Technische Universität München
Lehrstuhl für Kommunikationsnetze

Locator/Identifier Split Based Internet Architecture With Integrated Security & Privacy

Dipl.-Ing. (Univ.) Oliver G. Hanka

Vollständiger Abdruck der von der Fakultät Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr.-Ing. Ulf Schlichtmann
Prüfer der Dissertation:	1.	Univ.-Prof. Dr.-Ing. Jörg Eberspächer (i.R.)
	2.	Univ.-Prof. Dr.-Ing. Klaus Diepold

Die Dissertation wurde am 20.02.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 15.10.2012 angenommen.

Locator/Identifier Split Based Internet Architecture With Integrated Security & Privacy

Dipl.-Ing. (Univ.) Oliver G. Hanka

20.02.2012

To Julia and my Parents

Abstract

Due to its enormous growth and success the current Internet architecture is facing a set of challenging aspects. These aspects mainly arise from the discrepancy between the original design goals for the architecture and today's Internet usage pattern. To overcome those problematic issues, the locator/identifier separation paradigm was introduced. This novel addressing scheme provides solutions to many of those challenging aspects. In this thesis a novel clean–slate Next Generation Internet architecture based on the locator/identifier separation paradigm is developed and analyzed. The HiiMap mapping system builds the core of this architecture and provides a trustful mapping between identifiers and locators. Furthermore, it serves as a distributed public key infrastructure based on threshold cryptography. This has the advantage that no single point of trust within the architecture is required. The security framework of the HiiMap architecture is completed by a smart card based client key management. A downside of the locator/identifier separation paradigm is its disclosure of an end point's location information. To overcome the location privacy issue, a proxy-based mechanism is integrated into the architecture. The architecture was partially implemented and deployed to the G-Lab experimental facility.

Zusammenfassung

In dieser Arbeit wird eine neuartige Internet-Architektur basierend auf der Trennung von Adressen und Bezeichnern (Locator/Identifier Split) entwickelt und untersucht. Den Kern der Architektur bildet das HiiMap–Mapping–System, welches eine vertrauensvolle Zuordnung zwischen Adressen und Bezeichnern bereitstellt. Des Weiteren dient das Mapping System als Public Key Infrastruktur unter Verwendung von Threshold–Kryptographie. Dies bietet den Vorteil, dass kein zentraler Vertrauenspunkt in der Architektur benötigt wird. Auf Benutzerseite wird das Schlüsselmanagement der HiiMap-Sicherheitsumgebung mit Hilfe von kryptographischen Chipkarten bewerkstelligt. Ein Nachteil der Locator/Identifier–Split–Adressierung besteht darin, dass der Aufenthaltsort eines Teilnehmers jederzeit offengelegt ist. Die Architektur wird daher durch einen Mechanismus komplettiert, der die ortsbezogene Privatsphäre von Teilnehmern gewährleistet. Die Architektur wurde teilweise im Experimentalnetz G-Lab implementiert.

Acknowledgements

After my last exam Prof. Dr. Jörg Eberspächer asked me whether I would be interested in joining his research team at the Institute for Communication Networks. He easily convinced me by offering the prospect to work on novel Internet architectures within a joint research project of several universities and industrial partners. During my four years at the institute and on the project I discovered the freedom and creativity involved in investigating novel ideas and concepts. I grew fond of the research work and never regretted to haven taken this step.
I am, therefore, deeply grateful to Prof. Dr. Jörg Eberspächer for having offered me this position, supervising my thesis and including me in the G-Lab research project. Thank you very much for being a great and helpful supervisor; giving me support and guidance while at the same time leaving me enough freedom to follow and pursue my own ideas.
I also want to thank Prof. Dr. Klaus Diepold, who already supervised my bachelor and diploma thesis, for agreeing to be the second examiner of my thesis.
Furthermore, I want to thank all my current and former colleagues at the Institutes for Communication Networks and Media Technology. Being part of the "LKN" was a great experience. It not only meant to have colleagues to discuss research related topics but also participating in after–work activities or spending tea breaks among friends. For being able to discuss my work with and giving me advice I want to thank Christian Merkle, Julian Lamberty, Dr. Gerald Kunzmann, Dr. Michael Eichhorn and Dr. Moritz Kiese. In particular I want to thank Martin Pfannenstein for his advises and for proofreading this thesis.
In addition to the colleagues at the institute I had the pleasure to get to know and to work with other colleagues from the G-Lab project. I always enjoyed the conferences, workshops and project meetings which provided the chance for interesting discussions on various research topics as well as get–togethers afterwards. I am particularly thankful to Hans Wippel, Dennis Schwerdel, Dr. Bernd Reuther and Prof. Dr. habil. Michael Menth for the collaboration and fruitful discussions. Furthermore, I want to thank the participants of the Doctoral Consortium at the 10th GENI Conference.
Another thanks goes to my graduate students. In particular the collaboration with Wolfgang Fritz and Thomas Szyrkowiec provided much help in writing this thesis.
I am much obliged to my parents who always supported me, were there whenever I needed them and enabled me to be where I am today. Without your support and encouragement this thesis would have never happened.
Last but not least, I want to thank my wonderful, loving and encouraging girlfriend Julia for all the time we spent together as well as the love and support she gave me. Thank you so much!

Munich, Germany, February 2012 Oliver Hanka

Contents

List of Figures

List of Tables

1. Introduction

Within the last thirty years the Internet revolutionized the way people communicate. The invention of the telephone, radio and computer set the stage for its world–wide success. It evolved from a means to connect a couple of basement–filling super computers to a global network of billions of nodes. The Internet is the foundation of many business and drastically changed the way we distribute information, collaborate and interact with each other.

This immense success, however, is the major problem of today's Internet architecture. The fundamentals were designed in the 70s and early 80s of the last century—first concepts and prototypes even reaching back into the 1960s. The first TCP specification, for example, dates back to the year 1974 (Request for Comments 675 [RFC0675]). Back then, the personal computer was still unknown and the intention of the Internet was to interconnect a couple of super computers located at bigger companies, government sites or universities.

This highly contrasts with the usage pattern of the Internet today. While the protocols mostly remained the same, we see billions of nodes—a growing percentage even being mobile. The architecture has to cope with a bandwidth several magnitudes higher than the one in the beginning of the Internet. Further, nodes nowadays connect wirelessly to the network and communication partners can not be trusted anymore as in the early days where everyone knew everyone within the community. While the first Internet applications were destined to share research documents or duplicate data for backup reasons in a quicker way, today we want to transfer huge multimedia files across the globe. Packets were meant to be delivered in best effort manner, contrasting with the demand for (almost) real-time traffic where a fraction of a second can mean profit or loss of billions of dollars.

The first serious scalability concerns were raised in the early 1990s. Due to the rapid growth of the Internet it became evident that the 4,294,967,296 addresses provided by the deployed IPv4 protocol might not be sufficient in the future. Besides the exponential growth rate, a simple and obvious indicator for the address space exhaustion was the world population. Already in 1987 it reached five billion people. Among other reasons, this led to the specification of IPv6 which was proposed in 1995 (Request for Comments 1883 [RFC1883]) and reworked in December 1998 (Request for Comments 2460 [RFC2460]). Yet, the migration to IPv6 is not completed up to today.

IP address exhaustion, however, is not the only problem of today's Internet architecture. In Chapter 2 several other challenging aspects are described after a brief overview of the original design goals and principals is given. Within the research community it became evident that novel concepts to either extend the existing architecture or design a completely new one is inevitable (e.g., [RFC4984, GAB09]).

In this thesis, a novel clean–slate Internet architecture is introduced. It is based on the

locator/identifier separation paradigm and has a strong focus on trust, security and privacy. The contributions of this thesis are as follows:

- The first contribution is a scalable mapping system which maps identifiers to locators. One of its key aspects is the partitioning of the mapping system into regions. By assuring that the complete mapping system is not under a single authoritative domain, it increases its trustworthiness. The mapping system is DHT based, guaranteeing a reliable and highly extendable architecture which is able to cope with billions of entries.
- As security plays a major role in any Next Generation Internet (NGI) architecture, a security framework is introduced which is integrated into the mapping system. It secures the mapping protocol and provides a trust anchor for higher layer security protocols and applications. It is based on asymmetric threshold cryptography and is not dependent on a single point of trust. Furthermore, the framework provides a user–friendly client key management which is based on cryptographic smart cards.
- The locator/identifier split addressing scheme discloses location information about end systems to other peers. This is a problematic issue in regard to location privacy. The NGI architecture introduced in this thesis is extended by a privacy service which allows mobile nodes to conceal their location information. This service is designed to affect the communication flow between two peers as little as possible.

All of the architectural elements introduced in this thesis were evaluated in regard to their performance, security requirements or financial aspects. Furthermore, each element was prototypically implemented and their overall feasibility tested. The prototypes were deployed to the G-Lab experimental facility which provides a large scale testbed to evaluate Internet architecture concepts.

2. Motivation for a Novel Internet Architecture

Within the research community it is widely agreed that the Internet is facing certain problems and challenges (e.g.,, [Han06, GAB09, RFC4984]). These problems mainly originate in the contrast between the design goals of the Internet and today's usage pattern. Many researchers, therefore, believe that a change to the current architecture is unavoidable.
To motivate the research towards a novel Internet architecture, the challenges of the current one are outlined in this chapter after briefly discussing the original design goals. Resulting from these challenges a set of expectations for a novel architecture is defined, building the foundation of the Next Generation Internet (NGI) architecture introduced in this thesis.

2.1. Shortcomings of Today's Internet Architecture

The Internet as we know it has existed for roughly thirty years. While the architecture mostly remained the same, the way we use the Internet drastically changed in this time. This resulted in a set of problematic issues we encounter with the current architecture. To better understand the challenges, the original design principals of the Internet are briefly reviewed and, afterwards, problematic issues in the area of *scalability*, *mobility* and *security* are discussed.

2.1.1. Design Principals of the Current Architecture

In the early 1970s, researchers tried to find ways to interconnect existing local area networks. It soon became clear that a unified protocol suit was required which was supported by all sites in order to allow for communication across several independent networks. This is how the TCP/IP protocol suit (short for Transmission Control Protocol/Internet Protocol) came into being. This protocol suit is still the foundation of today's Internet and has remained unchanged at its core since its flag–day deployment to the Internet in 1983 [Cla88].
Feldmann ([Fel07]) and Clark ([Cla88]) list the following design goals which protocol designers were required to match while defining the protocol suit:

- **Connect existing networks:** Various local area or specific purpose (e.g., satellite communication) networks existed. The main goal was to interconnect all of these networks and enable nodes from different independent networks to exchange data.
- **Survivability:** Another key aspect was to design the network in a way that it would remain functional even in case parts of it failed for whatever reasons. Of

course, the failed parts would not be able to communicate anymore but the rest of the network should not be affected by this.

- **Support of multiple types of services:** Contrary to the then existing telephone network, the Internet should not be purpose–built for a single application (e.g., voice communication). The architecture should be open in a way that different services could co-exist and be used in parallel over the same network.

- **Accommodate a variety of physical networks:** The local area networks which the Internet should interconnect were built based on different technologies and protocols. This, however, should not hinder end–to–end connectivity between nodes of different physical networks. For example, a node attached via modem over a telephone line should be able to exchange data with a node equipped with a packet radio interface.

- **Allow distributed management:** As the various local networks were operated and funded by different authorities, the interconnection between two different networks should be realized by a mutual agreement between those two. The management authority for the local networks, therefore, was required to remain with its individual operators. Thus, the Internet should not rely on a single management entity but rather allow for distributed and independent management of autonomous networks.

- **Be cost effective:** While an interconnecting network was highly desired, the first participants consisted of research facilities mainly from the academic sector. Equally to today, funds were limited and in forecast to allow the Internet to grow, it needed to be designed as cost–effective as possible.

- **Allow host attachment with a low level of effort:** Attaching an additional node should be relatively easy and not require high expenses or a tremendous amount of programming/configuration. This goal was set to simplify the attachment of new sites and support the growth of the Internet.

Resulting from these design goals, the following design principles were derived which build the foundation of the TCP/IP protocol suit and, therefore, the architecture of today's Internet.

- **Layering:** The concept of layering was introduced to provide well–defined interfaces between functional components. The idea was to create functional black boxes which carry out a defined task. Other components relying on it do not require to know about the implementation specifics nor how the task is executed in detail. A layer designed to forward bits from one node to another, for example, needs to accept these from a higher layer and be able to hand over received ones from a peer node. The higher layer, however, does not need to know about the exact electrical encoding in which the bits are sent out on a copper wire or how light pulses are generated to use a fibre optical connection. This principal satisfies the design goals to support a variety of physical networks, the support

of multiple services and also touches the goal of cost effectiveness and ease of host attachment. It resulted in the creation of the Open Systems Interconnection model (OSI model) which defines seven layers from a physical one up to the application layer [ISO94]. The TCP/IP protocol suit uses a simplified four layer model which concentrates layer one and two of the OSI model into the link layer and number five to seven into a single application layer [RFC1122].

- **Packet switching:** This back then heavily debated design principle was about how data is delivered between two nodes—either by using a dedicated connection (like the telephone system) or by scrambling a data stream into packets which travel individually to their destination. The conclusion was to use the packet switching approach to satisfy the survivability and cost effectiveness design goals. A dedicated and pre–setup connection between two end nodes would be fatally effected by a network component failure on its path and the connection would be interrupted. By using individually routed packets, a small number of packets is also affected by a network failure. Subsequent packets, however, can be routed via a different route and still reach their destination. In terms of caused costs, a dedicated connection allocates a certain amount of bandwidth on a path even if no data is currently sent. The packet switching approach has the benefit that bandwidth is only required during the transmission of the packet. In the meantime. i.e., between two packets, the bandwidth can be utilized for packets of a different communication.

- **A network of collaborating networks:** This design principle resulted automatically from the historical and political background of the then existing networks. Individual networks, operated by different owners, already existed and the idea of the Internet sparked from connecting these networks. It, however, was not desired to bring the existing networks under a single authority which would have meant to sign over a respective amount of investment for the old owners. Thus, the design principal was set to interconnect the existing networks while leaving the operational control with the individual operators. The networks would only collaborate with each other and have mutual agreements with their neighbors to interconnect. A global routing scheme ensures the path finding between all networks even if they are not directly connected. This further supports the design goal of *survivability* and satisfies the goal to allow for a distributed management.

- **Intelligent end systems:** With each functionality placed within the network, it would become more and more expensive. This contradicts the design goal to be cost–effective. The principal, therefore, resulted in intelligent end systems which handle as much functionality as possible. Only functionality which is required by any end system is placed within the network (e.g., path finding and packet forwarding). Reliable packet forwarding, however, is not ensured by the network. This would increase the costs of the network even though not all applications benefit from it. Establishing reliable packet forwarding by retransmitting lost packets, for example, would not increase the service quality of voice

or video communication. The retransmitted packet might already been required prior to its arrival by the audio or video codec.

- **End–to–end argument:** The design principle following the end–to–end argument is tightly coupled with the principle of intelligent end systems. It states that functionality of a communication should range from end–to–end and is not supposed to be interrupted or reimplemented along the way. An example is TCP's flow control and error detection [RFC0675]. The TCP functionality is normally handled at the end nodes while the intermediate network elements are only required to deal with Internet Protocol (IP). It reduces the required functionality stack in the intermediate network components and, therefore, also satisfies the design goal of cost effectiveness. It further addresses the goal of survivability. By keeping complexity out of the network the probability of failure is reduced. Furthermore, a general implementation of a specific functionality provided to all nodes might not meet the exact requirements of a specific application and either introduces small errors to the application's operation or requires a reimplementation in the application itself [SRC84].

