sender path headers. The overall packet length should be fixed for all messages. The data segment is encrypted with the shared key of the sender with the receiver. The sender appends the data segment to the selected receiver path. Then it adds the headers for the sender hops in reverse order, and onion encrypts the packet accordingly.

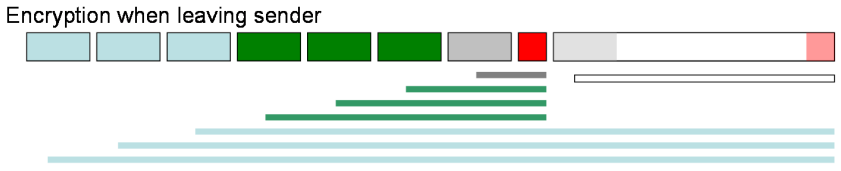


Figure 8.3: Packet structure after leaving the sender. The sender added the sender path in the header and onion encrypted the header with the key shared with the intermediate nodes (green lines). The white box shows the end-to-end encryption of the data segment.

The sender sends the message to the first hop on the sender path. This first hop processes the message. It will read the header and remove the first onion layer. It also removes its part of the header. To keep the size of the message constant, it needs to append data of

the same length. This could be random data or a reencrypted version of its header. Figure 8.4 shows the result. To fight replay attacks each hop calculated a hash value of a message and temporarily stores recently seen hash values. If the same hash value has been seen recently (few seconds), it will discard the message and not send it out.

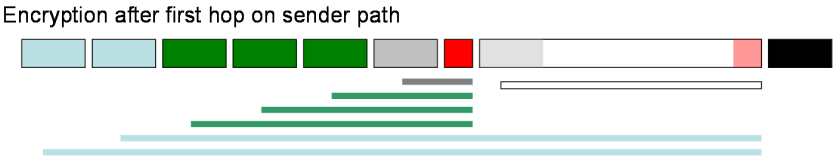


Figure 8.4: The first hop on the sender path decrypts the packet and removes its header. To keep the packet size constant it appends a noise segment (or reencrypted header segment) with the same size.

The last sender hop is the first hop (except for the sender) to see the first hop of the receiver path. This is the first header where the public key in the packet changes from sender to receiver public key. Since the headers should be identical for all hops, this is the only difference there should be. As the sender might use different public keys per hop, this can also be mitigated by the sender. Also, all of the sender's onion encryption has been removed from the packet. Figure 8.5 shows the structure of the message the last sender hop sends to the first receiver hop.

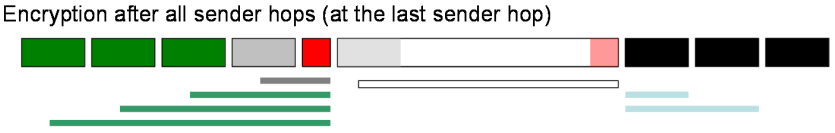


Figure 8.5: When the packet leaves the last sender hop, the first receiver hop is the next hop. All sender headers were replaced with noise at the end of the packet, each of the previous noise segment encrypted by the subsequent sender hops.

The receiver hops remove the onion encryption from the receiver path headers and add onion encryption to the data segment and noise padding at the end. Figure 8.6 shows the message that leaves the first receiver hop.

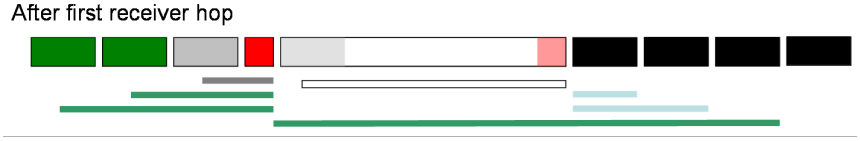


Figure 8.6: The receiver hop processing is similar to the sender hop processing. The public key in the packet changes from sender key to receiver key. As data was not encrypted with the shared key of receiver and this hop, each hop adds a layer of encryption onto the data segment.

Figure 8.7 shows the message that the last receiver hop sends to the receiver. For the last receiver hop, the receiver looks like a normal relay. The receiver, however, can decrypt its header and it will see that it is the receiver and that this is a particular receiver path it has created. With this information, it can remove the onion encryption of the receiver hops. If can also decrypt the data segment and process its content.

Figure 8.7: After passing all receiver hops, the next header is the receiver header. It looks identical to a relay. The receiver removes the header and the padding from the receiver path. Within its header it also needs to learn the receiver path ID in order to correctly decrypt the layers around the data segment. The data header includes information for removing end-to-end encryption and the noise segments.

8.2.2 System Architecture

The MORE architecture basically consists of a service directory and the peers of the MORE network. The peers register at the service directory as relays. The service directory also stores receiver paths to identities of anonymous services or individuals. The service directory additionally provides virtual IP addresses (the non-routeable 10/24 addresses of IPv4, e.g. 10.3.2.4) that are unique within the MORE network. Moreover, it provides a naming service to connect these IP addresses with names (domain names) for an easier human-friendly access to services.

Identities in MORE:

- As relay the node identity is its public IP and port as well as its relay public key (DH-value).
- A node is addressed by its virtual MORE IP address.
- A service is represented by its name (domain name).

It is important to different between node and relay. Relays need to provide their real identities in terms of underlay locators. The operation of the MORE network has to avoid any connection between the two. Otherwhise, anonymity would be broken.

MORE creates virtual anonymous networks. In contrast to most other proposals IP packets can be sent over the MORE network. To achieve this, MORE is implemented as TUN/TAP network device, see Figure 8.8. The IP configuration is DHCP-alike. The naming service is provided via DNS lookup that catches and processes all domains with '.anon' as top-level domain. MORE does not clean higher-layer protocols, e.g. like Privoxy does that is often used in combination with Tor.

For the bootstrapping to the network, a new MORE node needs to contact the service directory in order to get to know a number of other peers that it can use for creating anonymous paths and sending anonymous messages. It is also registered as relay, so that other nodes can use the node as relay for their paths. Figure 8.9 shows the basic steps of the registration process.

Once the node has a MORE IP address, other software can use MORE to communicate anonymously. The application simply uses DNS and IP. Figure 8.10 shows how application and MORE will interact. The service directory operates as global DNS server for the MORE clients and the 'MORE domains'. Identities in the MORE system relate to virtual IP addresses and the DNS lookup process involves this resolution. Applications are separated via ports above the IP layer.

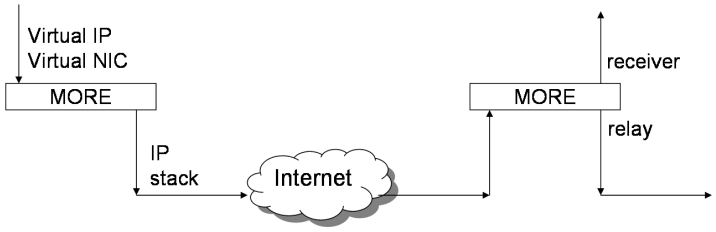


Figure 8.8: MORE provides a virtual network interface that can be used by any application. Messages that are sent via this interface are reencoded and sent as MORE packets to the Internet. Incoming messages for the node will arrive on this interface, messages to relay will be sent back to the Internet by the MORE software.