2.1.2. IPv4 Address Exhaustion

A problematic issue of today's Internet architecture is the exhaustion of the IPv4 address space. The IPv4 specification defines a fixed length of four octets ($32\,\mathrm{bit}$) for the source and destination addresses in the packet header [RFC0791]. This limits the address space to a theoretical maximum of $4,294,967,296$ unique addresses. Not all of these addresses, however, can be allocated for routing in the public Internet. Some addresses are reserved for special purposes such as private networks (~18 million addresses) or multicast addresses (~270 million addresses). Additionally, each subnetwork has an own network and broadcast address which limits the amount of allocatable addresses even further.
To the designers of the Internet protocol, the amount of available addresses seemed sufficient. They did not expect a very huge amount of participating nodes and considered the current architecture as an experimental network only [Cla88]. Contradicting with these design goals is the huge success of the Internet today and its wide–spread use. In February 2011, the Internet Assigned Numbers Authority (IANA) which overseas global IP address allocation assigned the address blocks out of its pool of unused blocks to the Regional Internet Registries (RIR) [Hus11]. This means that no more free address blocks are globally available. As soon as the RIR assign all of their addresses to providers and independent customers, all addresses of the IPv4 address space are taken and novel nodes cannot be connected to the Internet anymore without disconnecting old ones[1].
In the early 1990s, the rapid increase of assigned IP addresses raised the first concerns that the $32\,\mathrm{bit}$ address space might not be sufficient and one day be exhausted. Already in 1987 the world population outgrew the number of 5 billion people and it be-

[1] This is a somewhat simplified statement, as other means like, e.g.,, Network Address Translation allow multiple hosts to share a single public IP address.

came obvious that in the future people might hold multiple devices which they want to connect to the Internet. The development of a novel IP version, therefore, was fostered and resulted in the specification of IPv6. The first proposal dates back to the year 1995 and a reworked version was published in December 1998 [RFC1883, RFC2460]. Among other changes to the protocol header, the fixed length of the address field was increased to $128\,\mathrm{bit}$ (approx. $3.4 \cdot 10^{38}$ addresses).
However, up to today, the migration from IPv4 to IPv6 is not completed. IPv4 is still the dominantly used IP version although many devices already are capable of IPv6 and operate a dual network stack. Furthermore, mechanisms to use IPv6 as end–to–end communication protocol do exist (e.g., tunneling IPv6 packets through a IPv4 network). The transition mechanisms and reasons why the migration is not completed by now, however, are out of scope for this thesis.

2.1.3. Forwarding Information Base Growth

Each router in the so–called Default Free Zone (DFZ) of the Internet maintains a Routing Information Base (RIB). In this RIB all known prefixes and the according routes to the destination subnetworks are stored. This information is usually automatically learned from the router's neighbors or statically entered by the network administrator. Additionally to the RIB, most of the routers also hold a Forwarding Information Base (FIB). This table reflects the complete or parts of the RIB and is optimized for frequent read operations. Each incoming packet is matched against the FIB and forwarded according to its entries. In former days, only a subset of the RIB was cached in the FIB for fast access. Nowadays, however, router designer realized that too much delay is introduced to a router's forwarding plane in case an entry is not found in the FIB and, therefore, needs to be fetched from the slower RIB. As both information bases in up–to–date router designs hold basically the same information, they are no further distinguished in this thesis and only referred to as FIB.
The FIB marks another grave problematic issue with today's Internet architecture. More specifically, the more than linear growth of the FIBs in the DFZ is raising concerns [Hus11]. Today's routers in the core network have over $300,000$ entries in their FIB and need to parse it for each incoming IP packet[2]. Furthermore, the line rate of the routers' interfaces increases with each new generation which results in an even smaller timeframe in which the lookup in the FIB needs to be performed. Although the processing power of the router also increases, it can not cope with the computational demand caused by the table growth and increased packet throughput [RFC4984].
The research community has identified several reasons for the rapid growth of the FIB [RFC4984]. The most obvious reason is linked to the IPv4 address exhaustion described before. As more and more subnets are used, more entries in the FIB must be stored as long as no address aggregation can be used. The solution to the address exhaustion, IPv6, however, increases the FIB growths even further. IPv6 and its huge address space also allows for several magnitudes more subnets which again increases

[2]Packets forwarded based on *Multiprotocol Label Switching* cause no lookup in the FIB.

the amount of entries in the forwarding tables.
The more subtle reason for the FIB growth is address de–aggregation. The original thought was to aggregate smaller subnets in the FIB into fewer bigger ones. This means, instead of storing the same path for multiple /24, for example, only a single entry for the matching /8 subnetwork is stored. This shrinks the FIB size drastically and is a well–known mechanism from the telephone numbering plan [ITU05]. A call, for example, originated in Germany with the country code of the destination being +1 is always switched to North America, no matter what city the subsequent numbers indicate. The German telephone system only needs to store a single forwarding entry for the complete North American continent instead of an entry for each city. Analogous, /8 address blocks are assigned to the Regional Internet Registries (RIR) and they assign smaller blocks to the provider in their region. The providers then assign subnets to their clients. In that way, routes from other regions or providers can be aggregated into bigger address blocks. Figure 2.1 illustrates a simplified example. The 1.1.0.0/16 subnet is attached to Router 2 (R2) and two smaller (/24) subnets out of this address block are in use. Router 1 (R1), however, only needs a single FIB entry in order to forward a packet to either one of these subnets.

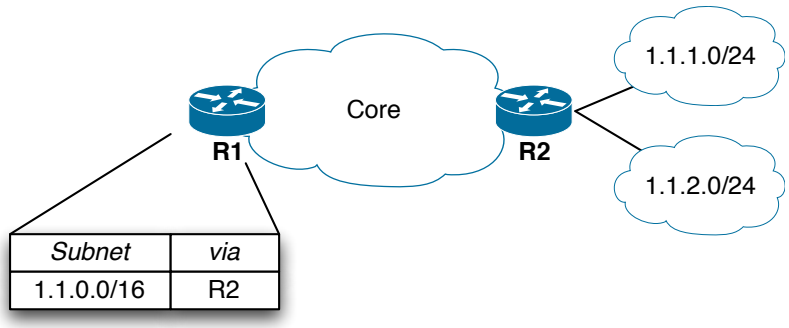


Figure 2.1.: Address aggregation.

This aggregation scheme, however, was broken due to multiple reasons. The three major ones are provider–independent addresses, multihoming and traffic engineering. Provider–independent addresses are subnets which the RIR assigns directly to end–user organizations. The end–user organization still needs a contract with a provider to obtain routing of the subnet addresses but is able to change the provider without having to assign different addresses to its nodes. This limits the provider change barrier for a company as the costs associated with the change due to reconfiguration are marginal. For the network, however, this means that a small address block cannot be aggregated with a bigger one anymore and requires a separate entry in the FIB of the routers in the DFZ. The example in Figure 2.2 shows that an additional entry in R1's FIB is required compared to the example in Figure 2.1
In case of multihoming an end–user organization has a subscription with more than one provider for fault tolerance reasons. The customer's subnet is connected via at least two uplinks with the rest of the Internet. Although one link might only be a

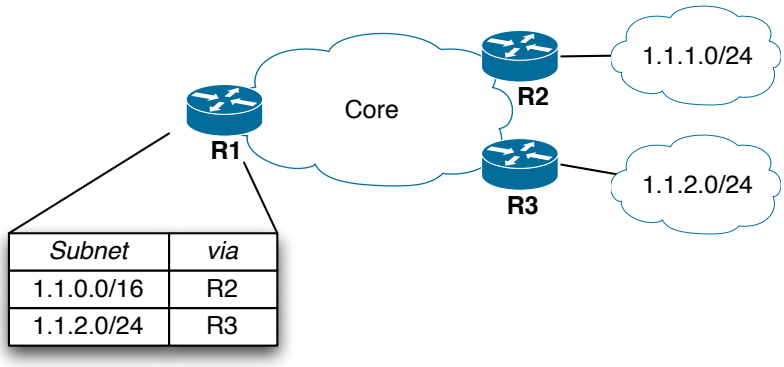


Figure 2.2.: FIB growth through provider independent addresses.

backup link, a route via this link must be propagated to other routers in order to be able to switch to the backup link in case of a failure of the main link. In case the second link is only active for backup reasons, the FIB does not necessarily need to grow—the RIB, nevertheless, does. Some end–user organizations, however, use the additional links for load sharing. In this case the additional routes must be present in the router's FIB[3] as depicted in Figure 2.3.

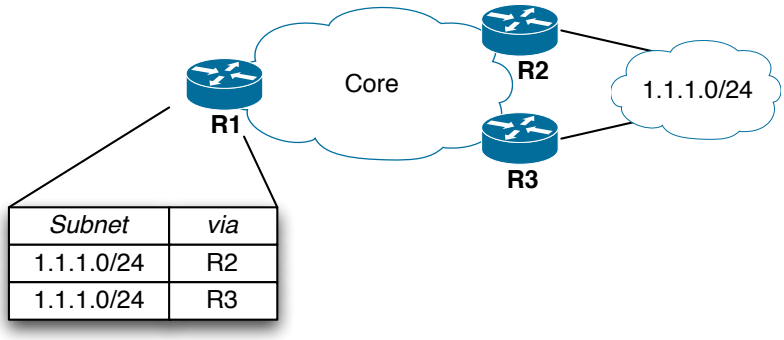


Figure 2.3.: FIB growth through multihoming.

Traffic engineering is another cause for route de–aggregation. As these mechanisms tend to be more complicated, only one example as shown in Figure 2.4 is given. The $1.1.0.0/16$ subnet is reachable via router 2 (R2) from a router 1 (R1) perspective. Between router 3 (R3) and router 4 (R4), however, a direct peering is maintained and it is agreed on that no transit fees are charged for any traffic passing this link. Traffic forwarded via R2 into the core of the network is subject to transit fees. The owner of the $2.2.0.0/16$ subnet, therefore, is interested in forwarding all traffic destined for the $1.1.2.0/24$ subnet via R3, and not R2. This causes route de–aggregation and an additional FIB entry for R1.

[3]Please note that it depends on the deployed routing protocol which route to store and select for forwarding.

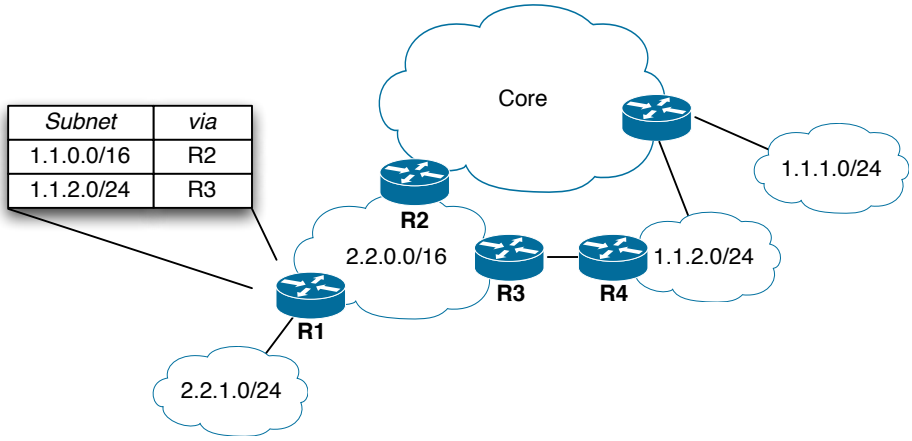


Figure 2.4.: FIB growth through traffic engineering.

2.1.4. Lack of Security and Privacy

Many security problems relate to the Internet's weak notions of identity and the ability to easily spoof everything ranging from IP addresses to routing updates and email addresses [RD10]. The Internet was originally designed to be an open network with mutual trust relationships between participants. It was intended to have no centralized authority which controls identities or serves as a guard whether an aspirant is allowed to join the network or not. This is why no natural trust anchor exists to build, e.g.,, integrity, authenticity, non–repudiation and privacy upon. Rather than supporting security mechanisms, the open architecture is by design diametrically opposed to a well–controlled system which would be ideal to provide security functionality [SGP+07].
The non existing trust and security concept was not an issue during the Internet's design phase. This is the reason why none of the design goals and principles described in Section 2.1.1 cover these aspects. This, however, changed drastically as soon as the Internet faced an exponential growth rate and became more and more economically relevant. Mechanisms were required in order to meet the emerging needs for secure and trustful communications. The open and decentralized collaboration of networks forming the Internet, however, does not support any kind of security or trustful links across multiple operators.
Until today, the research community and security specialists are debating which core security principles should be applied to the Internet's architecture in order to compensate for the initial lack. The principles taken are analogues from the information security field and extended to network security. There is, however, no real consensus about the specific principles included. In the following, only aspects are discussed which are included in most publications and listed in [Eck06].

- **Authentication:** In communication systems it is desired to be able to identify a person, legal entity or end–point. Access rights might be bound to a specific identity, and the subject claiming to own this identity needs to be able to proof

it. Furthermore, authentication can be applied in the context of transactions or information in order to prove that they are genuine.

- **Integrity:** To protect the integrity of any information means that it cannot be modified undetectably. Data exchanged between communication partners is required to arrive unchanged at the destination in order to be considered integer.

- **Confidentiality:** This principle covers the aspect of concealing information against third parties. Information should only be accessible to authorized subjects, and no unauthorized onlooker or intermediate forwarder should be able to obtain full or parts of the transmitted information.

- **Availability:** In communication systems it is important that information is available and can be accessed by authorized subjects. Third parties should not be able to prevent access to information by, e.g.,, interrupting communication channels or concealing end–systems from the rest of the network. The difficulty is to distinguish between security–relevant threads to availability or legit incidents which limit it (e.g., congestion).

- **Non–repudiation:** A system is providing non–repudiation in case actions cannot be denied afterwards by the subject initiating it. This means that a communication partner is not able to claim to neither have transmitted nor received a transaction even though it was involved. This principle is especially important in eCommerce systems and for accountability.

- **Privacy:** In order to provide privacy, any information which gives away clues about a natural or legal person must be concealed. Thereby, two levels of privacy can be defined. The first is to mask person–specific information to onlookers not involved in a communication. The second is to conceal the identity even against communication partners. The challenge, hereby, is to still allow for mutual authentication.

To address the initial lack of security and to fulfill the above mentioned principles in the Internet, novel protocols (e.g., IPsec, HTTPS, DNSSEC) were designed and purpose built to secure a specific application. This, nevertheless, has not resulted in a secure Internet. Combining secure components does not necessarily lead to an overall secure system. The weakest component still determines the security level of the complete architecture [Fel07].
Usability is another problematic issue regarding security. Mechanisms are often too complex for end–users to fully understand and utilize. This is worsened by having individual approaches by different applications. The average user is often not aware of the risks imposed by the lack of security. It is, therefore, common that the extra effort required to establish secure communications is not taken.

2.2. Expectations Towards a Novel Architecture

There are many expectations towards a NGI architecture which widely differ depending on the point–of–view of the observer/stakeholder. The European Future Internet Assembly conference compiled a general list of aspects, properties and desired mechanisms they expect from a NGI architecture [GAB09]. Addressing all of these properties would certainly go beyond the scope of this thesis. The following properties are only an excerpt and represent pressing challenges which are dealt with in this thesis. This, however, does not mean that other aspects (e.g., cost–effective network management, green–IT) are less important. Those properties not addressed were always kept in mind while designing the proposed components in order to not foreclose them by design.