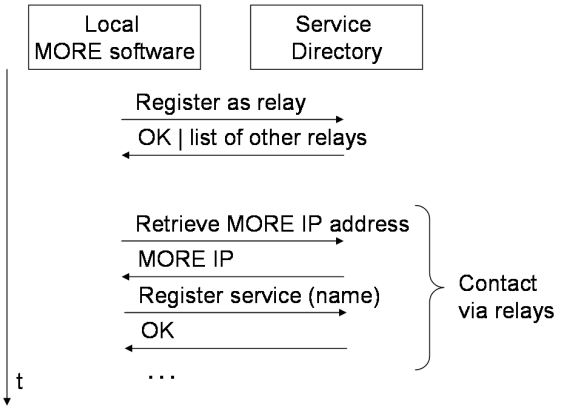


Figure 8.9: The local MORE software creates a MORE node and interacts with the service directory to register as relay and to receive a list of peers. Subsequently, it can retrieve an IP address and register its services or identites.

Currently, MORE is a hybrid design with peers as relays and a server for directory and bootstrapping services. There has also been the idea to use a DHT as service directory. From the basic primitives used in the MORE design, this may be possible. Security and appropriate protocols for DHT-based MORE are open issues as it is easier to model the trust for a server than for even less trustworthy arbitrary peers.

8.2.3 Path Selection

The basic idea of path selection is that each node knows all relays and then selects them randomly to create a path. In reality each node will only know a subset of relays. Although not yet incorporated, there are also proposals to optimize first and last hop selection. The problem with equal sharing of bandwidth among multiple links is that the bandwidth patterns are sent only at a smaller rate. Yet, each node can see and estimate sender bandwith. [SeMu05] analyses this problem and suggest to use an exponential distribution to share bandwidth over parallel links.

For subsequent analysis in Section 8.3 we usually consider uniform random selection. This is the only selection in the prototype and most likely the only one implemented at first when the MORE concept is developed for real use.

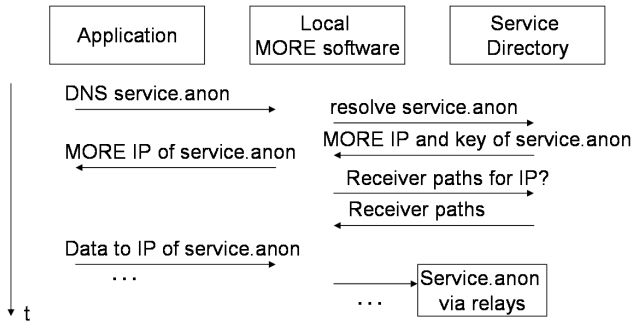


Figure 8.10: When the application wants to access service.anon, it first sends a DNS query. MORE translates this to an anonymous request to the service directory. MORE will then receive the MORE IP and key for the service and subsequently ask for anonymous receiver paths to the MORE IP. Once a receiver path is available the data communication to service.anon can begin.

8.2.4 Performance Optimization and Analysis

The practical tests were done in a testbed with eight Linux hosts interconnected by a 100 Mbps network. The processors were either Pentium IV 2.2 GHz or Xeon 2.8 GHz.

The first experience with a first version of MORE was that it was not able to send more than a few kbps. The reason for this bad performance is the heavy use of asymmetric cryptography. This problem was solved by reusing the keys between sender and relay. This means that the sender does not select a different Diffie-Hellman value anytime it sends a packet. Now, the relays can cache the keys and use the cached value. Given a reuse of the key for 1,000 packets, the performance reaches 2.5 Mbps.

There were also issues to be solved and that needed an explanation. The end-to-end performance for data was still low. We analyzed the problem to be a problem with TCP. The multipath sending of packets creates an extreme variance in the arrival times of packets. A problem is the out-of-order delivery of packets, which becomes standard when multipath is used in such a dynamic way. There are also TCP over TCP issues. Similar problems can be found with TCP in wireless networks. So, the TCP flows suffer as congestion control thinks that these effects are caused by congestion and will reduce the sending rate. MORE mitigates the problem with a buffer and play-out smoothing of packets at the receiver. So, the TCP on top of MORE will see a slower yet in comparison more friendly network.

What was not emulated within the experiments is the impact of churn. Churn is the rate how nodes join and leave the network. While a join only imposes the networking cost for the initial bootstrap of the node, a leave invalidates many active paths and in most cases even more receiver paths that were stored at the service directory. Furthermore, nodes will still create corrupt paths as long as they did not recognize the failure of the respective node. We have to expect even more problems with packet loss and congestion control when using MORE as a Peer-to-Peer anonymization system on the real Internet. Thus, we can take the conclusion that MORE needs mechanisms to let information age and to remove outdated information. Furthermore, MORE needs to spread new information and keep its nodes uptodate. The right choices for corresponding methods and timeouts heavily depends on the network dynamics and yet unknown user behaviour. Correcting errors in advance seems also to be an appropriate choice as it is likely that some messages will fail

due to invalid paths. Forward Error Correction could be a good answer if single failures are likely for small sets of subsequent packets.

8.2.5 Networking Aspects

MORE interacts with network components and network resources. In this section we discuss middleboxes as well as congestion and heterogenious network access.

MORE is designed for the classic Internet. It does not assume the existence of middleboxes like Network Address Translators (NATs) and firewalls. A NAT usually translates local IP addresses of a subnet into a global IP. The various flows between the subnet and the Internet are usually separated using the ports on transport layer. The important property of NATs for MORE is that local computers cannot be reached from the Internet unless they either have a connection open or they interfered with the NAT. Now, MORE nodes connect different next hop nodes any time. So, if nodes are behind a NAT, they either will not be able to communicate or, more elegantly, will permanently have to perform NAT traversal algorithms. This is a severe problem and if NAT traversal is per packet practically prohibitive. Most other Peer-to-Peer approaches avoid this problem by using long-lived connections which is contrary to the MORE paradigm. Most likely NATs will also exist when IPv6 may replace IPv4, even though in theory they would not be necessary due to the large address space.

Heterogeneity is another problem that MORE currently does not solve. Peers have different capabilities. The computational overhead of MORE is immense due to the heavy use of asymmetric cryptography. Furthermore, bandwidth of devices and their access network may vary extremely. Another limit is bandwidth asymmetry of usually large downstream and comparably small upstream. MORE can only use the upstream and is thus limited to lower rates. This is, however, a general problem for Peer-to-Peer systems.

The effect of these and other issues is that networks or nodes may be congested. MORE needs a congestion control that adapts the bandwidth accordingly. This means that it uses the possible bandwidth, it does not severely overload links and nodes, and that the bandwidth is shared fairly among all peers. As MORE uses different paths for each message, the reaction to congestion of some nodes may not be relevant for the next paths via different nodes. So, there will be a general problem of a slow reaction.

One may think of possible approaches for MORE congestion control. MORE may limit the bandwidth to a rate per node that can be supported. Nodes may announce rates they are willing to relay, which could be used to further differentiate the datarate among the nodes. As this comes down to a least consensus and also needs incentive mechanisms, it does not seem to be the best choice. Another option is to transfer the classic AIMD that is also used within TCP to MORE. All nodes are rated with an initial rate. The overall rate provides a congestion window for each sender. If a packet successfully reached the receiver, the rate for each node on the path is increased. If a packets failed, the rate for all nodes on the path is reduced. The reduction needs to be larger than the increase in case of a successful transmission.

8.3 Security Analysis of MORE

In this section we first discuss general aspects of security for MORE. Building an anonymous system is more complex than it seems and, so, we will see a variety of potential attack vectors. We already know a mitigation for some of them. Future research on systems like MORE needs to resolve the open ones or find use-cases where they may be acceptable.