2.2.1. Scalability

Scalability, besides security, is one of the most referred–to properties expected from a NGI architecture. The term, however, is a bit ambiguous and often used to abstract different characteristics. In the context of this thesis, scalability refers to an architecture's ability to support a large amount of participants while not disproportionately increasing complexity. In Section 2.1.2 and 2.1.3 two scalability problems of today's Internet architecture are discussed which need to be addressed by a Next Generation Internet architecture.
The first challenge is the address space exhaustion. A Next Generation Internet architecture requires an addressing scheme which either does not limit the amount of available addresses or provides a very large address space. The size of the address space, however, must not only satisfy projected subscriber growth based on today's growth rates. It even needs to provide enough addresses for novel, yet unforeseen usage patterns. This could, for example, be a novel class of Internet devices boosting the amount of required addresses. Another scenario could be to not only address end–systems but also services, persons and content in the future. A novel addressing scheme, therefore, must be capable to support an enormous amount of end–points[4].
The second challenge refers to the computational load of the intermediate network elements. As discussed in Section 2.1.3, the FIB in the DFZ is growing at a much higher pace than the development in hardware technology can keep up with. The reasons for the FIB growth are mainly identified as the requirement for multihoming, provider–independent addresses and the vast subscriber growth [BGT04]. These factors or respectively the underlying causes are, however, desired aspects of a Next Generation Internet architecture. Their benefit should not be restricted in any way. Contrariwise, the novel architecture should provide native support for this functionality without the downsides of today's mechanisms.
While IPv6 is a solution for the first challenge, it does not address the second one at all. It even worsens the FIB growth by supporting many more end–points. Fur-

[4]The term *end–point*, thereby, means any kind of addressable entity participating in or being affected by the Internet.

thermore, the very large address space inveigles the lavishness of subnets amplifying de–aggregation.

A Next Generation Internet architecture, therefore, must provide an addressing scheme which features a very large address space but at the same time addresses the route de–aggregation problem. Furthermore, the original design goals of the Internet: *cost effectiveness*, *host attachment with a low level of effort* and *survivability* must still be satisfied.

2.2.2. Mobility

Mobility of content, services and end systems is another important challenge a NGI architecture has to support. During the initial design of the Internet, mobility was not an issue at all. Network enabled end systems were large and heavy, and devices with a mobile power supply were nearly non–existent. This drastically changed during the last 20 years. Nowadays, lots of Internet–capable mobile devices exist and mobility becomes more and more important.

On layer 2 of the OSI model, mobility is already addressed. Most Media Access Control (MAC) layer protocols are capable of roaming and support seamless handovers between base stations, for example. Modern devices, such as smart phones, also support virtual seamless handovers between different access technologies (e.g., 802.11 vs. UMTS). This is achieved by maintaining connections over both access technologies and forwarding data over the faster one (e.g., 802.11). Whenever one access link becomes unavailable, the device forwards data on the remaining link, still providing Internet connectivity without the requirement of user interaction.

Layer 3 mobility, however, is not supported by either the original design of the Internet nor today's devices. Each change of network attachment point which is beyond the scope of the local subnet results in the assignment of a novel IP address (either statically or dynamically). This means that all connections based on the old IP address are interrupted and need to be re–established using the new address. The architecture does not support address changes for connections (or connectionless streams for that matter), and functionality within the application layer must compensate for this lack. Although mobility support for layer 3 has been specified to address this (MobileIP [RFC5944]), it has not seen wide acceptance. A NGI architecture, therefore, must support layer 3 mobility without the disadvantages of MobileIP[5].

In a future Internet, mobility, however, might not only be restricted to end–systems. It is also feasible that virtual components and services migrate from device to device while requiring a steady connection. An example would be a conference call service which is executed on a sever as close as possible to the participants. In case participants join and leave the conference during the call, it might be useful to migrate the service to another physical machine. This kind of mobility must also be supported by a NGI architecture.

[5]One of the main disadvantages of MobileIP is *triangular routing* which is discussed in Chapter 5.4.

2.2.3. Security

Security must be a key aspect and play a fundamental role in any NGI design. The challenge is to find a good balance between an open architecture and a closed system [SGP+07]. While the open architecture reflects the current Internet design, a closed system is the ideal foundation for a strong security architecture. Please note that *closed system* in this context does not mean proprietary mechanisms or protocols—they can be open–source ones. It rather refers to accessibility of the network and strong control mechanisms to grant or deny functionality within the network.
In [Han06] Handley argues that it will not be possible to find a single magic bullet to fix all of the Internet's security issues. Nor is it possible to foresee any future vulnerability. He states that security will always remain a race between attackers and those trying to prevent the attacks. It is, therefore, impossible to design a security framework which is able to withstand all of today's and any future attack scenarios. Rather than trying the impossible, the efforts should be put into designing a NGI architecture which provides a strong foundation to build security mechanisms upon. Contrary to a closed system, the architecture should remain open and extendable. An extendable architecture can be adapted and extended with novel security mechanisms to counter future vulnerabilities. It also does not contradict the original design goals of the Internet.
Another important aspect of security is trust. This holds true for the security mechanisms itself but also the basis on which the mechanisms are build upon. In case one does not trust the starting point of a security chain, the rest of the chain becomes untrustworthy as well even if the links are sound and meet one's security requirements. The design goal in terms of security for a NGI architecture, therefore, must be to implement a strong trust anchor. This trust anchor must enable security mechanisms to provide the aspects described in Chapter 2.1.4 (i.e.„ *authentication, integrity, confidentiality, availability, non–repudiation, privacy*). In order to introduce trust, the anchor must not be controlled by a single authority. This contradicts the design principle of distributed management, and no political, commercial or non–government organization is unreservedly trusted by any Internet participant.

2.2.4. Privacy

Similar to security, privacy was not on the agenda while designing today's Internet. While being a subset of the security field, the aspect of privacy only raised more attention in recent years. Especially social networks and companies collecting huge amounts of personalized data brought privacy to the attention of the public [Coo11].
A user's privacy is most likely violated by unauthorized collecting and combining of personalized information of any kind. This, for example, could be data about one's habits, work related information, social relations or movement profiles. The information leakage, thereby, can be intentional or unknowing and occur on all layers of the OSI model. In recent privacy discussions, it is frequently stated that users often voluntarily give away personal information. It, therefore, is unnecessary to research or implement mechanisms to protect a user's privacy [Coo11]. This can be outlined

as the social network paradox. While users demand data confidentiality and privacy, they voluntarily give away personal information without any hesitation.
The argument, however, neglects an important aspect. A user (or in this context more suitable: *person*) should have the right to remain in control over which information is made public and which not. Information should only be disclosed in case the user is aware of it and intentionally does so. Gathering information without the user's knowledge should be prevented by the network architecture and the applications.
Considering location privacy, for instance: Gaining knowledge over a user's current whereabouts could be valuable for advertising reasons. Taking advantage of the location information should be possible in case the user decides to reveal his Global Positioning System (GPS) coordinates for whatever reason. It, however, should not be possible to learn the user's coordinates by other means, e.g., layer 3 address to coordinate matching.

2.3. Next Generation Internet Architecture

The challenging aspects the Internet faces serve as motivation for a NGI architecture. Many researchers agree that a change to the current architecture is necessary to be able to address the challenges and meet the expectations [RFC4984]. How these changes might look like, however, is not yet clear. Two approaches for a NGI architecture design are discussed which are outlined in the following. This chapter ends with an overview of the NGI architecture presented in this thesis.

2.3.1. Evolutionary vs. Clean–slate Approach

The research community is vividly debating whether it is necessary to address the challenges listed in Section 2.1 by a completely new design or whether it is sufficient to continually try to improve the current architecture. The *evolutionary* approach is an evolvement of today's Internet with incremental changes to the architecture to meet the challenges. The so called *clean–slate* approach, in contrast, suggests to design a novel architecture from scratch without the requirement to provide backward compatibility.

Evolutionary: The evolutionary approach aims at understanding the problematic issues and its impacts on the current Internet. Afterwards, a design is proposed which resolves the challenges with the constraints of backward compatibility and incremental deployment [RD10].
Advocates for this approach argue that there is no history of any large–scale deployment of a clean–slate architecture so far. Revolutionary ideas have been well–researched (e.g., active networks, CLNP, Nimrod). But even less radical and backward compatible protocols which are not incrementally deployable (e.g., IPv6, S-BGP, RSVP) have not been in large–scale productive use. Evolutionary elements (e.g., NAT, CDN, DiffServ), in contrast, are widely deployed and became an integral part of today's architecture. Dovrolis states in [RD10] that economics forbids to replace even

an inferior technology unless the additional benefits of the new protocol significantly outgrow the high transition costs. The last fundamental change of the core protocols date back to the year 1993 when Classless Inter Domain Routing (CIDR) was introduced [Han06]. This change was only possible because of the pressing challenge of class B subnet shortage. To realize this introduction, changes to end–systems were postponed to the natural operating system upgrade cycle, and the majority of the routers were from a single vendor (Cisco). The packet forwarding was largely realized in software and could easily be changed by installing a new firmware. This is in high contrast to the massive hardware acceleration implementations in today's routers manufactured by various vendors.
The last *flag–day* event—representing a clean–slate like protocol deployment—was on the 1st of January 1983 when the transition to the TCP/IP protocol suit was carried out [RFC0801]. Even back then the transition lasted three days although it was planned to complete the switch within a single day. Attempting such a switch today would result in a significant financial loss in regard to the Internet's fundamental role in today's economics.
Furthermore, Dovrolis argues that the current architecture is "debugged" for over 30 years now. Even a very carefully designed revolutionary architecture will suffer from software bugs and design flaws due to its complexity. A clean–slate architecture, therefore, will not be better than today's architecture with incremental improvement applied to it.

Clean slate: The goal of the clean slate approach is to design an architecture which is significantly better in regard to performance, security, scalability and other properties than the current Internet. This architecture should not be limited by constraints of today's Internet and the requirement to ensure backward compatibility.
While advocates of the evolutionary approach consider add–ons and enhancements to be part of the solution, those favoring a clean slate approach see the incremental fixes as part of the problem of today's architecture. They argue that the old, fixed structures limit the development of novel principles and a flexible architecture. Furthermore, the constant bug fixing and enhancing of the architecture led to a complex patchwork of protocols and add–ons. Not uncommon, some add-ons designed to solve a problem conflict with another protocol introduced to deal with another limitation of the architecture (e.g., NAT vs. IPSec [Han06]).
Another problematic issue is the limited scope of some incremental enhancements as described in [Han06]. In the mid-1980s, for example, the Internet suffered from a series of congestion collapses. The network was busy with forwarding a lot of packets but a high percentage were retransmissions. The reason was that original packets did not come through or were acknowledged after the retransmission timer expired. Congestion is a network level problem and should be addressed at this level. It, however, was not considered to be feasible to implement a novel layer at all network elements and participants in a flag–day like manner, and an incremental solution was desired. The bug–fix was to deploy a TCP congestion control mechanism which was backwards compatible. This transport–layer solution limits the congestions caused by TCP

flows but does not address any other protocols also capable of provoking congestion. It, furthermore, does not factor in traffic from other protocols into its algorithm.
So–called *ossification* is another problem which evolved from incremental enhancements to the architecture. Some network elements (e.g., NATs and firewalls) were introduced to provide a certain desired functionality. These middle–boxes, however, do not fit into the layered IP architecture and operate at layer 4 or above. This violates the end–to–end design principle as these boxes intercept network traffic without being either source or intended destination of the packets they process [Han06]. These network elements are not transparent to higher layers. This means that the originally intended flexible and layered architecture is broken (or ossified), and novel protocols or applications require workarounds or additional bug–fixes to function properly. As HTTP is a common and widespread used protocol, many novel applications mimic HTTP traffic to communicate to be able to pass the middle–boxes. A clean slate approach could reintroduce a defined architecture which is flexible enough to provide the desired functionality of the middle–boxes without its downsides.
In [Fel07], Feldmann et al. outline that pressing challenges of today's architecture are rooted deeply in early design decisions underlying the Internet. Some of these design principles, however, became obsolete or even contradict with today's challenges and usage patterns (e.g., TCP slow start mechanism vs. wireless access networks). Sometimes even advances in technology may question some of the design principles. Rexford [RD10] and Handley [Han06] describe in their works that the current architecture was only designed to be a testbed and not intended for large scale productive use as seen today. Advocates of the clean–slate approach, therefore, favor this approach because they state that a novel architecture has the chance to be freshly designed, analytically and practically evaluated and redefined before being deployed. This is the way the ARPA net was build—no heavy constraints from previous architectures limited the design.

2.3.2. The HiiMap Next Generation Internet Architecture

In this thesis, the Hierarchical Internet Mapping Architecture (HiiMap)—a Next Generation Internet architecture—is introduced which was developed at the Technische Universität München. This architecture follows the clean–slate approach in order to present an ideal concept which reflects today's requirements and is flexible enough to be adapted to unforeseen future usage patterns. The authors are aware, however, that HiiMap will probably never replace today's architecture in a flag–day like manner. Rather, the idea behind the clean–slate approach is to research mechanisms and algorithms benefitting from the green–field design without the ballast of old constraints. Only environmental factors (e.g., scalability, technology advances and usage patterns) are taken into account. In a first step, no backward compatibility to the current legacy architecture needs to be considered. After having defined, analyzed and prototypically verified the novel architecture, elements can be back ported to the current Internet architecture. The reason behind this approach is to set a goal in which direction the current architecture should evolve. Therefore, an ideal proposal needs to be defined

before further enhancing and pushing the current architecture in a specific direction. The design goals of the HiiMap architecture were to provide end–to–end connectivity between peers while overcoming the problematic issues of today's Internet architecture as described in Section 2.1. The main focus was to develop an architecture which addresses the challenges of scalability, mobility, security and privacy. While security and privacy can sometimes be seen as diametral views on the same aspect, they also inflict with the challenges of scalability and mobility. Whenever scalability is important—meaning a huge amount of users are involved—secure and trustful means of communication and data sharing are required. HiiMap provides underlying security mechanisms which are anchored in the architecture and can be employed by higher layers. As a constraint, the architecture needs to be trustful in order for participants to be able to beneficially utilize the security mechanisms. This was considered during the design of HiiMap. The aspect of mobility likewise raises security issues and also introduces the challenge of privacy. The HiiMap architecture, therefore, provides means to ensure a mobile user's location privacy.

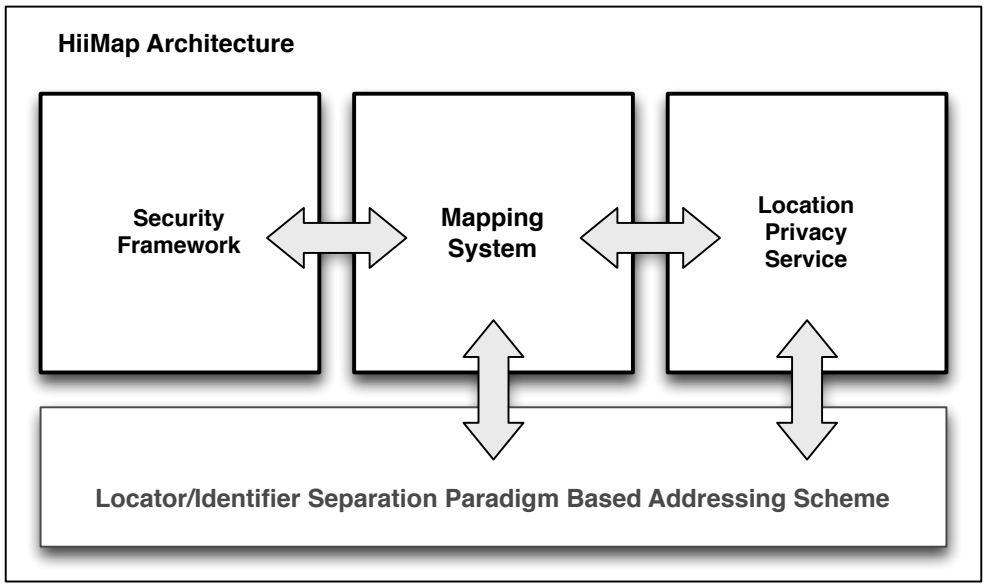


Figure 2.5.: HiiMap architecture overview.