8.3.1 General Analysis

MORE operates between layer 3 and layer 2. This is not necessarily a fundamental part of the MORE architecture from our point of view, although it was highlighted in the papers. The benefit is that it may directly run with all kinds of legacy applications. The drawback is that this exposes all information of the application and the system's network stack. This allows to identify the system with fingerprinting. One basic difference to higher-layer approaches is that lower-layer information travels across the network. This information is sent to the receiver and not only exposed to a first hop.

The application is not aware that it has to avoid using identifying information. Proxy-based approaches propose to use a protocol sanitizer first before sending it to the anonymizer proxy. Even there this is a problem as sanitizers are limited in their capabilities and need to be defined for all application protocols. For many applications this information is even necessary and cannot be removed, which worsens the situation. It seems that to prevent information leakage through the application or operating system, one needs to write special anonymous applications that hardly expose information.

These threats are not MORE-specific and all approaches face these problems. MORE can reduce their impact if it would be implemented as proxy solution or even be added to anonymous applications.

As in the MORE design intermediate nodes are considered to be the attackers, MORE does not expose this additional information to them. As MORE avoids proxying to the Internet as a general service, it avoids many typical man-in-the-middle cases where this information could be used. However, the general threat against hidden servers by identifying them via system properties is not to be neglected and cannot be solved by other means than to turn off all services that communicate too much information, e.g. time as in the attack by Murdoch et al.[Murd06] against Tor hidden servers on the basis of clock skew deviations.

8.3.2 Integrity Protection and Replay Protection

It is usually recommended that for anonymizers each intermediate hop should be able to check the integrity of a message and to prevent replays. The latter of the goals can only be achieved with the other. If the integrity of a message is clear for any hop, no altered version of the message can be sent through the network. Slightly altering a message can help to circumvent the replay mechanisms like storing hash values of recently seen messages for some time.

Now, what about integrity protection in MORE? MORE only provides end-to-end integrity. As the sender cannot edit the header fields in the receiver path section, MORE is unable to provide hop-by-hop integrity. To achieve this, the sender would have to add a message authentication code (MAC) into the header of every hop. It cannot do this. The receiver is also not able to write the correct MAC values into the header fields for its receiver path. At the time of creation of the receiver path it is not even clear who will use it, despite what it will send. Thus, the concept of path concatenation does not allow integrity to be checked on all intermediate hops.

As the integrity problem is caused by the receiver path section, one may add integrity check into the header of the sender path and with this tell each hop if it is on a sender or receiver path section. This would allow a partial solution for integrity in MORE, but the leakage of information about the positioning is also not desirable.

Replay protection is closely related to integrity checks. To prevent replays, MORE relays store hash values of messages that they have recently seen. If a message with the same hash value reaches the relay, the relay will discard the message. Due to limitations in storage and computational costs, MORE only stored hash values for a few seconds.

However, since MORE cannot protect the message integrity on all intermediate hops, it cannot stop an attacker to replay a slightly modified or tagged message. Thus, replay protection for MORE does not work. MORE is prone to replay and tagging attacks.

We are not aware of a good solution to fix this problem. There is, however, an argument that mitigates the effectiveness of these attacks. As MORE is a Peer-to-Peer approach, it suffers from potentially short peer lifetimes, and the invalidation of paths due to peers leaving the system. With respect to these attacks, the dynamics of a Peer-to-Peer network may be beneficial. If paths do not exist anymore, replays do not work. This argument may not be strong. There is, however, a further argument. Paths of MORE are long and of random length. The last hop cannot decide wheather it is last hop or not. As these attacks primarily aim at identifying subsequent hops and path sections, they may not necessarily help to identify sender or receiver. Still, they may help statistical attacks that we discuss in later sections.

8.3.3 Naming

Naming is a crucial issue. In particular, if names have meanings and may imply something. MORE implements a DNS-like naming system with the service directory as name server and authority. Thus, names do have value and attacks on the naming could be profitable for an attacker.

One of the major problems should be domain grabbing where an attacker registers a lot of name under her name and blocks them for other usage. MORE might have to consider a limit on the number of domains per identity. However, MORE cannot really limit the number of identities and, therefore, only slightly increase the computation effort to steel enough domains.

The problem is also related to the definition of lifetime of domains. MORE does not yet have a concept for it. Given a short lifetime, nodes have to register quite often. This may reduce the grabbing, but could be used to catch domains owned by others.

In case of a DHT service directory, malicious DHT nodes could cooperate with the attacker. Domain hijacking and impersonation become more likely. As the relation IP, DNS, and keys is arbitrary, it is not possible to protect via the keys. In particular, the keys can also change from time to time. Thus, the changes seen by the service directory do not necessarily describe illegitimate operations.

One could, of course, differentiate between timeouts and use keys to protect change requests. So, the first timeout when a refresh is missing would put the domain name on hold and disable it until the expected refresh arrives. If this does not happen within a longer timeout period, then the domain would be free again and could be registered by other users. In general, such solutions do not solve the naming issues. In particular, we want short timeouts to fight domain grabbing, and we want long timeouts to fight impersonation.

8.3.4 Service Directory

The service directory is an important anchor of trust in the MORE architecture. It is in control of the bootstrapping and can mislead nodes, no matter if new or old. This could be an Eclipse attack by providing only attacker nodes to a victim.

The service directory can also deny the existence of a node or service. In particular, it could deny the knowledge of receiver path sections to the node or service. Without receiver path section, no contact can be initiated to the node or service by other nodes.

For the bootstrapping to the network, a new node needs to ask for other nodes in order to be able to build paths. MORE assumes that the service directory gives the node a set of nodes and follow-up communication is anonymous. The node contacts the service directory using a sender path for sender anonymity. The service directory then assigns a MORE IP address to the node.

There are also various ways to find out about the real IP and MORE-IP identities. The first variant is with the help of colluding nodes. The service directory only tells its fellow attacker nodes and then break the subsequent request to the service directory as only attacker nodes are used. The second variant is more subtle. Hiding the sender path from the service directory is generally problematic as the service directory knows the limited knowledge of a node, e.g. the initially known 4-5 nodes. Therefore it can break the hidden service construct, with the help of this limitation: one of the next requests with these nodes is probably the node hosting the service. Generally, the timing is a problem as the MORE-IP needs to be created timely after the first contact. So, even ignoring the knowledge of the nodes, the sequence and timing of the IP requests, provides valuable information and most likely the relation IP and MORE-IP.

Another option for the service directory is a Peer-to-Peer service directory. This directory basically is prone to all Peer-to-Peer-related attacks. Malicious and honest nodes are then part of the service directory. An advantage of a Peer-to-Peer service directory would be that requests spread among all nodes and that attackers may not see related requests.

8.3.5 Definition of Receiver Path Processing

In the article [LPWN+07] the description of the path concatenation is not very precise. Actually, there are no documents that exactly specify the system. One may read the path concatenation as if the sender did not chose but knows the receiver path nodes except for the last one. Even if the sender only knows the public keys of the receiver relays at first, the sender may use this to determine the nodes on the path by looking up relays until all keys are assigned to their corresponding relay node.

If the sender knows the receiver path, it can break the anonymity of the receiver. Probabilistically, it is with $p = \frac{1}{n}$ the last node in the path and can, thus, directly identify the receiver.

Another issue that may lead to problems is if the header leaks the position due to fixed size and intermediate cutting of the headers for the previous nodes. This may help an attacker to identify its position. The basic countermeasure we discussed is to append the removed header at the end of the packet. This is, however, not specified or in the corresponding paper.

The packet length allows to draw conclusions on the position. A long packet may indicate being one of the first, a smaller packet being one of the last nodes. In particular the size also limits the number of layers and it can have implications on the number of layers. Thus, simplifications not to hide the packet size can help to execute predecessor and successor attacks as well as possibly help also other attacks.

8.3.6 Patterns in MORE

In this section we discuss problems in MORE related to patterns and information leaked to nodes in the MORE system. This is important as all nodes see parts of a communication and thus, information may be flooded to all nodes instead of only a few in a path. We end this text with a revisiting the problem of forward secrecy in MORE.

Some basics, n is the number of nodes in MORE. Say p is the average length of one path section (e.g. sender path section). A typical value would be $p_{avg} = 3$. u_k is the average

number of uses of one DH key. Each node has a temporary DH key for each other node which it uses when the other node is used as relay. The key of a relay, however, is not temporary, but valid for its MORE session.

Attacks by a single node

In this section, we discuss attacks that can be executed by a single node.

Correlating key and sender ID = first hop identification

Attack

A sender uses all nodes in the system as relay. As a consequence all nodes could be on a path as first hop or as second, third, ... On average, each node will be first hop for every nth packet.

So, let us examine a flow from the sender to an arbitrary relay. The sender will send on average $1/n$ of her packets through this node and other nodes will use the sender as relay. We are now interested in the traffic mix that is coming from the sender. We can distinguish the senders of the packets from the keys that are cached. The sender will send $rate_{sender}/n$ directly to the relay. The other senders together will send on average

$(2p) * rate_{avg}/n$

through the sender. This rate is divided onto all $n-1$ nodes. Thus, one will expect roughly

$(2p) * rate_{avg}/n^2$

rate from other nodes. Assuming the sender is close to the average rate, this is roughly

$(2p/n) * rate_{sender}$.

With typical values, say $p = 3$ and $n = 1000$, the sender rate is roughly 150 times higher than the rate of arbitrary other nodes that it uses as relays. This analysis is not only related to rates. Even better, if the DH keys are used on average u_k times, then the sender is using her one in the same way more often from herself to the relay than other nodes use their keys via her to the relay.

So, we end up with a variant of the predecessor attack. Using this information about key usage or rates, we have a strong indication that the sender is the sender of all packets with certain keys.

Mitigation

The perfect solution would be to use $n/(2p)$ keys in parallel for each first hop. This is not practical. Updating first hop keys more often and keep a limited number in parallel could help, but needs further analysis. In general, this is also be a threat to the attacker path problem (Sender $->$ only attackers $->$ Receiver) as this information could be used to determine a full attacker path with high probability. The security of MORE depends to some degree on the hardness of determining for a group of relays whether they form the complete path or not.

The main arguments, though not completely satisfying, are probably that such an attack is costly and frequent key changes make this even worse as many relays need to coordinate their suspicions. As computers become faster, the key changes might be done at higher rate or even for every packet. Moreover, there is always a time necessary to determine the key of the sender after it changed. To store all the packets before the key is identified is known is also expensive. The rather long paths in MORE also reduce the number of such events. It is mainly a threat to long-lived high-volume connections.

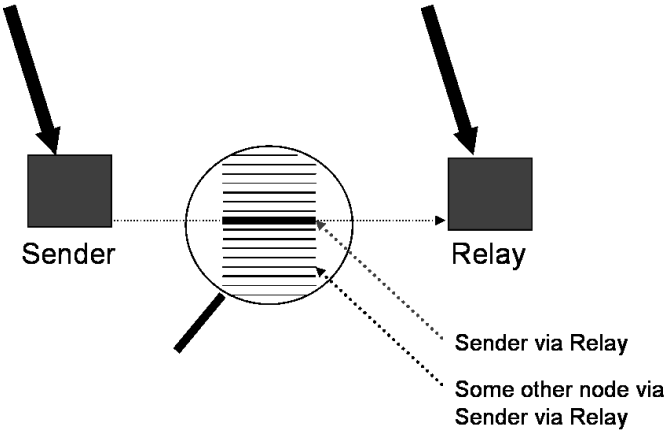


Figure 8.11: Each flow is identified on the basis of its key. Sender and arbitrary relays differ in their statistical properties. The sender is more likely to be seen for its own flows as predecessor than for flows of other senders. It sends its own flows to the relay at a higher rate.

Aonther potential mitigation could be to order nodes like I2P and only use them in this order, when creating a sender path. It distorts the assumptions of this attack. However, most likely, this order can be misused to help the peers in the first places of this order to even better attack the sender. There is a tradeoff, a real first node is more likely to be seen with a high peak. However, as the rate is unknown, this may not be enough knowledge. As the first node in the ordering only sees the correct first node, it can conclude from the 100 % node and ID relation that that its predecessor is the sender. Thus, at least for MORE the concept is not suitable.

Identifying all packets by sender using cached keys

Attack

The previous attack showed that a relay can identify if it is first hop. Thus, it can determine the relation sender ID and IP. It can use this information for all packets, not only for the ones where it is the first hop. Thus, on all positions on the sender path section this node can identify the sender. As this node is arbitrary any node can do so.

Mitigation

The best solution is to have separate keys for using a node as first/last hop and intermediate hop.

Correlating sender and receiver

Attack

With respect to the receiver, a similar analysis can be used to identify the receiver on relays on the receiver path section. As a consequence, any node can launch an intersection attack on its data about rates from the sender and to the receiver.

Mitigation

If only first hop and last hop information can be used, then MORE already mitigates this attack. This is achieved by using the message splitting idea to divide the rate not equally over the first/last hop relays, but with some statistical distribution, e.g. close to the exponential distribution.

8.3.7 On patterns inserted by active attackers

With patterns of an active attacker we consider on/off periods where the attacker drops or forwards all packets of a node that pass through an area the attacker controls.

On patterns inserted by local active attacker

Attack

In this case the attacker controls the local access network of the node under attack, say the sender. This attacker is actually not in the attacker model of MORE as it is an rather strong attacker.

The first hop nodes directly see the patterns if there is enough traffic to them to be able to speak of on and off periods. Thus, the period of the pattern needs to be $>> n$ in number of packets.

As the first hop node does not need these patterns to analyze traffic from the sender, we have to look at intermediate hops and the last hop.

Intermediate hops try to find these patterns to connect the intermediate key to the first hop key and therefore the IP of the sender. The last hop may try to connect a potential receiver to the sender.

Due to the MORE design all nodes play the roles of first hop, intermediate hop, and last hop. While the first hop nodes may have their own information from observations mentioned in the previous sections to get some information about patterns. For intermediate nodes it is hard to determine that a key is not used because of a pattern or because the key is simply changed as the nodes should change the keys to a relay frequently or simply randomness or traffic patterns of the communication.

Mitigaton

The main purpose of the key caching was to make MORE efficient. However, the benefit of caching is only small once the cost of a key change is smaller than the cost for computing the $u_k - 1$ other packets. If $u_k = 100$, then the key reuse is 99% and according to the measurement, we are still close to the 100% key reuse case with its 2,5 Mbps. So, u_k can be kept relatively small (compared to potentially way more users).

Having more connections open using the same keys further distorts the traffic analysis for intermediate and last hops.

On patterns inserted by first hop node

Attack

In this case one relay inserts such patterns for packets from the sender. It is similar to the case of the active attacker in the last section. However, the patterns are factor n weaker.