Figure 2.5 depicts the overall HiiMap architecture. There are three core components which are introduced in this thesis. These components are the HiiMap mapping system (Chapter 3), the integrated security framework (Chapter 4) and the location privacy service (Chapter 5). Following the introduction of these components, details of the prototypical implementations used to verify the practicality of the components are given in Chapter 6.

3. Addressing and Mapping in a NGI architecture

The Hierarchical Internet Mapping Architecture (HiiMap) Next Generation Internet (NGI) architecture is a clean–slate based approach. It is possible to build it around a novel addressing scheme not backwards compatible with today's Internet Protocol (IP) architecture. Although elements of the current architecture are reused, novel concepts and ideas are introduced. In this chapter the novel addressing scheme—the locator/identifier separation paradigm—is described, and the benefits of this approach are outlined. This addressing scheme, however, requires a novel entity within the network, the so–called mapping system. This mapping system forms the core of the HiiMap architecture and is the major scientific contribution of this chapter.

3.1. State of the Art

In this section, the locator/identifier separation paradigm and its two different variations are explained. For each variation a related work concept is briefly described before evaluating the benefits and shortcomings of each approach. Subsequently, the requirements of a mapping system for any locator/identifier split implementation are outlined.

3.1.1. Locator/Identifier Separation Paradigm

To meet the requirements towards a Next Generation Internet architecture as described in Chapter 2.2, the locator/identifier separation paradigm has been proposed in the research community [FFML10, RFC4423], including the IETF and IRTF standardization and research bodies [RFC4984]. It was recognized that today's IP address assignment typically does not follow a topological but organizational structure, leading to the greater than linear growth rate of the Forwarding Information Base (FIB) in the Default Free Zone (DFZ). The reason for this is the overloading of IP address semantics.
In today's IP–based architecture (IPv4 or IPv6), the IP address has two semantical meanings. First, it serves as the address stating with *whom* we want to communicate and, second, *where* this person or node is attached to the network. This overload of meanings becomes problematic as soon as one of them changes. In case the attachment point to the network changes, reflected by a different IP address assigned to the node, the identity changes as well.
The locator/identifier separation paradigm, therefore, provides two disjunct addresses for each end point. An *identifier*, refecting the identity of the node and a

locator. The locator is used to determine a node's attachment point towards the network. This split between the naming and routing numbering space allows for a more topology–driven assignment of a node's routable address. As a benefit, a high level of aggregation is possible, thus shrinking the FIB in the DFZ drastically by reducing the number of globally announced prefixes. Additionally, the paradigm enables providers to handle multihoming elegantly by assigning multiple locators to a single identifier or identifier–space. Each attachment point toward the network can be addressed by an own locator while the node logically is only addressed by a single identifier. Furthermore, the locator/identifier separation paradigm allows for a more flexible traffic engineering by individually altering the identifier to locator mapping [QIdLB07].
An important aspect of the locator/identifier split is the idea behind *persistent identifiers*. This means that once an identifier is assigned to an end system, it ought not change other than specifically requested by the end system.
In the following, the two different variations of the locator/identifier separation paradigm are explained.

Transparent Approach

In the transparent approach, the end node is not aware of the locator/identifier separation paradigm. Like in today's IP architecture, the end-node uses only a single address throughout layer 3 and above to identify a destination. The semantical separation between the logical identifier and routable locator is carried out within the network. This means that the edge network is unaffected whilst the core of the network experiences the major benefit of the locator/identifier split approach.

Locator/ID Separation Protocol One example for a transparent approach is the Locator/ID Separation Protocol (LISP) by Farinacci et al. [FFML10]. In this concept, only a few nodes within the network are aware of the locator/identifier separation. The end nodes and the core network are not affected by this approach. LISP introduces so–called Endpoint Identifiers (EID) and Routing Locators (RLOC) which represent the identifier and locator. Ingress Tunnel Router (ITR) and Egress Tunnel Router (ETR) are involved in the locator/identifier split paradigm and keep the process hidden from all other network elements. LISP is an evolutionary proposal based on today's IP architecture and is compatible with both IPv4 and IPv6. Figure 3.1 illustrates the basic principle of LISP.
Whenever an end node wants to send a data packet to another end node not residing in its own local network, it uses the destination's EID as address. The EID is the destination's IP address assigned by its access provider. While the EID is routable in the source and destination network, so–called LISP sites, it is not within the core of the Internet. At the border of the LISP site, an ITR intercepts the packet and tunnels it to the ETR of the destination LISP site. This tunneling is done by encapsulating the original packet with an additional IP header. The destination address in this second IP header is the RLOC of the ETR. The RLOC for a specific EID is found based on EID prefixes which specify the LISP site which the EID node is attached to. A lookup in

a database system returns one or multiple RLOCs, and the ITR chooses one of these RLOCs as the tunnel endpoint. RLOCs themselves are normal IP addresses which are routed within the default free zone of the Internet. In case the destination end node is attached to the same LISP site as the source, the packet is forwarded to the destination solely on the EID.

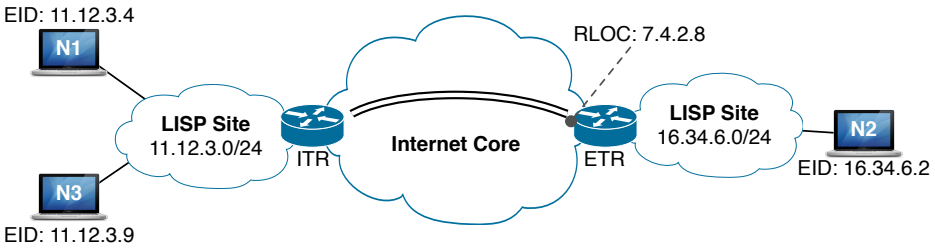


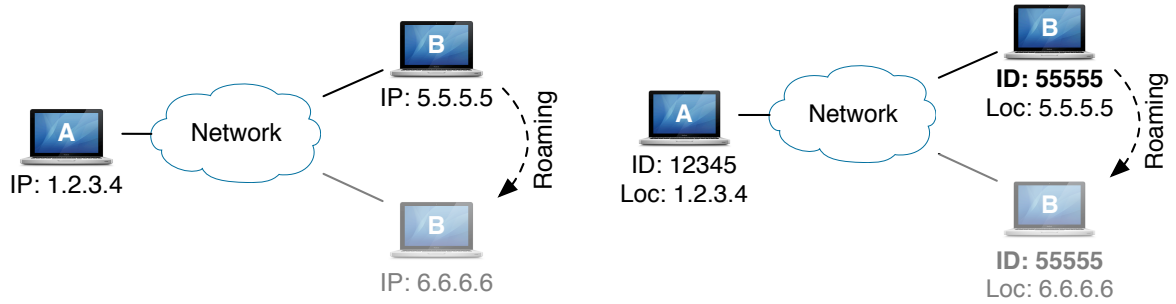
Figure 3.1.: Cisco LISP example.

In this example (Fig. 3.1), N1 wants to send a packet to N2. It addresses the packet to N2's EID and sends the packet to its default gateway which is the LISP site's ITR. The ITR checks the destination address and initiates a lookup for the LISP site with the network address `16.34.6.0/24`. This returns `7.4.2.8` as the RLOC. Subsequently, the ITR tunnels the original packet to the ETR with the corresponding RLOC by encapsulating it in a novel packet. The ETR strips off the outer packet and forwards the original one into its attached LISP site. Forwarded within the LISP site, the packet reaches its destination in the traditional way.

End Node Aware Approach

Contrary to the transparent approach, the end node aware one includes the nodes in the locator/identifier addressing scheme. Each end node holds one or more identifier and is assigned a locator for each network attachment point. Throughout the network only locators are routable, and the end node is responsible for the identifier to locator resolution. To obtain a locator for a specific identifier, the node needs to query a database in its reach. In this approach, the intelligence is pushed from the network to the edge and network components, e.g., router, do not need to handle identifier during packed forwarding. Figure 3.2 illustrates the difference between today's IP addressing scheme and the end node aware locator/identifier split approach. Notice how the identifier of node B in Figure 3.2b remains the same even when the network attachment point of that node changes.

Host Identity Protocol An example of end node aware architectures is the Host Identity Protocol (HIP) by Moskowitz et al. [RFC4423]. HIP introduces a novel namespace on top of today's IP address, called the Host Identity (HI). This HI fulfills the role of the identifier of the locator/identifier separation paradigm while the IP address is the locator. The HI represents a statistically globally unique name for any system with an IP stack. In [RFC4423], the authors state that "any name that can claim to be



(a) IPv4 (b) locator/identifier split

Figure 3.2.: Comparing the IP and locator/identifier split addressing schemes.

statistically globally unique may serve as a Host Identifier". They, however, strongly recommend to use a public key of a public/private key pair as the HI, thereby introducing a cryptographic namespace. The cryptographic aspect is discussed in Chapter 4 while the focus in this chapter is on the locator/identifier split aspect of HIP.
Different public/private key algorithms use different key length, therefore, HIP introduces a so–called Host Identity Tag (HIT). The HIT is a fixed 128 bit representation of the Host Identity. It is created by cryptographically hashing the corresponding HI. The HIT, therefore, is a 128 bit hash of the host's public key. The size of 128 bit was chosen to match the length of an IPv6 address. In a HIP packet, the HITs identify the sender and recipient of a packet and are meant to look like IPv6 addresses to higher layers.
Other proposals following this approach are [AAER06, KCC+07, FCM+09]. [AAER06] and [FCM+09] focus on addressing end nodes while [KCC+07] takes the concept of locator/identifier separation to the field of content addressing.

Comparison

While both approaches are designed to fix the most pressing downside of today's IP architecture—namely, the scalability issue in the core—, each has specific advantages and disadvantages. Within the DFZ, both approaches can be treated as equal in terms of countering the more than linear FIB growth rate due to aggressive aggregation of routable addresses.
The transparent approach requires no change at the edge of the network. End nodes are unaware of the locator/identifier separation paradigm. It, therefore, will be easier to migrate from today's architecture to such an approach, compared to the end node aware one. As a downside, however, this approach does not utilize the whole potential of the separation paradigm. To provide end node mobility from an address layer point of view, additional mechanisms like in today's architecture are required (e.g., MobileIP [RFC5944, RFC3775]). Furthermore, the tunnel routers, carrying out the separation need to be very powerful in terms of computational power and cache

memory. Each packet needs to be processed and the header manipulated in addition to the work of the forwarding plane. In case an identifier to locator mapping is not present in the router's cache, a query must be initiated to resolve the mapping. Until the response is received, the data packet must either be cached or discarded. At this point, no research has been carried out yet, whether the scalability problem is solved by this approach or simply shifted from the router in the DFZ to the tunnel router.
The end node aware approach has the benefit of native mobility support without any further mechanism. Additionally, it does not face the resource problem at the tunnel routers (which do not exist in this approach). The major challenge of this concept is the migration from today's IP architecture towards the locator/identifier addressing scheme. Each participating end node must be modified. The effort can be compared to the current ongoing transition from IPv4 to IPv6.
Tabel 3.1 summarizes the strengths and weaknesses of both locator/identifier split approaches. A "+" is used to mark that a specific feature is supported or a requirement is fulfilled. "-" indicates that the requirement or feature is not met/supported and "o" represents a partial coverage. This convention is used throughout the thesis.

	Transparent	End node aware
Counters FIB growth in DFZ	+	+
No additional network entity required	-	-
Inherent support for end node mobility	-	+
End node not required to be modified	+	-
Network entities not required to be modified	-	+
No add. packet dropping within the network	-	+
No migration efforts required	o	-

Table 3.1.: Comparison between transparent and end node aware locator/identifier split.

Although the transparent approach has benefits in regard to migration efforts, the end node aware one is chosen for the architecture introduced in this thesis. As discussed in Chapter 2.3, the architecture is a clean–slate approach. The migration advantage of the transparent locator/identifier separation paradigm, therefore, is not an important aspect. Mobility support and no additional packet loss within the network, however, are desired properties of a NGI architecture. These are inherently supported by the end node aware approach.

3.1.2. Mapping

A crucial aspect of any locator/identifier split architecture is the mapping between the identifier and locator plane. As described in Section 3.1.1, applications use identifiers to address the destination in the end node aware approach. Within the network, however, data is forwarded based on locators. A node's network stack, therefore, needs to be able to resolve an identifier to the destination's currently assigned locator. Before

discussing the required attributes for a mapping system in a future Internet scenario, selected existing mapping processes in today's Internet architecture are described.

Address Resolution Protocol (ARP) The Address Resolution Protocol is used for resolution of network layer into link layer addresses [RFC0826]. Although not specifically designed according to the OSI model, its functionality can be described as being positioned between layer 2 and 3. The ARP protocol can be employed with many different layer 2 and 3 protocols but is most commonly found in an Ethernet and IP scenario. In this scenario, ARP is used to map an IP address to a Media Access Control (MAC) address in a local area network. In case node A intends to send a packet to node B, it sends out a broadcast ARP message requesting an answer for node B's IP address. In this message node A's MAC address is included. Node B responds with its own MAC address and sends the message to node A, using A's MAC address learned from the broadcast message. In this way, node A is aware of B's MAC address and able to send packets to node B. After the initial address resolution the learned MAC addresses are cached at each involved host, eliminating the further need for ARP messages[1].
The ARP protocol is a well–established mapping mechanism in local area networks. Due to its broadcast character, however, it does not scale in a global scope. ARP broadcasts are only forwarded by layer 2 entities and not copied by layer 3 devices (e.g., routers). A mapping mechanism for the locator/identifier separation paradigm requires to be not only locally contained. Sending broadcast beyond local network boundaries would clog the network with uncountable broadcast messages and, therefore, is not feasible.

Domain Name System (DNS) DNS is a hierarchically distributed mapping mechanism to resolve human readable addresses (i.e., domain names) into machine readable IP addresses [RFC1034, RFC1035]. DNS also serves other purposes which, however, do not add to its mapping character and, therefore, are omitted in this context.
DNS is organized in a tree–like way resulting in a highly structured domain namespace. The root of the domain space is `"."`. The next layer consists of so–called top level domains (TLDs), for example, `"com"`, `"org"` and `"net"`. The levels below, called subdomains, are not predefined and can be registered freely with the according TLD authority. The example domain `"example.com."`, thus, represents a valid hierarchical domain name under the `"com"` TLD. The root of the DNS tree is controlled by 13 logical root servers (A through M) [Ass11]. These root servers host a list of authorities responsible for the TLDs. Each individual TLD authority controls its part of the DNS tree branch. It is, thereby, up to the authority to either resolve a domain by itself or further delegate the resolution of a specific subdomain to another authority. The latter is almost always the case, further extending the tree of domain name servers and authorities. In a next step, the delegate for the `"example.com."` domain, for example, might choose to register two additional subdomains under its own (e.g., `"munich.example.com."` and `"frankfurt.example.com."`). The resolution of

[1] Cache table aging, however, may result in novel ARP requests.

these novel subdomains might also be delegated to another authority. Figure 3.3 illustrates a sample DNS tree wherein the right branch reflects the `"example.com."` domain.

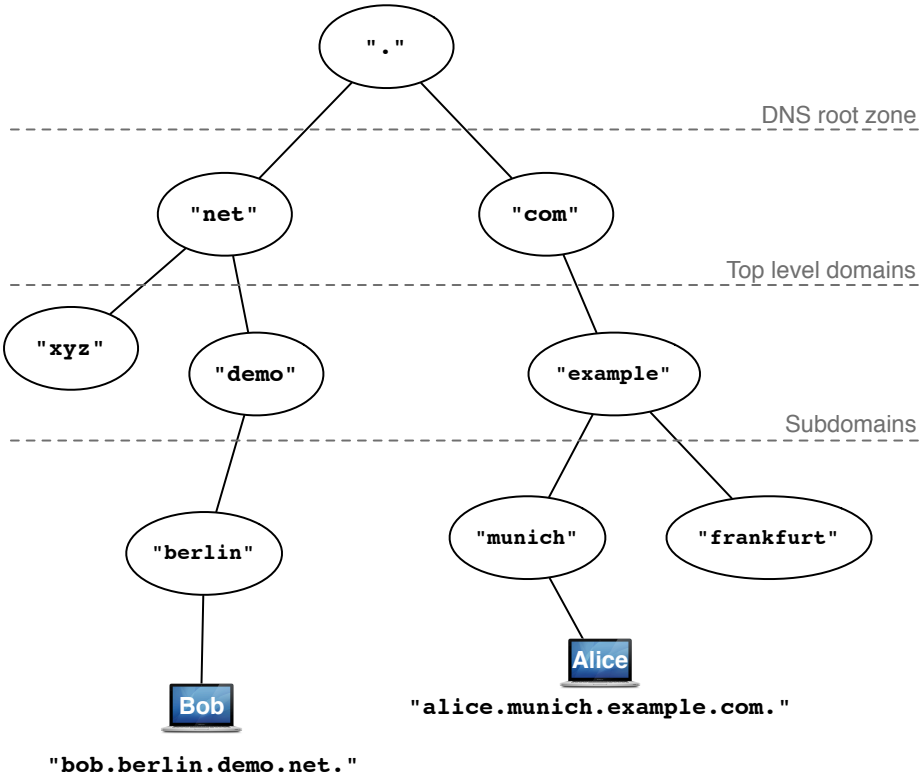


Figure 3.3.: Domain Name System tree.

A client requesting the current valid IP address for a specific domain name (e.g., `"alice.munich.example.com."` sends his request to a domain name server of his choice. This is most likely a server operated by his service provider. In case the domain name server is responsible for the queried subdomain, it looks up the corresponding IP address and responds to the request. If the subdomain is not within the domain name server's responsibilities, it forwards the request up the tree to its superior domain name server. Assuming that the client and his provider domain name server reside under the `"net"` TLD, the request gets forwarded to one of the root domain servers. These report back the IP address of the corresponding sever responsible for the `"com"` TLD. Once the request reaches the `"com"` domain name server, the subdomain `"example"` is looked up and the delegate for this subdomain determined. The request is forwarded to the delegate and, thus, travels down the branch of the DNS tree until the responsible domain name server for the `"munich.example.com."` domain is found. This server then is able to resolve the domain name into an IP address. Please note that the process of finding the responsible domain name server is sped up by the heavy use of caching in the domain name servers. The provider's domain server might directly contact the `"example.com."`

server in case neither itself nor one of its superior servers control part of the namespace of the requested domain.
Contrary to ARP, DNS is a global mapping mechanism. Thus, its infrastructure could be used to map identifiers to locators. Some researchers even proposed extensions to DNS in order to use it as a mapping system [ABH08, JCAC+10]. Their proposals include to use domain names as identifiers. For example, "alice.munich.example.com." would be the identifier of Alice and the domain name system would resolve the identifier to Alice's actual IP address. There are, however, two properties of DNS which limit its usefulness as a mapping system.

- The first problematic issue is the strictly hierarchical nature of domain names. In order to be able to traverse the DNS tree, each branch is unique and shares nothing with its neighbors other than the predecessor part of the domain. This is to ensure the global uniqueness of each full domain name. This, however, is a problem in case an end system wants to change its position in the DNS tree. Imagine this simplified example (cf. Fig. 3.3). Alice lives in Munich and her domain name is "alice.munich.example.com.". Bob is a friend of Alice and addresses packets always to her domain name (i.e., her identifier). As a benefit of the locator/identifier separation paradigm, Bob is not required to know Alice's exact whereabouts in Munich as the domain name system takes care of it during the domain name resolution. Whenever Alice roams within Munich, she notifies the domain name system about her new network attachment point and, thereby, her new IP address. The problem occurs when Alice decides to permanently relocate to Frankfurt. Because of the city change her identifier changes to "alice.frankfurt.example.com.". After the relocation Alice is updating her IP address with the domain name server of the "frankfurt.example.com." subdomain. In case Bob sends a packet to Alice's old identifier, the domain name server in Munich is not able to resolve the mapping any more. In order to be able to send messages to Alice, Bob needs to learn Alice's new identifier.

  This, however, heavily contradicts the locator/identifier separation paradigm. Over a node's lifetime, the identifier is expected to never change other than the node's specific request to obtain a novel identifier. This means, a node's identity should not change. Ignoring this principle obsoletes the whole separation between the identity and routing plane introduced by the locator/identifier split. A solution to this problem without breaking the DNS tree structure would be that Alice keeps her old identifier even in case she relocates to Frankfurt. This would require the domain name server of the "munich.example.com." domain to still resolve her mapping even though she is not a citizen (i.e., customer) anymore. In that way a provider would gain a lot of power over its customers. Once signed with one provider, it could charge huge fees for maintaining a mapping entry in case the customer chooses to sign with another provider. Another solution would be to deploy a global database which is aware of which domain name can be resolved at which provider (e.g., "alice.munich.example.com." at "frankurt.example.com."). This feasible solution, however, ob-

soletes the tree structure of DNS. There is no need to time–consumingly traverse a tree while holding a direct pointer to the leaf of it.

- The time consuming lookup in the DNS tree structure is the second problematic issue. Traversing a branch down from the root takes a notable amount of time. This linearly increases with each new level introduced to the tree structure. In order for DNS to scale a lot of new levels are required. This is because each participant of the Internet requires his own entry within the domain name system. DNS already faces the delay problem for each lookup today. There is, however, a solution to it for the domain name use case. This solution is caching which means that domain name servers learn shortcuts between branches. A single query, therefore, is not required to traverse the whole tree in case a cached entry is found. Furthermore, domain name servers also hold cache entries for previously resolved queries. This means, that Bob's provider in the `"berlin.demo.net."` branch already knows about Alice's IP address without being required to query the `"frankfurt.example.com."` domain server. This solution, however, is only feasible for very static entries. Caching only makes sense in case the cached information is valid for a certain amount of time. Quickly outdating information retrieved from a cache is of limited use as it might not be valid anymore upon retrieval. Domain name to IP address mappings are expected to change very rarely. Locator updates in a locator/identifier split architecture, however, occur whenever a node roams. A node traveling in a car or train might face many roaming events in which it changes its attachment point towards the network in a short amount of time. With each novel attachment point, a new locator might be assigned to the node, resulting in a locator update in the domain name system. The locator information for each identifier, therefore, needs to be considered as quickly outdating information due to mobility. This means that caching in DNS must be disabled when used as a mapping system for a locator/identifier split addressing scheme. As a result, the tree structure of DNS causes a very huge lookup delay because caching cannot be used. It is even further worsened by the requirement to provide a large degree of levels in the tree. DNS used as a mapping system is required to handle billions of entries, and only additional levels and branches in the tree are possibilities to reduce the load of each domain. The lookup performance of DNS with caching disabled is evaluated in Section 3.3.3.

3.1.3. Properties of a Mapping System

A mapping system's primary task is to store locator information for each registered identifier. It, therefore, needs some form of database entry for each identifier and must be able to accept both queries and updates for these entries. A mapping system can further be extended to fulfill additional purposes (e.g., security functionality, cf. Chap. 4). The following list, however, only considers required properties of a mapping system with regard to its elementary functionality.

- **Scalable:** Scalability is one of the major properties expected from a mapping system. This mainly results from the experience with today's architecture and the projected subscriber growth. Two aspects, thereby, determine whether a mapping system is scalable or not. The first one is the amount of entries a mapping system is able to store. The complexity of the mapping system, thereby, should not increase more than linearly in relation to the number of stored entries ($\leq \mathcal{O}(N)$). The size of the identifier address space forms the upper bound for the amount of identifiers to store. This number in a $128\,\mathrm{bit}$ address space, however, is probably hardly ever reached. The second scalability aspect is the mapping system's ability to cope with frequent updates of the mapping entries. Each layer 3 roaming event of an end system results in a novel locator which causes an update message sent to the mapping system. The challenge is to process these messages and be able to provide the novel information almost instantly for all subsequent queries regarding the updated mapping entry.

- **Fast:** This property is in regard to the caused lookup delay. Each query and update takes time to be processed by the mapping system. For instance, a database must be searched to find the corresponding mapping entry for a specific query. Due to the amount of entries a mapping system has to handle, it is most likely that the mapping dataset is split between multiple logical and physical entities. This means that it is possible that a mapping request cannot be answered by the recipient of a query. The recipient must forward the query to another entity of the mapping system which either stores the mapping entry or is able to further forward the request towards the storage location. In order to provide a minimal lookup delay, the amount of message forwards should be kept to a bare minimum.

- **Trustful:** The complete architecture is dependent on the correct resolution of identifiers to locators. Users, therefore, must be able to trust the mapping system or need functionality to verify the response. Furthermore, in case the mapping system design is built around multiple independent authorities, these authorities should not be required to rely on each other. Dependency means that distrust between the involved authorities could break the functionality of the mapping system or malicious information could be exchanged premeditatedly between the parties.

- **Persistent identifier:** A design goal of the locator/identifier separation paradigm is that an identifier should not change other than requested by its owner. The mapping system must support this paradigm of persistent identifiers and not require a change of identifier in any case. This could, for instance, be that a customer decides to change provider subscriptions or the mapping system is required to be reorganized for whatever reason. The paradigm of persistent identifiers results from today's challenging issue of de–aggregation in the IPv4 architecture (cf. Sec. 2.1.3).

3.2. NGI Mapping Proposals

Due to the lack of proper mapping technologies for a locator/identifier split architecture, novel approaches have been proposed. These proposals can be categorized into two main groups. The group membership, thereby, is determined by the structure of the mapping system. The first group relies on a tree–based structure and the second group utilizes Distributed Hash Tables (DHTs).

3.2.1. Tree–Based Mapping

In order to cope with the scalability issue, tree–based structures have been proposed. A tree can easily be extended by another branch or level in case the existing resources are not able to handle the load. Two different flavors of tree–based proposals exist and are reflected in the address space of the identifier. The first is using a structured identifier representing the tree hierarchy whereas the second uses flat identifiers.
Using a structured identifier, the identifier consists of several segments, each segment representing a certain level of the mapping tree. The length of the identifier, thereby, can be of fixed or variable size and extended as needed. An example for a structured variable length identifier is DNS as described in Section 3.1.2. To resolve a mapping for a structured identifier, the tree is traversed from the root according to the segments of the identifier. The corresponding data record is stored at one of the leafs and can be returned to the enquirer[2] (cf. Fig. 3.3).
Various tree–based mapping system designs have been proposed. Variants with structured identifiers are, for example, [PJPSi10, ABH08, Jia06, JCAC+10]. [PJPSi10] uses today's DNS as a mapping resolver, and [ABH08] allows for a mix of mapping systems, one being DNS. [Jia06] proposes to group identifiers in realms which are highly hierarchically structured for security reasons. [JCAC+10] uses a DNS based mapping system to support the transparent locator/identifier split approach LISP.
Flat identifiers are not structured, and no information is given by the identifier in which branch the corresponding mapping entry is stored. The information about which mapping entry is stored in which tree must be propagated throughout the tree. A leaf storing a new entry sends some kind of REGISTER message to its parent. This message is processed and forwarded until it reaches the root of the tree. The root knows which entries are stored in which branch of the tree and is able to forward a query to the respective arm. Caching mechanisms, thereby, allow to store information from previous queries in different branches. This cached information can be used as shortcuts throughout the tree to take load off from the root. Furthermore, the mapping information for flat identifiers can be stored in multiple branches, contrary to the structured ones.
[KCC+07] proposes an architecture with flat identifiers. It is briefly outlined as an example flat identifier architecture in the following.

[2] Note that some architectures also store information on non–leaf nodes in the tree in case not all levels are specified in the identifier (e.g., DNS)

Example tree–based Mapping Architecture Koponen et al. proposed *A Data-Oriented (and Beyond) Network Architecture (DONA)* [KCC+07]. They use flat identifiers which are based around principals. A principal is associated with a public key, and each datum, service or other named entity is associated with a principal. The form $P : L$ denotes a valid identifier, where P represents the principal and L the datum or content governed by the principal.
The mapping between a $\langle P : L \rangle$–tuple and the current associated locator is done by so–called resolution handlers (RH). RHs are organized in a tree structure and reflect the Autonomous System (AS) topology of the network. Each AS operates its own RH and the RH of a higher tier provider is the superior to a lower tier provider's RH. The root of the tree is built by tier–1 RHs.

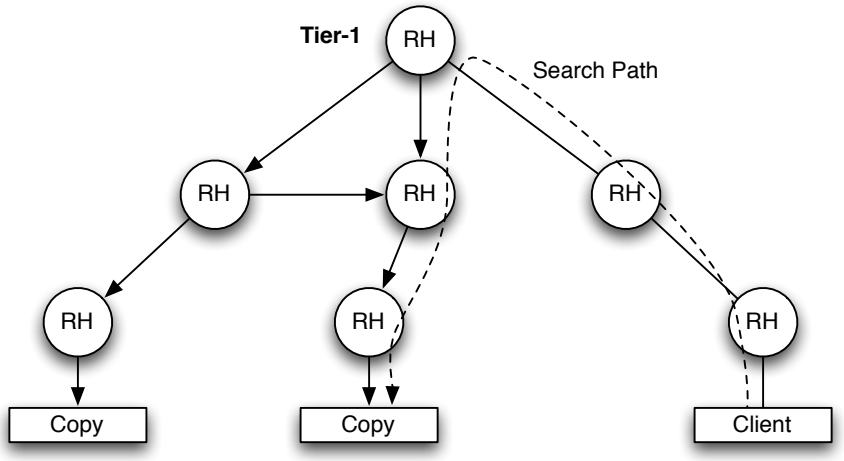


Figure 3.4.: DONA example topology [KCC+07].

Upon connecting to a network the nearest RH is advertised to the client, similar to DNS today. To resolve a mapping the client issues a FIND message including the $\langle P : L \rangle$–tuple to its nearest RH (cf. Fig. 3.4). In case the RH cannot resolve the mapping, it forwards the message to its superior RH. This process is repeated until either a RH is able to resolve the mapping or the FIND message reaches a tier–1 RH. This tier–1 RH knows in which branch a copy of the datum is stored and forwards the message down the respective branch—given that the datum exists. The leaf RH is able to resolve the mapping and points to the actual content. To speed up the lookup process, each RH maintains a registration table that maps an identifier to the next–hop RH and the distance to the copy. In that way direct peerings between RHs of different branches can be used.