Mitigaton

Similar to the last section, keep u_k small enough.

8.3.8 Ordering of nodes does not prevent predecessor attacks in MORE

I2P implements a local ordering scheme to fight predecessor and successor attacks. The basic idea is that distorting the statistics of nodes seen as predecessor for certain requests, makes these attacks harder. In the I2P chapter we have already seen that this is doubtful. In this section, we show that this does not help against predecessor attacks in MORE.

MORE is predestined for predecessor attacks as it frequently changes paths (per message), keys can be used to identify flows, and it is a Peer-to-Peer system that uses all nodes as relays. The ordering idea of I2P means that each node creates a virtual order of all nodes. When a node creates a path, it always orders the participants according to the position in its virtual order.

The first node in the order will thus see only one node as predecessor (= 100 % of the cases). The second node will see two, the third three, etc. For all nodes in the front of the order, the real sender will still be the way most likely node as it is most likely that other nodes later in the order were selected. As a consequence, with high probability nodes early in the order will be able identify the sender. Other nodes will see from the statistics that they are not in a good position. Relating these results with the pattern statistics in our attacks of this chapter. The rate of the sender was an order stronger for the sender than for arbitrary nodes. We may, thus, assume that not only the first few nodes will be able to stage this attack, but that many more nodes are still in good position.

8.3.9 On Forward Secrecy

MORE does not support forward secrecy for its messages. If an observer breaks the DH key of a relay, the observer can read all packets to the relay, the new ones as well as old messages it has recorded. This could be used for traffic analysis. With this information along this attacker is, however, not more powerful with respect to traffic analysis than a relaying attacker that owns a fraction of the relays.

Most relays are peers that are only online for some time and will thus leave after some time and will use a different key when they join again. So, the problem of a broken key of a relay is limited in time. Here, the use of the DH instead of standard PKI is even an advantage, as a relay could simply change its key, distribute it and the attacker can not read the messages again. There is no such mechanism in MORE though, except for leaving and joining after a while.

With respect to the end-to-end delivery of contents, MORE can achieve forward secrecy, although this is not yet included. Forward secrecy for a message flow between sender and receiver could be achieved by adding a real DH exchange between them. One could also consider this a task for a security application on top of MORE that needs this kind of security, in particular if one believes into the argument that MORE nodes will only use their ('longterm') relay key for a short period of time as they would delete it when leaving the network. Thus, key changes may alleviate the need for forward secrecy. In case MORE has a flaw in the key generator, the loss of longterm keys may not stop the attack as they might be reconstructable. Although a similar weakness has happened in a real cryptographic library, we would rate this as a rather unlikely way to attack MORE.

8.4 Relation to Architecture and Design

The concept of MORE is radical as a new path for every packet is proposed. This was meant to fight attacks that are based on tracing patterns. With this concept patterns cannot be followed along a path. However, patterns about the end-to-end communication do still exist. To our best knowledge, none of the inventors of MORE is working on the

system anymore. As this is still a fascinating idea, we hope that there will be future research to resolve the yet unsolved issues.

With respect to design we learned that asymmetric cryptography is still very expensive and should not be used extensively. Methods with reasonable short keys are necessary to avoid extensive overhead. Key establishment, however, conflicts with the idea to reduce state and avoid connections. Here. the key caching in MORE is a good compromise. Each relay may trade-off state for key caching and computation.

The idea of going different paths on a per packet basis, conflicts with TCP streams, so it may not be a good choice for general purpose content. This extensive multipath idea is not very well suited to deal with congestion and to provide a simple flow and congestion control.

We have also seen that aiming on one attack does not solve a general security problem. MORE is still prone to some pattern attacks. Hop-by-hop integrity protection is considered to be important to avoid replay attacks, yet MORE cannot support it. The design choice of MORE to go on layer 3 makes it difficult to anonymize the overall communication due to higher-layer leakage. This may need more analysis to find appropriate answers. The MORE anonymization concept, however, can run on any layer.

As most anonymizers, MORE has to tweak its performance and overall performance is not a completely resolved issue. One may also find other scenarios that could benefit from similar multipath methods, e.g. for resilience.

9. Conclusions

In this thesis we presented Peer-to-Peer systems as well as approaches for secure and anonymous system design. Peer-to-Peer systems have more aspects than abstract routing algorithms or DHT functionality. They also have more use-cases than Filesharing or Instant Messaging. We divided their functionality into many components. On abstract level, this is server, client, and routing functionality. There are many more subcomponents that deal with a variety of issues and they may provide functionality within any of these abstract components. We also discussed the optimization of Peer-to-Peer systems. It may not always be necessary and performance optimization may also conflict with other goals. Nonetheless, load is usually imbalanced and messages may take long detours. Both issues can be improved, although one should not expect perfect results.

Security was also the focus of many chapters in this thesis. Most prominent the chapter on Authentication and Authorization for Peer-to-Peer systems. The key idea is to use the social clustering of nodes to build a domain structure within the Peer-to-Peer network. Since no global authority can help, trust is established over time or per configuration by skilled administrative users. Two authentication protocols were presented, the protocol of Holz (PDP-A) as well as a protocol that uses fewer messages at the price of not providing freshness for all parties. This approach positioned itself between centralized highly-secure concepts and decentralized concepts with limitations in security. It can provide full security if the necessary trust exists, but it also resolves the frequent cases when this is not the case. Future work needs to adapt applications to make them interact with such a system. Trust establishment also needs more research and in particular more precise definitions on how to weight information and how much trust is associated with a certain amount of interactions. Further studies on the security of such systems is also necessary as well as a standardized risk assessment. A standardized API should also be defined and one may study how to integrate applications that do not support feedback and risk assessment. Other security results include that the performance of cryptographic operations can no longer be seen as a reason not to implement security in networked systems. The use of public key or identity-based cryptography should nonetheless be done carefully as they still are expensive – but by far not so much as to avoid them. Peer-to-Peer systems face a variety of additional security threats. Reputation systems can be attacked with the cheapriding attack. The cheapriding attack can be fought by adapting the benefit in reputation to the cost or revenue of an operation. This can not stop related strategy-change attacks though.

Finally, we looked at the design of anonymous systems. Anonymity is hard to achieve and all systems with realtime requirements have some weaknesses. I2P as an anonymity system

from the open-source community rather than from the academic world has been studied. The idea of Garlic Routing is fascinating, but only partially used by I2P in reality. There should be further studies on how to implement Garlic Routing and on its security. For I2P, it has been shown that methods like their ordering of peers to prevent predecessor attacks do not achieve their goal. It may be even counterproductive to do this. The MORE system tried to fight another class of dangerous attacks, the attacks on the basis of patterns. While it is true that MORE reduces the amplitude of patterns, participants in the network can still use them to do successful predecessor attacks. Hop-by-hop integrity protection is also a problem that cannot be solved within the MORE system and tagging attacks are possible. Future work could define and analyze a complete system on the idea, most likely above layer 3. MORE's performance security tradeoffs may need some changes. The radical idea of MORE to route each packet along a different path is nonetheless exciting and other use-cases may also be found. One could use this for resilience, in special delay-tolerant networks, or as a kind of source routing in heterogeneous networks with multiple connectivity domains.

To conclude, the design of Peer-to-Peer systems is not an open issue where designers need to be innovative to build the system. We have referred to many solutions for subproblems in the design process and gave guidelines to build such systems. If the prerequisites for typical security methods hold, then this is also true for the integration of elementary security services. Security is primarily a question of control and ways to infer correct decisions. The combination and application-specific use of security protocols can, however, add weaknesses to even secure protocols. Security on lower layers of the protocol stack is only partially a solution for security goals on higher layers. Anonymity is even harder to achieve and most anonymity systems do not fight strong attackers. It is hard to overcome attackers that can observe packet flows. Privacy within many applications, however, may not need such strong mechanisms and it might also be achieved sufficiently in systems where privacy is not the original purpose of the system.

Literature

[AAGG+05] K. Aberer, L. Onana Alima, A. Ghodsi, S. Girdzijauskas, M. Hauswirth und S. Haridi. The essence of P2P: A reference architecture for overlay networks. In *The Fifth IEEE International Conference on Peer-to-Peer Computing, Konstanz, Germany.*, 2005.

[AbNe96] M. Abadi und R.M. Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering* 22(1), 1996, S. 6–15.

[AdHu02] L. A. Adamic und B. A. Huberman. Zipf's Law and the Internet. *Glottometrics* Band 3, 2002, S. 143–150.

[anon] anonymous. I2P. http://www.i2p2.de.

[AVIS07] AVISPA Project. Automated Validation of Internet Security Protocols and Applications (homepage). http://www.avispa-project.org/, July 2007.

[BaAl99] Albert-László Barabási und Réka Albert. Emergence of scaling in random networks. *Science*, 1999.

[BaSc04] S. A. Baset und H. Schulzrinne. An analysis of the Skype Peer-to-Peer Internet telephony protocol. *arXiv:cs/0412017v1, http://arxiv.org/abs/cs/0412017v1*, 2004.

[BaSc06] S. A. Baset und H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *25th IEEE International Conference on Computer Communications (INFOCOM2006)*, 2006.

[BeEM04] K. Berket, A. Essiari und A. Muratas. PKI-based security for Peer-to-Peer information sharing. In *Proceedings of the 4th IEEE International Conference on Peer-to-Peer Computing, Zurich, Switzerland (P2P 2004)*, 2004, S. 45–52.

[BiFa06] Philippe Biondi und Desclaux Fabrice. Silver Needle in the Skype. *Black Hat Europe Conference*, 2006.

[BoFr01] D. Boneh und M. Franklin. Identity Based Encryption from the Weil Pairing. In *Advances in Cryptology - Proceedings of CRYPTO 2001*, Band 2139 der *Lecture Notes in Computer Science*. Springer Verlag, 2001, S. 213–229.

[BoMa03] C. Boyd und A. Mathuria. *Protocols for authentication and key establishment.* Information Security and Cryptography. Springer. 2003.

[Boyd93] C. Boyd. Security architecture using formal methods. *IEEE Journal on Selected Topics in Communications* Band 11, 1993, S. 694–701.

[ByCM03] John Byers, Jeffrey Considine und Michael Mitzenmacher. Simple Load Balancing for Distributed Hash Tables. 2003, S. 80–87.

[ByCo03] John Byers, Jeffrey Considine und andere. Simple Load Balancing for DHTs. In *Proc. of IPTPS*, Berkeley, CA, 2003.

[CDGR+02] M. Castro, P. Druschel, A. Ganesh, A. Rowstron und D.S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Dezember 2002.

[CDHR02] M. Castro, P. Druschel, Y.C. Hu und A. Rowstron. Exploiting network proximity in Distributed Hash Tables. In O. Babaoglu, K. Birman und K. Marzullo (Hrsg.), *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, 2002, S. 52–55.

[CDHR03] Miguel Castro, Peter Druschel, Y. Charlie Hu und Antony Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. In *Microsoft Technical report MSR-TR-2003-52*, 2003.

[CGMA85] B. Chor, S. Goldwasser, S. Micali und B. Awerbuch. Verifiable Secret Sharing and achieving simultaneity in the presence of faults. *Proceedings of the 26th IEEE Annual Symposium on Foundations of Computer Science*, 1985, S. 383–395.

[CKSH+06] T. Condie, V. Kacholia, S. Sankararaman, J. Hellerstein und P. Maniatis. Induced Churn as Shelter from Routing-Table Poisoning. In *Proc. Network and Distributed System Security Symposium, San Diego, CA, USA, Feb. 2006*, 2006.

[Cohe] Bram Cohen. BitTorrent. http://www.bittorrent.com.

[Cohe03] Bram Cohen. Incentives build robustness in bittorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[Crem06] C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols.* Dissertation, University Press Eindhoven, 2006.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley und Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science* Band 2009, 2001, S. 46.

[DCKM04] Frank Dabek, Russ Cox, Frans Kaashoek und Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *In SIGCOMM*, 2004, S. 15–26.

[DeAb04] Zoran Despotovic und Karl Aberer. A Probabilistic Approach to Predict Peers' Performance in P2P Networks. In *Proc. of CIA'04*, 2004.

[DiFM00] Roger Dingledine, Michael J. Freedman und David Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In H. Federrath (Hrsg.), *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability.* Springer-Verlag, LNCS 2009, July 2000.

[DiHa06] Jochen Dinger und Hannes Hartenstein. Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06)*, Washington, DC, USA, 2006. IEEE Computer Society, S. 756–763.

[DiMS04] Roger Dingledine, Nick Mathewson und Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, August 2004. S. 303–320.

[Ding00] Roger Dingledine. The free haven project: Design and deployment of an anonymous secure data haven. Diplomarbeit, MIT, June 2000.

[DLLKA05] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek und R. Anderson. Sybil-resistant DHT routing. In *In Proc. ESORICS, pages 305-318, 2005.*, 2005.

[Douc02] J. R. Douceur. The Sybil Attack. In *Peer-to-Peer Systems: 1st International Workshop, Cambridge, MA, USA (IPTPS 2002). Revised Papers*, 2002, S. 251–260.

[DoYa83] D. Dolev und A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory* 29(2), 1983, S. 198–208.

[DZDK+03] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz und I . Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[(EFF09] Electronic Frontier Foundation (EFF). Tor Website, 2009.

[ElSc00] C. Ellison und B. Schneier. Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure. *Computer Security Journal* 16(1), 2000, S. 1–7.

[Feld87] P. Feldman. A practical scheme for non-interactive Verifiable Secret Sharing. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, 1987, S. 427–437.

[FNKC07] A. Fessi, H. Niedermayer, H. Kinkelin und G. Carle. A Cooperative SIP Infrastructure for Highly Reliable Telecommunication Services. In *Proceedings of the ACM conference on Principles, Systems and Applications of IP Telecommunications, New York, USA (IPTComm 2007)*, 2007.

[FrGa03] Pierre Fraigniaud und Philippe Gauron. The Content-Addressable Network D2B, 2003.

[fSta] The R Project for Statistical Computing. http://www.r-project.org.

[GeRBF+03] L. Garcés-erice, K. W. Ross, E. W. Biersack, P. A. Felber und G. Urvoy-keller. Topology-Centric Look-Up Service. In *in COST264/ACM Fifth International Workshop on Networked Group Communications (NGC.* Springer, 2003, S. 58–69.

[GGGR+03] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker und I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 2003. ACM Press, S. 381–394.

[GuSh07] P. Gupta und V. Shmatikov. Security analysis of Voice-over-IP protocols. In *20th IEEE Computer Security Foundations Symposium, Venice, Italy (CSF '07)*, 2007, S. 49–63.

[Gutm02] Peter Gutmann. PKI: It's not dead, just resting. *IEEE Computer* 35(8), August 2002, S. 41–49.