3.2.2. Distributed Hash Table Based Mapping

Using a DHT to realize the mapping database results in a highly scalable mapping system. DHTs are mechanisms often used in structured peer–to-peer overlays. They

are designed to handle a huge load and proved to work with thousands of peers, storing billions of entries [SMK+01, KBR06].
DHTs are data organizing structures which are based on hash functions. A hash function, thereby, is an algorithm which maps an input value to a defined output. For DHTs a *variable–length cryptographic hash function* is used. This type of hash function takes a datum of variable length as input parameter and maps it to a fixed–size hash space. The size of the hash space, thereby, is determined by the hash function. To achieve this, the input datum is split into chunks the size of the hash space. Each chunk is then combined with the previous one by a mathematical operation (e.g., xor). Listing 3.1 shows an example implementation in pseudo code.

```
initialize(hash)
foreach chunk in datum do
      hash ← hash ⊕ chunk
return hash
```

Listing 3.1: Sample hash function

One of the properties of a cryptographic hash function is that a minimal change in the input datum results in a completely different and unrelated hash value. This is important in order to achieve an even distribution over the complete hash space in case the input datums are very similar to each other.
For a DHT two groups of input datums are hashed in order to realize the internal data organization. Both datums, thereby, are hashed to a common m bit hash space. As the resulting values are used as keys to find datums in the DHT, it is from here on referred to as key space. The first group of input datums are the addresses of the nodes participating in the DHT (i.e., peer). The second group are the identifiers of the objects which are supposed to be stored in the DHT. The key space, thereby, must be large enough to provide a nearly unique key for all peers and objects[3].
In order to form the DHT, the peers and objects are arranged in a defined structure according to their hash values. Depending on the protocol used the structure can be a ring (Chord), a binary tree (Kademlia) or a d-dimensional torus (CAN). Each peer, thereby, accepts responsibility for a defined part of the key space. Substitutionally for them all, the Chord protocol which will play a role in the subsequent introduced NGI mapping systems is briefly outlined in the following.

Chord

Chord is a structured peer–to–peer protocol based on DHTs [SMK+01]. A DHT stores key–value pairs by assigning keys to different peers. A peer, thereby, stores all the values corresponding to the keys it is responsible for. In Chord, the key space is represented by a ring structure whereat the highest key is succeeded by the lowest one, thus, forming a ring. To map information into the key space, the information is

[3]Note that variable–length hash functions usually are not so–called perfect hash functions and, therefore, collisions can occur.

hashed, resulting in a hash–value equaling the size of the key space. The information can be retrieved at the peer responsible for a specific key subspace. This subspace is determined by the node's key itself and all preceeding keys which are not assigned to another peer with a lower key.

In the following, a brief example of a Chord ring is given to clarify the protocol (cf. Fig. 3.5). In this example, the key space has a size of $4\,\mathrm{bit}$ which means that 16 keys are available. In the ring a peer with the key number 8 is present, and the peer with a lower key value is peer number 4. This means that peer number 8 is responsible for the key subspace 5 to 8. Any information which has a hash value falling into that key subspace is stored at peer number 8. Analogous, information with the bash value 4 is stored at peer number 4. The information with the hash value 2, however, is not stored at peer number 4 as peer number 3 is present in the ring and thus resonsibility for this key value. In case a peer with the key value 6 joins the Chord ring, it takes over the responsibly for the key subspace 5 to 6 from peer number 8. Thus, the information with the hash value 5 is transferred to peer 6.

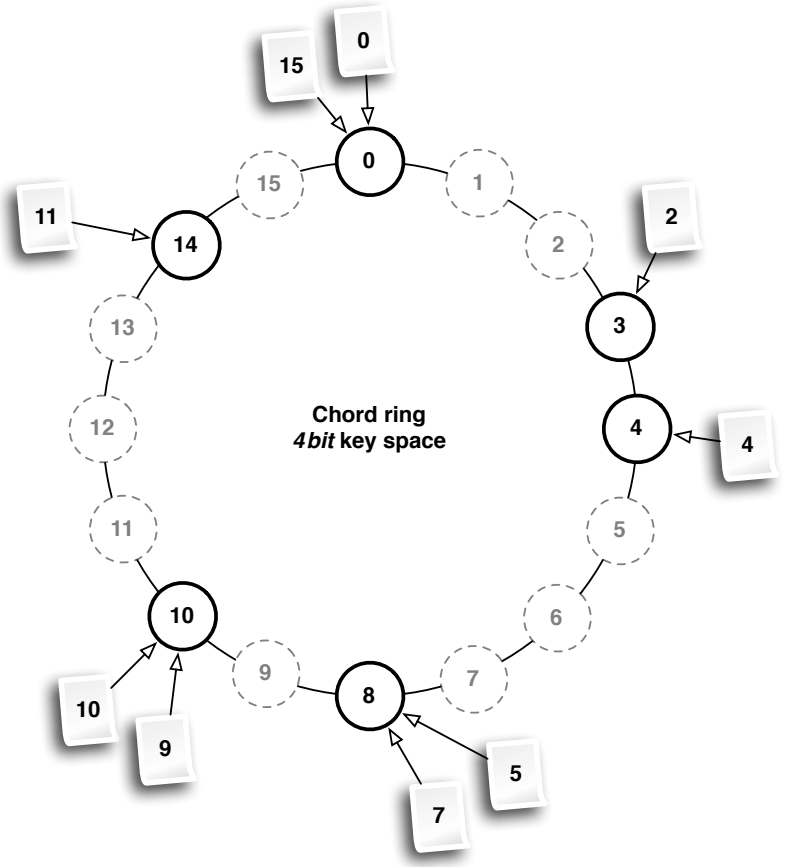


Figure 3.5.: Storage example in a Chord ring.

Example Chord Based Mapping Architecture

As a predecessor to HiiMap, a chord based architecture was presented which used a global DHT as basis for the mapping system [HSKE09]. In this architecture, the network is—as today—divided into ASs, and each provider remains in control over its subnets. The global DHT is hosted on so–called Core Network Routers (CNR) which are located at the border of ASs. A mapping entry, consisting of the tuple $\langle identifier, locator \rangle$ for each subscriber, is stored in the global DHT. The CNRs can be compared to BGP's border gateway router and are responsible for forwarding traffic in and out of their AS. As each provider requires CNRs, they automatically provide resources for the global mapping system. Within each AS a local DHT is provided. This local DHT stores mapping entries of all end points residing within the AS and is hosted on so–called Local Network Routers (LNR). Additionally, cached entries from the global DHT are stored in the local one. The LNRs are regular routers within the AS which are powerful enough to host the local DHT. A sample topology is shown in Figure 3.6.

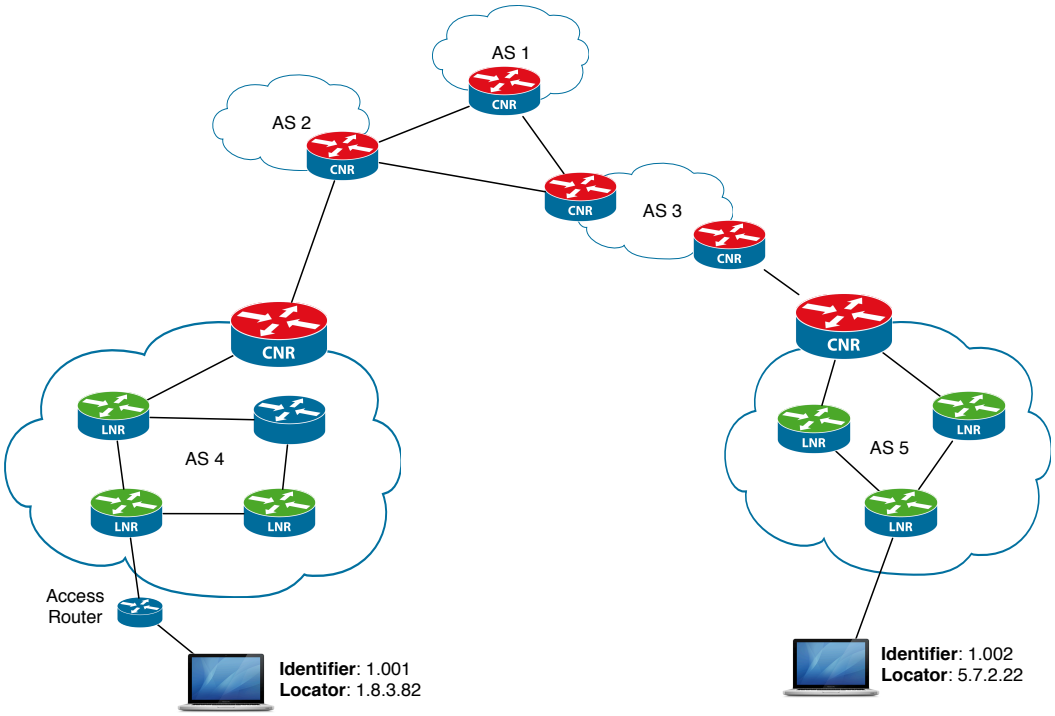


Figure 3.6.: Example topology following the architecture in [HSKE09].

To resolve the mapping between an identifier and a locator, an end point first queries the local DHT of the AS it is attached to. In case the identifier is listed within the local DHT, the query can be answered, and a response is sent back to the end point. An identifier can be listed in the local DHT due to either being a cached entry or the corresponding end point is currently residing within the same AS. If the mapping can not be resolved in the local DHT, the request is forwarded to the global one.

Other DHT–based approaches are [AAER06, FCM+09]. [AAER06], thereby, uses the mapping information for a source routing approach. [FCM+09] uses multiple levels of mapping resolvers (implemented as DHT).

3.2.3. Limitations/Evaluation of Existing Proposals

The main goal of a mapping system is to map identifiers to locators. Any of the previously mentioned mapping architectures is able to fulfill this task. They, however, do not meet all of the requirements listed in Section 3.1.3 (*scalable*, *fast*, *trustful* and *persistent identifier*).

Tree–based with structured identifiers: This approach is able to scale well in regard to the amount of entries as well as to the update dynamic. In case the maximum storage capacity or bandwidth of the existing leaf nodes in the tree is reached, the tree can easily be extended by another branch or an additional level. Each new level in the tree, for example, doubles the amount of mapping servers and, therewith, the available storage capacity.
The initial lookup delay, however, is increased by each level of the mapping tree (cf. Sec. 3.3.3). *Initial*, thereby, means that the client sending a request to the mapping system has no cached information about the storage location of the identifier's mapping entry. In order to find the responsible node, the tree must be completely traversed starting at the root. To avoid this long lookup delay for subsequent queries, each client is required to maintain a cache which stores learned storage locations for mapping entries. The average lookup delay over all queries is determined by the aging algorithm of the cache and the cache miss rate. This miss rate again is determined by the relation between re–occurring and first time queries.
Tree–based mapping systems with structured identifiers provide the foundation of the requirement to be trustful. A mapping is resolved at a leaf node which is responsible for the subpart of the identifier (compare with DNS in Section 3.1.2). The leaf node, thereby, is not dependent on any other node within the tree which stores the mapping entry. This means that the query can be answered by the responsible node and does not need to be further forwarded to another authority. Additional mechanisms, though, must be provided to enable the client to verify the correctness and integrity of the response (e.g., cryptographic signatures).
The structured identifier, however, breaks the requirement of persistent identifiers. This is important in order to allow for relocation of mapping entries without fostering de–aggregation. Whenever a customer decides to subscribe with a novel provider, it might involve that his identifier is required to be resolved in a different branch of the mapping tree. As the tree organization is reflected in the identifier's structure, a branch change would mean that a novel identifier is assigned to the client. This contradicts the property of the locator/identifier separation paradigm to assign life time valid identifiers. The section discussing DNS as a mapping system features an example of this problem (cf. Sec. 3.1.2).

Tree–based with flat identifiers: A similarity between this approach and the tree–based one with structured identifiers is the possibility to increase the storage space by adding branches or levels to the tree. Contrary, however, the lookup process does not scale as well. The root of a flat identifier tree must be aware of all identifiers stored in the tree. It is required to direct queries into the correct branch as the identifier provides no hints about the storage location. Again, the complete mapping tree must be traversed in order to find the leaf storing the mapping entry.
This approach is slightly inconclusive. Traversing a tree only makes sense in case each level provides a piece of the storage location information (compare to DNS, root -> TLD -> subdomain). The benefit of this system is that each level is only required to store a limited set of information. In DNS, for example, the root server must only be aware of all TLD and the responsible authorities. It does not need to store any information about the subdomains prepending the TLD. With the flat identifier tree–based approach, however, the root must already be aware of all identifiers and store information in which branch the mapping can be resolved. Instead of only storing the branch, it could directly store which leaf node holds the mapping entry. The intermediate levels, therefore, do not provide any benefits but add to the lookup delay.
The proposed tree–based mapping systems based around flat identifiers scale in regard to dynamic updates of the mapping entries. An end system usually is aware of which node in the mapping tree is holding its own mapping entry, and the update can be directly sent to this node. The lookup delay, however, is longer compared to the approach with structured identifiers or a DHT–based mapping system. Because the proposed approaches implement mechanisms to reduce the query rate at the root of the mapping tree. This is done by requiring a query to be directed to a leaf node nearest to the issuing client. This leaf node then forwards the query up the branch. The reason behind this procedure is to intercept all queries which can be resolved within the own branch. The worst case, however, is that a query is first required to traverse the mapping tree up and then down in another branch.
Similar to the approach with structured identifiers, this design provides a foundation for a trustful mapping architecture. Again, the node storing the mapping entry is not dependent on any other nodes. The mapping information is likely to be stored on a node close to the customer's home network and might even be under the authority of the customer's provider.
Flat identifiers satisfy the requirement of persistent identifiers. In case a customer decides to change provider subscriptions, the mapping entry can be relocated to another leaf node of the mapping tree. This change is propagated in the tree, and the customer's end system is still reachable under its old identifier.

DHT–based: A mapping system designed around a DHT is able to scale well in regard to the amount of stored entries and the update frequency of its entries. In case the existing mapping peers within the DHT are not able to cope with the load, additional peers can be added without unproportionally increasing the complexity [SMK+01].
One property of DHTs, however, is that an owner of an object cannot control where

in the key space his object is stored. The location is predefined by the hash value of the object and the mapping between the peer and object key space. In terms of the mapping system, this means that a provider is not able to control where the mapping entries of its subscribers are stored. The consequence is a trust issue because the mapping entry of a provider's customer might be stored outside its own responsibility.