[Heis07] Heise. Anonymisierungsnetz Tor abgephisht, Teil 2, 2007.

[HMMB+09] Christian Hübsch, Christoph P. Mayer, Sebastian Mies, Roland Bless, Oliver P. Waldhorst und Martina Zitterbart. Reconnecting the Internet with ariba: Self-Organizing Provisioning of End-to-End Connectivity in Heterogeneous Networks. In *Demo at ACM SIGCOMM 2009, Barcelona, Spain*, 2009.

[HNHC08] Ralph Holz, Heiko Niedermayer, Peter Hauck und Georg Carle. Trust-Rated Authentication for Domain-Structured Distributed Systems. In *Proc. 5th European PKI Workshop: Theory and Practice (EuroPKI 2008)*, Trondheim, Norway, 2008.

[Holz07] Ralph Holz. Secure Domain-based Peer-to-Peer Networks. Diplomarbeit, University of Tübingen, October 2007.

[IiHK04] Takuji Iimura, Hiroaki Hazeyama und Youki Kadobayashi. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *Proceedings of the 3rd Workshop on Network and System Support for Games, NETGAMES 2004*, Portland, OR, August 2004.

[KaKa03] Frans Kaashoek und David R. Karger. Koorde: A Simple Degree-optimal Hash Table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, 2003.

[Ka+ot03] Sepandar D. Kamvar und andere. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proc. of WWW'03*, 2003.

[KaRu02] D. Karger und M. Ruhl. Finding nearest neighbors in growth-restricted metrics, 2002.

[KaRu04] David Karger und Mathias Ruhl. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In *Proc. of IPTPS*, San Diego, CA, 2004.

[KLLP+97] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine und Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, New York, NY, USA, 1997. ACM, S. 654–663.

[KMNC05] Andreas Klenk, Marcus Masekwosky, Heiko Niedermayer und Georg Carle. ESAF - an Extensible Security Adaptation Framework. In *NordSec 2005 - The 10th Nordic Workshop on Secure IT-systems*, October 2005.

[KoLR09] Michael Kohnen, Mike Leske und Erwin Rathgeb. Conducting and Optimizing Eclipse Attacks in the Kad Peer-to-Peer Network. In *Proc. 8th International IFIP-TC 6 Networking Conference (Networking 2009), Aachen, Germany, May 11-15, 2009*, Band 5550/2009 der *LNCS*. Springer, May 2009, S. 104–116.

[LaNW05] Olaf Landsiedel, Heiko Niedermayer und Klaus Wehrle. An Infrastructure for Anonymous Internet Services. In *International Workshop on Innovations In Web Infrastructure (IWI 2005), 14th International World Wide Web Conference (WWW 2005), Chiba/Tokyo, Japan*, 2005.

[Lowe97] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop, Rockport, MA, USA (CSFW '97)*, 1997.

[Lowe99] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security* 7(2), 1999, S. 89–146.

[LPWN+07] Olaf Landsiedel, Alexis Pimenidis, Klaus Wehrle, Heiko Niedermayer und Georg Carle. Dynamic Multipath Onion Routing in Anonymous Peer-To-Peer Overlay Networks. In *IEEE Global Communication Conference (GlobeCom), Washington D.C.*, 2007.

[LRNW+05] Olaf Landsiedel, Simon Rieche, Heiko Niedermayer, Klaus Wehrle und Georg Carle. Anonymous IP-Services via Overlay Routing. In *5. Würzburger Workshop IP Netzmanagement, IP Netzplanung und Optimierung, Würzburg, Germany*, 2005.

[Lu04] Honghui Lu. Peer-to-Peer Support for Massively Multiplayer Games. In *Proceedings IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[MaMa02] P. Maymounkov und D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[MaNR02] Dahlia Malkhi, Moni Naor und David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC*, 2002, S. 183–192.

[Mao04] W. Mao. An identity-based non-interactive authentication framework for computational grids. Technischer Bericht, Trusted Systems Laboratory, HP Laboratories Bristol, 2004.

[MeOV97] A. J. Menezes, P. C. van Oorschot und S. A. Vanstone. *Handbook of applied cryptography.* CRC Press. 1997.

[Mich98] Aviel D. Rubin Michael K. Reiter. Crowds: Anonymity for Web Transactions. *ACM TISSEC*, 06 1998.

[MiDr04] A. Mislove und P. Druschel. Providing Administrative Control and Autonomy in Structured Peer-to-Peer Overlays. In *3rd International Workshop on Peer-to-Peer Systems, La Jolla, CA, USA (IPTPS 2004). Revised selected papers*, 2004, S. 26–27.

[Milg67] Stanley Milgram. The Small World Problem. *Psychology Today*, 1967.

[Mitz96] Michael David Mitzenmacher. The power of two choices in randomized load balancing. Technischer Bericht, IEEE Transactions on Parallel and Distributed Systems, 1996.

[MLMB] Alberto Medina, Anukool Lakhina, Ibrahim Matta und John Byers. BRITE: Boston University Representative Internet Topology Generator. http://www.cs.bu.edu/brite/.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta und John Byers. BRITE: An Approach to Universal Topology Generation. *MASCOTS* Band 00, 2001, S. 0346.

[mStp] Stunnel multiplatform SSL tunneling proxy. http://stunnel.mirt.net.

[MuDa05] S. J. Murdoch und G. Danezis. Low-Cost Traffic Analysis of TOR. In *In Proc. of the IEEE Symposium on Security and Privacy*, May 2005.

[Murd06] S. J. Murdoch. ot or not: revealing hidden services by their clock skew. In *In Proc. of 13th ACM conference on Computer and Communications Security(CCS)*, October 2006.

[Naps] Napster. Napster history on http://en.wikipedia.org/wiki/Napster.

[NaTY03] M. Narasimha, G. Tsudik und J. H. Yi. On the utility of distributed cryptography in P2P and MANETs: the case of membership control. In *Proceedings of the 11th IEEE International Conference on Network Protocols 2003*, 2003, S. 336–345.

[NRWC05] Heiko Niedermayer, Simon Rieche, Klaus Wehrle und Georg Carle. On the Distribution of Nodes in Distributed Hash Tables. In *Proceedings of Workshop Peer-to-Peer-Systems and -Applications, KiVS 2005*, Band 61 der *LNI*, Kaiserslautern, Germany, March 2005. GI, S. 193–196.

[NWSC04] Heiko Niedermayer, Klaus Wehrle, Thomas Schreiner und Georg Carle. Considering Security in Distributed Hash Tables. In *Abstract at 4th Würzburger Workshop IP Netzmanagement, IP Netzplanung und Optimierung, July 2004, Würzburg, Germany*, 2004.

[Open] OpenVPN. http://www.openvpn.org.

[Over] Overlay-Networks.info. http://www.overlay-networks.info.

[OvSy06] L. Overlier und P. Syverson. Locating Hidden Servers. In *In Proc. of IEEE Symposium on Security and Privacy (SP)*, May 2006.

[Pede91] T. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology - EUROCRYPT '91*, Band 547, 1991, S. 522–526.

[RaBO89] T. Rabin und M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the 21 Annual ACM Symposium on Theory of Computing*, 1989, S. 73–85.

[ReSt99] Michael K. Reiter und Stuart G. Stubblebine. Authentication Metric Analysis and Design. *ACM Transactions on Information and System Security* Band Vol. 2, No. 2, 1999, S. 138–158.

[RFHK01] S. Ratsanamy, P. Francis, M. Handley und R. Karp. A scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM Conference 2001*, 2001, S. 161–172.

[RFNP+06] Simon Rieche, Marc Fouquet, Heiko Niedermayer, Leo Petrak, Klaus Wehrle und Georg Carle. Peer-to-Peer-based Infrastructure Support for Massively Multiplayer Online Games. Technical Report WSI-2006-04, Wilhelm-Schickard-Institute for Computer Science, University of Tübingen, Tübingen, Germany, August 2006.

[RFNW+05] Simon Rieche, Marc Fouquet, Heiko Niedermayer, Klaus Wehrle und Georg Carle. On the Use of Structured Peer-to-Peer Systems for Online Gaming. In *5. Würzburger Workshop IP Netzmanagement, IP Netzplanung und Optimierung*, Wuerzburg, Germany, July 2005.

[RiPW04a] Simon Rieche, Leo Petrak und Klaus Wehrle. A Thermal-Dissipation-based Approach for Balancing Data Load in Distributed Hash Tables. In *Proceedings of 29th Annual IEEE International Conference on Local Computer Networks - LCN'04*, Tampa, FL, November 2004.

[RiPW04b] Simon Rieche, Leo Petrak und Klaus Wehrle. A Thermal-Dissipation-based Approach for Balancing Data Load in Distributed Hash Tables. In *Proceedings of 29th Annual IEEE International Conference on Local Computer Networks - LCN'04*, Tampa, FL, November 2004.

[RiSc04] Chris Riley und Christian Scheideler. Guaranteed Broadcasting Using SPON: Supervised P2P Overlay Network. In *International Zürich Seminar on Communications*, 2004.

[RLSK+03] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp und Ion Stoica. Load Balancing in Structured P2P Systems. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems - IPTPS'03*, Berkeley, CA, Februar 2003.

[RNGW05] Simon Rieche, Heiko Niedermayer, Stefan Goetz und Klaus Wehrle. *Peer-to-Peer Systems and Applications*, Band 3485 der *Lecture Notes in Computer Science, LNCS*, Kapitel Reliability and Load Balancing in Distributed Hash Tables, S. 119–135. Springer, Heidelberg, Germany. September 2005.

[RoDr01] Antony Rowstron und Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM International Conference on Distributed System Platforms, Heidelberg, Germany*, 2001.

[RWFN+07] Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Leo Petrak und Georg Carle. Peer-to-Peer-based Infrastructure Support for Massively Multiplayer Online Games. In *Proceedings of 4th Annual IEEE Consumer Communications and Networking Conference (CCNC 2007)*, Las Vegas, Nevada, USA, January 2007.

[RWFN+08a] Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Timo Teifel und Georg Carle. Clustering Players for Load Balancing in Virtual Worlds. *International Journal of Advanced Media and Communication (IJAMC)*, 2008.

[RWFN+08b] Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Timo Teifel und Georg Carle. Clustering Players for Load Balancing in Virtual Worlds. In *Proceedings of 1st International Workshop on Massively Multiuser Virtual Environments at IEEE Virtual Reality 2008 (MMVE 2008)*, Reno, Nevada, USA, March 2008.

[SaTY03] N. Saxena, G. Tsudik und J. H. Yi. Experimenting with admission control in P2P. In *International Workshop on Advanced Developments in Software and Systems Security (WADIS)*, 2003.

[SCDR04] Atul Singh, Miguel Castro, Peter Druschel und Antony Rowstron. Defending against Eclipse attacks on overlay networks. In *Proc. of SIGOPS European Workshop, Leuven, Belgium, Sept. 2004*, 2004.

[SEAA+09] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri und Lorrie F. Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *Proceedings of the 18th Usenix Security Symposium*, August 2009.

[SeMu05] Andrei Serjantov und Steven J. Murdoch. Message Splitting Against the Partial Adversary. In *Proceedings of Privacy Enhancing Technologies Workshop*, Cavtat, Croatia, June 2005.

[Sham79] A. Shamir. How to share a secret. *Communications of the ACM* 22(11), 1979, S. 612–613.

[Sham85] A. Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology – Proceedings of CRYPTO '84*. Springer, 1985, S. 47–53.

[SiMo02] E. Sit und R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[Skyp08] Skype. Skype (homepage). http://www.skype.com, September 2008.

[SMKK+01] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek und H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM'01, San Diego*, 2001.

[SNDW06] A. Singh, T.-W. Ngan, P. Druschel und D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proc. 25th IEEE International Conference on Computer Communications (INFOCOM 2006), Barcelona, Spain*, April 2006, S. 1–12.

[Staj02] Frank Stajano. Security for Whom?: The Shifting Security Assumptions of Pervasive Computing. In Mitsuhiro Okada, Benjamin C. Pierce, Andre Scedrov, Hideyuki Tokuda und Akinori Yonezawa (Hrsg.), *ISSS*, Band 2609 der *Lecture Notes in Computer Science*. Springer, 2002, S. 16–27.

[StAn99] F. Stajano und R. Anderson. The Resurrecting Duckling: security Issues for Ad-hoc Wireless Networks. In *Proceedings of the 7th International Workshop on Security Protocols, Cambridge, UK*, 1999.

[Stei08] M. Steiner. *Structures and Algorithms for Peer-to-Peer Cooperation.* Dissertation, Universtität Mannheim; École Nationale Supérieure des Télécommunications, 2008.

[StENB07] M. Steiner, T. En-Najjary und E.W. Biersack. Exploiting KAD: possible uses and misuses. *SIGCOMM Comput. Commun. Rev.* 37(5), 2007, S. 65–70.

[Stro] StrongS/WAN. http://www.strongswan.org.

[StWe05] R. Steinmetz und K. Wehrle (Hrsg.). *Peer-to-Peer Systems and Applications.* Springer. 2005.

[TaTS08] S. Tarkoma, D. Trossen und M. Särelä. Black Boxes: Making Ends Meet in Data Driven Networking. In *Proceedings of MobiArch'08*, Seattle, Washington, USA, 2008.

[TKLB07] Wesley W. Terpstra, Jussi Kangasharju, Christof Leng und Alejandro P. Buchmann. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. In *Proceedings of the 2007 ACM SIGCOMM Conference*, New York, NY, USA, August 2007. ACM Press, S. 49–60.

[WaSt98] Duncan J. Watts und Stephen H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 1998.

[WBHH+08] Oliver Waldhorst, Christian Blankenhorn, Dirk Haage, Ralph Holz, Gerald Koch, Boris Koldehofe, Fleming Lampi, Christoph Mayer und Sebastian Mies. Spontaneous Virtual Networks: On the Road towards the Internet's Next Generation. *it – Information Technology Special Issue on Next Generation Internet* 50(6), Dezember 2008, S. 367–375.

[Wölf05] T. Wölfl. Public-Key-Infrastructure based on a Peer-to-Peer network. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS38)*, 2005, S. 200a.

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons und Abraham Flaxman. SybilGuard: Defending Against Sybil Attacks via Social Networks. In *Proc. of ACM SIGCOMM 2006, September 11-15, 2006, Pisa, Italy*, 2006.

[YlLo06] T. Ylonen und C. Lonvick. The Secure Shell (SSH) Protocol Architecture (RFC 4251). *Network Working Group. The Internet Society*, 2006.

[Zimm07] Phil Zimmermann. Zfone (homepage). http://zfoneproject.com, 2007.

ISBN 3-937201-13-0

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)