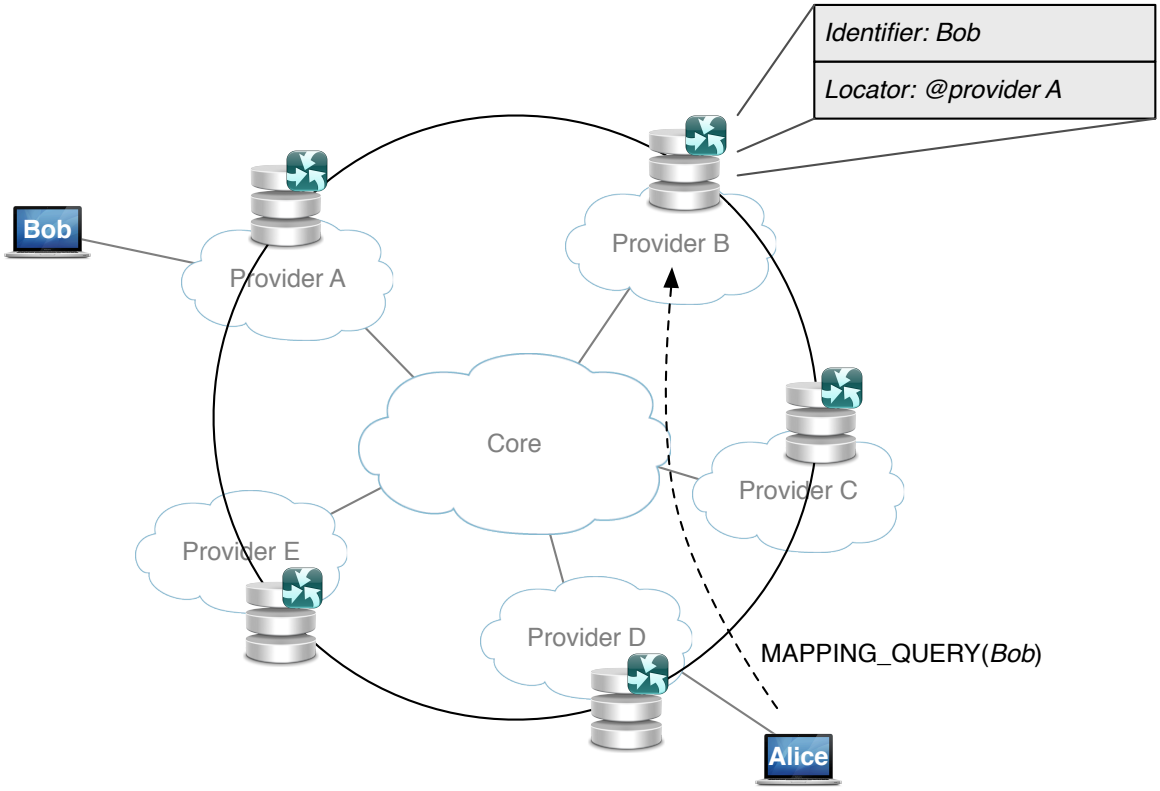


Figure 3.7.: Trust problem of global DHT–based mapping systems.

In Figure 3.7 an example is given where Alice queries the mapping system for Bob's locator. Bob's identifier is his name, and provider B is responsible for the key subspace starting with the letter B. Once the query from Alice arrives at the mapping server at provider B, the server searches its database for the corresponding mapping entry. The entry lists Bob to be a customer of (and currently located at) provider A. The correct behavior would be to return this information to Alice and thereby resolving the mapping request.
In this example, however, provider A and B are assumed to be in a legal dispute and do not act based on fair play. As a result, provider B could either drop the query, respectively respond that the identifier is unknown, or return a wrong locator. In the first case any communication with Bob would be prohibited which could force Bob to change his provider. The second possibility represents a man–in–the–middle attack. By providing a wrong locator, pointing to an end system controlled by provider B,

any traffic between Alice and Bob would pass this third end system before being forwarded to Bob. The latter can be lessened by using only encrypted communication. The first, however, represents a serious trust issue which affects the mapping system. The lookup delay of DHT–based approaches can be weighted as being neutrally depending on the deployed DHT protocol. Using a one hop protocol (e.g., D1HT), a mapping query can be resolved with a maximum of two hops. The first hop is the peer which receives the query from a client, and the second hop already holds the mapping entry.
Similar to the second tree–based approach, the flat identifier guarantees that a customer is able to change his provider without requiring a new identifier. In that way the design goal of the locator/identifier separation paradigm—identities should only change if requested by the owner—is met.
Table 3.2 summarizes the aspects of the different mapping system architectures. The table reveals that no approach satisfies all requirements. The HiiMap architecture introduced in the next section, therefore, combines elements of all approaches to overcome the outlined issues.

	Tree–based *structured identifier*	Tree–based *flat identifier*	DHT–based
Scalable (amount of entries)	+	o	+
Scalable (dynamic updates)	+	+	+
Fast	o	-	o
Trustful	+	+	-
Persistent identifier	-	+	+

Table 3.2.: Comparison between tree and DHT–based mapping systems.

3.3. HiiMap Mapping System

The HiiMap NGI architecture is built around the locator/identifier separation paradigm. It follows the end node aware approach due to the advantages summarized in Table 3.1. The mapping system introduced in this section is a hybrid solution between the hierarchical and DHT–based mapping systems. A special property of the mapping system is its trustful, distributed organization to overcome the limitations of the global DHT–based approach. The work introduced in this section has been published in [HKS+09].

3.3.1. Concept

Before outlining the details of the HiiMap mapping system, a few components and abbreviations are introduced. The interaction between these parts will be explained subsequently. The following subsection covers the trust aspect of the architecture.

- **UID:** The *Unique Identifier (UID)* is a flat, randomized, worldwide unique 128 bit address. It serves the purpose of the identifier in the locator/identifier separation paradigm and therefore is a non–routable address. It is assigned for lifetime and does not change, unless the end point requests a new UID.

- **RP:** In addition to the UID a *Regional Prefix (RP)* is provided. This 8 bit prefix identifies the region in which the mapping for the corresponding UID can be resolved. The notion of regions and the benefit of dividing the mapping system into these will be explained later on. An important property of the RP is that, contrary to the UID, it is allowed to change over an end point's lifetime.The RP is used as a shortcut throughout the hierarchical HiiMap mapping system. This shortcut, however, is not mandatory, and a mapping is resolved solely based on the UID. The RP only speeds up the process of finding the responsible mapping region. Figure 3.8 illustrates the RP and UID combination.

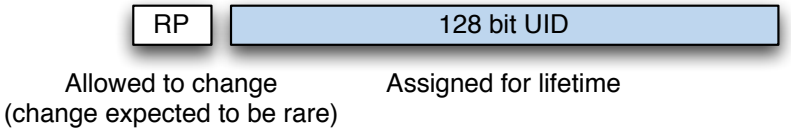


Figure 3.8.: UID with regional prefix (RP).

- **Locator:** In contrast to the UID, the locator is used to route and forward packets to a specific end point. The address family of the locator is not defined by the HiiMap architecture and can be of any feasible protocol (e.g., IPv4, IPv6). For the sake of simplicity, the locator is assumed to be an IPv6 address from now on. A locator is temporarily assigned to an end point from a provider's address pool upon connecting to a provider's network. The locator is allowed to change whenever the end point roams or the provider decides for whatever reason to assign a novel locator.

- **Region:** As mentioned earlier, the mapping system is divided into multiple regions. Each region, thereby, is responsible for the mapping entries of its member end points. An end point's affiliation with a specific region is based on its primary Internet Service Provider (ISP) subscription. This means that the responsible region for an end point does not change even when the end point is temporarily roaming to another region. The mapping responsibility only changes in case an end point permanently relocates to another region, reflected by a change in ISP subscription. The allocation of regions is outlined in Section 3.3.2 covering the trust aspect of the HiiMap mapping system.

- **GA:** The *Global Authority (GA)* represents the root of the HiiMap mapping hierarchy. It, however, does not store any mapping entries. The GA's task is to store the information in which region an UID's mapping entry is stored. It, therefore,

serves as a fallback component in case the current valid RP for a specific UID is not known.

Although the GA logically represents a single point, it can be realized in a decentralized way. This is similar to the DNS root server architecture [Ass11]. A group of servers host the same synchronized information and can handle queries in parallel in order to provide resilience and scalability.

As the affiliation of end points (and, therefore, their UIDs) to regions changes very rarely, the amount of update requests seen by the GA are expected to be low. Additionally, the RP query rate for a specific UID is reduced by caching $\langle RP, UID \rangle$–tuples on end nodes, further decreasing the load of the GA servers.

- **D1HT** D1HT by Monnerat et al. [MA06] is a variant of the Chord protocol. Similar to Chord it organizes the DHT key space in a ring form in which the highest key is superseded by the lowest. The speciality of D1HT is its internal routing database. In Chord each node is only aware of its predecessor and its successor. Additionally, logarithmic fingers are known to take shortcuts through the ring which results in a lookup complexity of $\mathcal{O}(\log N)$ [SMK+01]. D1HT maintains a full routing table. and each peer is aware of all other peers in the ring. The lookup complexity of D1HT, therefore, reduces to $\mathcal{O}(1)$ at the cost of additional update traffic within the ring. In case the ring faces a huge churn rate of its peers, the update traffic produces a significant amount of overhead, and the ring might even become unstable [SMK+01]. As the mapping servers hosting the DHTs in the HiiMap architecture are dedicated servers and expected to be very stable, churn rate is no concern. The lookup optimized D1HT, therefore, is taken as the DHT protocol.

Hierarchical Mapping System

The HiiMap mapping system is a hybrid between a DHT–based and a hierarchical mapping architecture. It is structured in three layers, wherein each layer stores different information. The GA constitutes the root of the mapping system, representing the top layer. The second layer is formed by independent regions, and the third is comprised by the end systems. The left part of Figure 3.9 illustrates a sample topology of the HiiMap mapping system.
The live mapping information is stored in layer 2 of the mapping hierarchy. Whenever a locator change occurs, the end point reports this change to the mapping instance of the region it is associated with. The regions in layer 2, therefore, always hold the most up–to–date mapping information. Layer 1 and 3 of the mapping system hold information to speed up the lookup process and take load off from the second layer. Layer 1, thereby, keeps track of which mapping is resolved in which layer 2 region, and each end system in layer 3 caches mapping information from previous queries. The right part of Figure 3.9 shows a simplified scheme illustrating which information is stored at which layer. The bold printed entries, thereby, serve as the primary key. Please note that additional information might be stored along with the ones shown (e.g., multiple locators, timestamps, public keys).

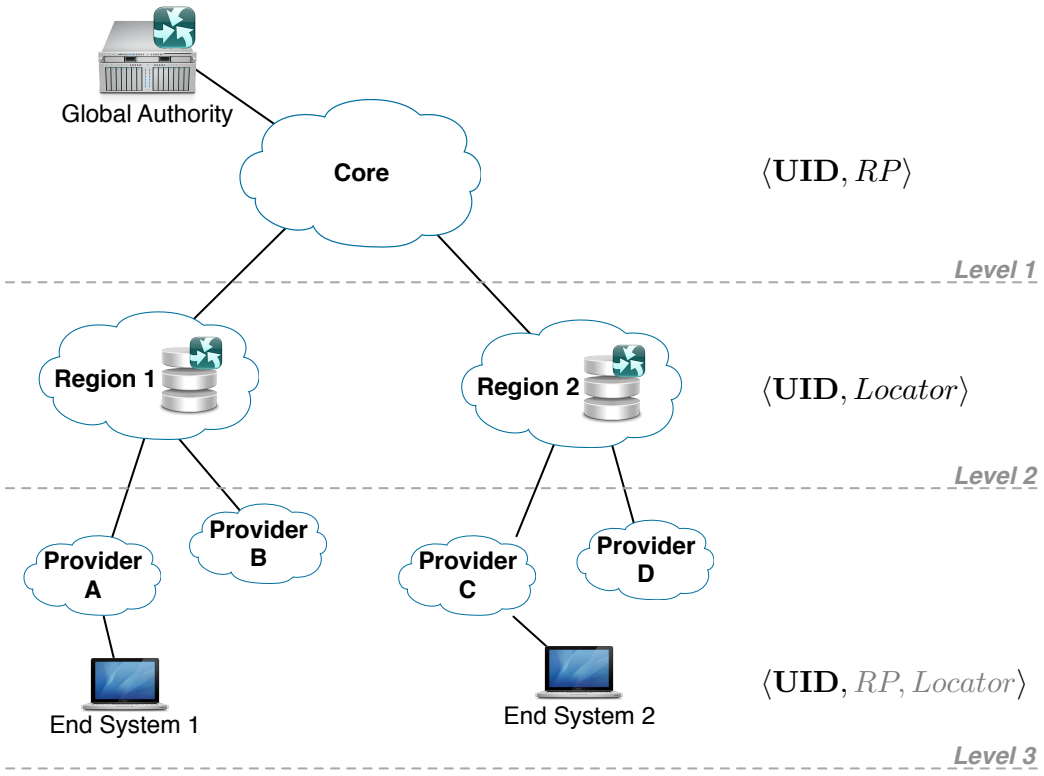


Figure 3.9.: Hierarchical structure of the HiiMap mapping system.

The independent mapping instances in the regions are most critical to the scalability of the architecture. These instances are required to handle a huge amount of queries and updates to the mapping information—contrary to the GA where the queries largely outnumber the amount of updates. Although the HiiMap mapping system does not specify a particular implementation of the layer 2 mapping instances, the use of DHTs is advised. More specifically, the D1HT protocol provides a good tradeoff between the caused lookup delay within the DHT and its signaling overhead for this application. Nevertheless, the operator of a mapping instance may decide to employ other technologies or protocols. From here on, however, the usage of the D1HT protocol is assumed in case the implemented protocol is important (i.e., performance measurement). Figure 3.10 shows the internal structure of a region when using DHTs. The figure also shows a load balancer which is in front of the DHT. Its purpose is to evenly distribute the request and update load over the DHT peers while providing a single entry point per region. Without the load balancer end systems would be required to know all addresses of the peers in the DHT and randomly choose one to average the load over the whole DHT. This, however, would limit the flexibility of the DHT operator to assign to and take off peers from the DHT.

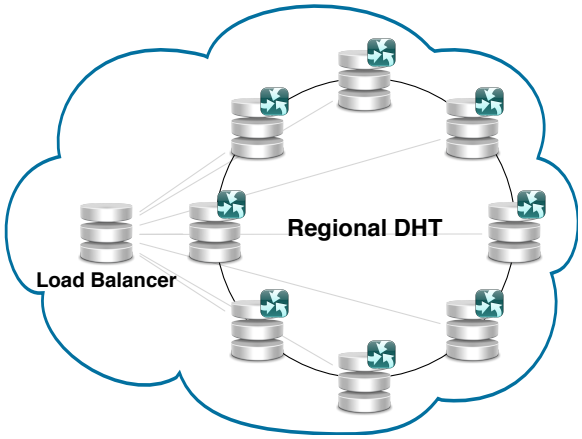


Figure 3.10.: Internal structure of a DHT–based mapping region.

Mapping and Lookup Process

Whenever an end system connects to the network, a two phase connection setup takes place. The first, i.e., local, phase only involves the end system itself and the network access point it wants to connect to. This phase involves admission control, billing issues, etc. and is not covered in this thesis. The outcome of the first phase is an assigned locator and network connectivity provided by the access point for the end system. The second phase has a global scope in which the end system and the mapping system are involved. The end system is required to update its mapping entry with the new locator in order to be reachable by other hosts. This is done by sending a `USR_LOCATION_UPDATE` message towards the mapping region it is associated with. This message contains the end system's UID, one or more locator and optional information to be stored in the mapping system. To prevent identity hijacking, the message must be signed by the end system. The details about the cryptographic aspects are covered in more detail in Chapter 4 and have been published in [Han12]. The complete mandatory message has the following format (omitting optional fields):

$$\text{USR\_LOCATION\_UPDATE}\,(\texttt{UID, locator, signature}_{UID,locator})$$

The message is verified by the mapping instance, and afterwards the corresponding mapping entry is updated.
To resolve a mapping for a specific UID, at least the UID must be known. In case the corresponding RP has not been cached or learned by other means (e.g., link on a website, DNS entry), the GA must be queried in a first step. This will return the responsible region for the queried UID. The following two messages are involved for the request to and response from the GA:

$$\text{USR\_REGION\_REQUEST}\,(\texttt{UID})$$

USR_REGION_RESPONSE(UID, RP, $\text{signature}_{UID,RP}$)

As the responsible region for a certain UID is expected to change very rarely, this information can be cached to avoid further USR_REGION_REQUESTs for this UID. In a next step, the responsible region is contacted to resolve the actual mapping. The required messages are as follows (again omitting optional fields):

USR_LOCATION_REQUEST(UID)

USR_LOCATION_RESPONSE(UID, locator, $\text{signature}_{UID,locator}$)

Only the UID is required to be sent to the responsible region. The returning message will contain the UID itself, the locator listed in the mapping entry and optional fields also stored in the mapping system. Attached is also a signature from the mapping instance to prove that the returned values have not been modified along the way. The UID is also again included in the message in order to be able to identify responses in case multiple request have been sent out.
In case no entry for a specific UID at neither the GA nor the responsible region was found, an error message is returned in which the error_code lists NX_IDENT:

ERROR_MSG(UID, error_code, $\text{signature}_{UID,error_code}$)

The complete resolving process is illustrated in the message flow diagram in Figure 3.11.

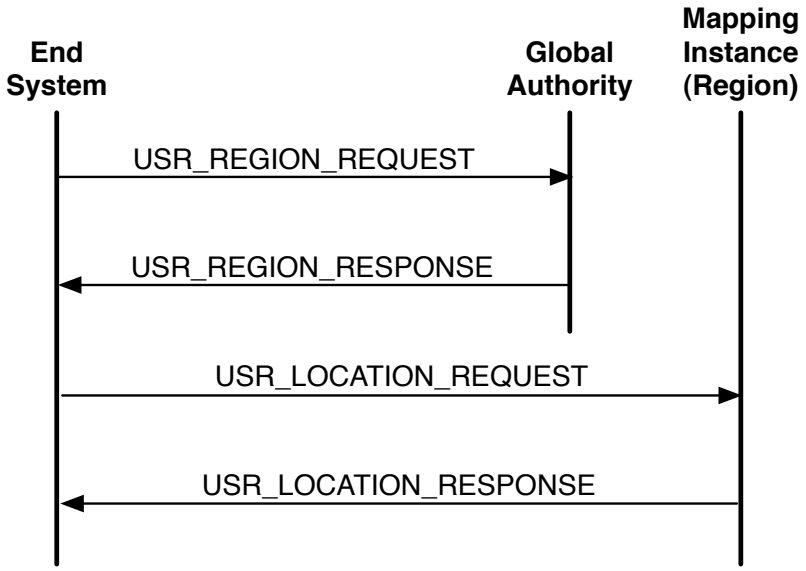


Figure 3.11.: Message flow diagram of the HiiMap mapping resolution.

End System Mobility

An important requirement for an NGI architecture is to support mobility as outlined in Chapter 2.2. The locator/identifier separation paradigm addresses this functionality but the mapping system must also be able to handle mobility. To this end, the HiiMap mapping system differentiates between two kinds of mobility. The first is referred to as *roaming* and reflects any temporary connection towards the network outside an end system's home network. The second kind is called *relocation*. A relocation occurs whenever an end system decides to change its home network (i.e., ISP subscription), and the new home network is associated with another region.

Roaming Whenever an end system connects to an alien network outside of its home network, a new locator is assigned to it. This locator must be updated in the mapping entry of the end system. To initiate this update the end system sends a signed `USR_LOCATION_UPDATE` message to its responsible region. Each subsequent mapping query will return the new locator and the end system is reachable by other nodes. The location update, therefore, does not differ from the one sent due to a novel assigned locator within an end system's home network.

Relocation The more challenging aspect is *relocation*. This occurs whenever an end system permanently changes its home network and associates with another region. The challenge, thereby, is the expectation that the identifier of the end system should not be required to change. This relates to the requirement of persistent identifiers.
One solution would be, once associated with a region that this region remains the responsible region as long as the end point holds on to the identifier. In that way the identifier is never required to change, and mapping for the identifier can always be resolved. This approach, however, has a major drawback. The responsible region would be required to charge the end system for hosting its mapping entry as the end system might not be a customer of the region anymore. For the end system this means that it is dependent on the region, and the region has a monopole over the identifier. The end system would not be able to choose its mapping provider and has no ability to associate its identifier with another region. The region, therefore, could increase the mapping fees or set terms to the disadvantage of the end system which depends on its identifier (e.g., because customers or applications rely on a service behind a specific identifier).
In HiiMap, therefore, it is possible to associate an UID with a different region. This process is called *relocation* and results in a change of the RP. In that way the identifier (UID) remains the same while the mapping is resolved by a different region. To initiate the relocation, a release of the UID is signed by the end system and sent to the old region. The new region then can request a transfer of the UID, again authorized by the end system. Finally, the mapping entry in the new region is created and the RP at the GA updated to reflect the new region. The relocation process is illustrated in Figure 3.12.
All communication partners caching the old $\langle UID, RP \rangle$–tuple still send their mapping request to the old region due to the cache entry. The old region, however, re-

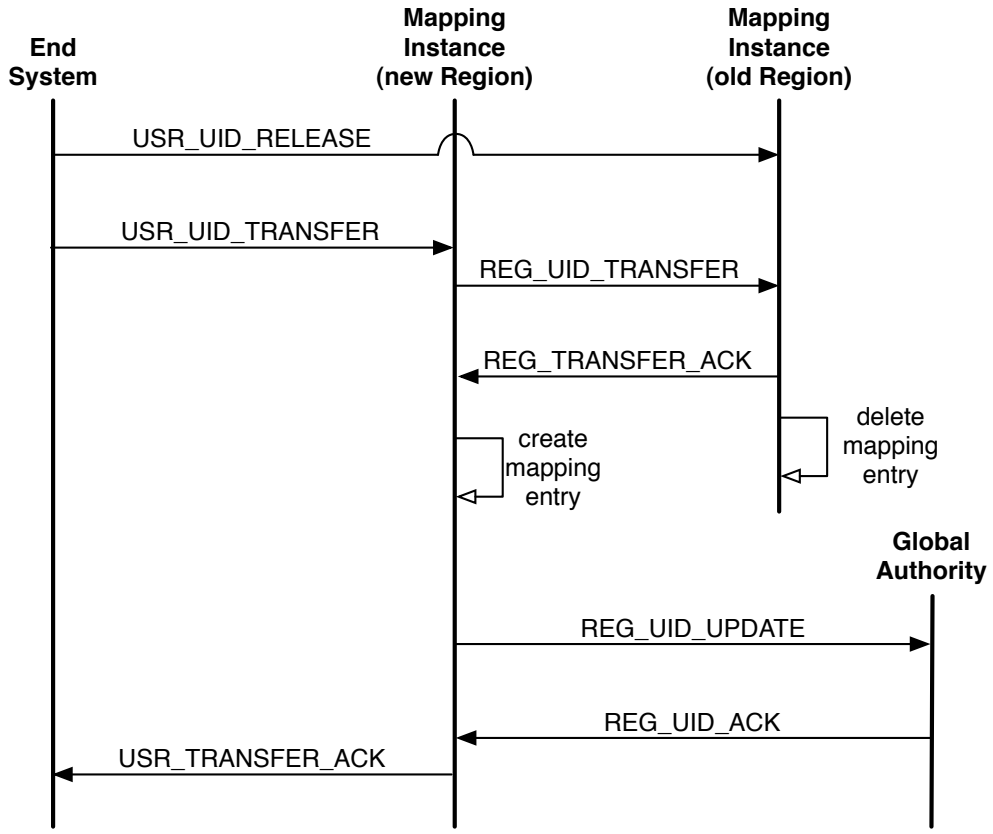


Figure 3.12.: Message flow diagram illustrating the HiiMap relocation process.

sponds with `NX_IDENT`, and the communication partners are required to query the GA for the new RP.

3.3.2. Trust

The major disadvantage of the DHT–based mapping systems introduced in the previous section was the *trust problem*. A provider is not able to control on which peer in the DHT its client's mapping entry is stored and, therefore, can not guarantee its resolution.
The goal for the HiiMap mapping system was to build a system which solves this trust problem. This led to the introduction of regions which split the mapping space into several subparts. These subparts on their own, however, are not a solution to the trust issue. Rather, the design and setup of the regions help to overcome the problem. In HiiMap, providers are grouped into regions based on their geographical location. Each region, thereby, represents a country[4], and providers of the same country work

[4]The term *country* in this context refers to a political state or territory acknowledged and recognized by the United Nations [Uni06].

together to fund the mapping system in their region. In case a provider operates in multiple countries, it either registers all of its clients in its home region or splits the affiliation based on its national subsidiaries. In either way, each provider is required to fund a share of the mapping instance in its region based on the amount of its subscribers. This guarantees a fair split of the mapping costs in each region.
In each region a non–profit organization operates the mapping system which also collects the funds from the providers. The reason behind this approach is that no provider should have direct access to the mapping servers. In that way no provider is able to manipulate the mapping entries to the disadvantage of its competitors. An example for a non–profit organization are the domain name registries DENIC in Germany or Nominet in the United Kingdom.
There are two reasons as to why limiting a region to the boundaries of countries. The first is the relatively stable state of countries. Although it does occur, it is rather rare that a country vanishes or a novel one is constituted. This means that the region prefix table and the mapping system partition are not required to change often. Furthermore, in case a country splits or is reunited, the change only affects the old region/regions and non of the other regions. As of today (late 2011) 192 countries are recognized by and are members of the United Nations [Uni06]. There are further few countries which are recognized but not member–states. This means that the $8\,\mathrm{bit}$-wide RP is sufficient and also future country splits can be reflected without a change to the architecture.
The second reason for the partitioning based on countries is the common legal system within each country. A legal dispute between competing providers or the non–profit organization can be solved by local law and does not require international contracts or courts. It is also locally contained and does not affect other regions.
Contrary to the global DHT–based mapping system approach, this architecture introduces three aspects in order to solve the trust problem. First, the mapping system is not provided by commercially oriented providers. Second, disputes can be solved based on a common law among the involved parties, and third, conflicts are constricted to a single region and do not affect the global mapping hierarchy.
A downside of the country–based partitioning into regions is that the government of a country might request to take control over the mapping instance. This would enable the government to manipulate mapping entries and restrict the resolution of certain mapping requests, rendering the affected end system unreachable. The government, therefore, would be able to foreclose communication with specific end systems. Although this is an undesired aspect, it does not open novel possibilities for a government. Even in today's Internet architecture, governments could filter network traffic or cut–off end systems from the rest of the Internet.

3.3.3. Evaluation

In this section, the HiiMap mapping system is evaluated. The scalability aspect is assessed analytically whereas the performance of the system is measured with the help of a prototype. Information about the prototype design and the specification of

the HiiMap mapping system protocol are given in Chapter 6.2 and Appendix A.

Scalability

The HiiMap mapping system stores information in two different layers. The topmost layer, represented by the GA, is required to store the mapping between all UIDs and their responsible regions. The lower layer is formed by the regions which store the actual mapping information between UIDs and locators. Due to the utilization of DHTs within the regions, the storage capacity can be easily increased as new mapping entries must be stored. Although it is not possible to add an unlimited amount of peers, DHTs have been tested to be stable with a couple of thousand peers [SMK+01]. The GA, therefore, is the challenging element in the mapping hierarchy. There are three constraints which determine the GA's capabilities. These are:

- Storage capacity for the database
- Number of transactions per second in the database
- Network bandwidth

Storage capacity: The GA must store information for all registered UIDs. A single entry in the GA's database, thereby, consists of a $\langle \mathrm{UID}, \mathrm{RP} \rangle$–tuple. The HiiMap prototype implementation (cf. Chap. 6.2) uses a SQL compatible database to store the mapping information at the GA. To estimate the size of a single entry, the actual storage requirement of the prototype is taken for the calculation. The UID entry serves as a primary key and is stored in a `VARCHAR(16)` field[5]. In addition to the $16\,\mathrm{byte}$ of the `VARCHAR` content, $2\,\mathrm{byte}$ are required for SQL's internal organization of the data type. This results in $18\,\mathrm{byte}$ per UID field. The RP is stored as a single byte in the database. Each database entry at the GA, therefore, sums up to:

$$\begin{aligned} \mathcal{S}_{GA\_entry} &= \mathcal{S}_{UID} + \mathcal{S}_{RP} \\ &= 19\,\mathrm{byte} \end{aligned} \tag{3.1}$$

A prediction of the amount of registered UIDs for a future point of time is quite difficult. A linear projection of the latest subscriber growth, however, might serve as a lower bound. Based on the growth rate according to the *IPv4 address report* by Huston [Hus10], the amount of Internet participants will reach approximately $6\,\mathrm{billion}$ in 10 to 15 years. This would result in a minimum GA storage capacity requirement of $114\,\mathrm{GB}$.

Number of transactions: Various aspects of a database system determine the number of transactions a database system is able to process. The main factors are processing power, memory capacity, harddisk speed and the size of the database. [Tra11] lists the time it takes to retrieve an entry from a database in relation to its size on modern database systems. Although HiiMap is designed as a NGI architecture and projected

[5] The prototype uses $128\,\mathrm{bit}$ UIDs.

to be used in 10 to 15 years, this evaluation is based on technology available today[6]. The highest ranking, non–clustered database system tested by [Tra11] is able to process $101,419$ transactions per second in a $728.97\,\mathrm{GB}$ database. This data is used in the following.

Network bandwidth: Each request causes two network packets, i.e., the query from the client and the response sent by the GA. Assuming a full–duplex connectivity, only the larger of these two packets is relevant. The larger one is the response listing the queried UID and the RP. Each packet of the HiiMap protocol features a $7\,\mathrm{byte}$ header—$1\,\mathrm{byte}$ protocol version, $1\,\mathrm{byte}$ command code, $4\,\mathrm{byte}$ payload length and $1\,\mathrm{byte}$ checksum (cf. App. A). The payload of the `USR_REGION_RESPONSE` packet has a total size of $17\,\mathrm{byte}$. This results in a packet length of $24\,\mathrm{byte}$. Including the IPv6 header ($40\,\mathrm{byte}$) and an Ethernet MAC frame ($18\,\mathrm{byte}$), $82\,\mathrm{byte}$ are transferred per packet. A single $1\,\mathrm{GBit/s}$ interface, therefore, is able to handle $1.52\,\mathrm{million}$ requests per second.

Analogous, the values for the peers in the DHT can be estimated. A peer's storage capacity, thereby, is dependent on the size of the DHT and thus not considered. To calculate the number of requests in relation to the network bandwidth, two packets per peer must be considered. A peer of the DHT must accept and respond to queries from clients and also be able to forward the query into the DHT in case it does not store the corresponding mapping entry. The size of the two packets combined (forwarding `USR_LOCATION_REQUEST`s and sending `USR_LOCATION_RESPONSE`s) is $94\,\mathrm{byte}$[7]. Table 3.3 summarizes the values for the GA and a single peer within the region.

	min. capacity (GB)	transactions (/sec)	requests (/sec)
GA	114	$101 \cdot 10^3$	$1.52 \cdot 10^6$
DHT peer	–	$101 \cdot 10^3$	$822 \cdot 10^3$

Table 3.3.: Scalability evaluation for the Global Authority.

The table shows that the limiting factor in terms of scalability is the number of transactions the GA's database is able to process. To verify whether the architecture in regard to the scalability of the GA is feasible, values from today's DNS system are taken.

[6] Projecting technology advancement over a 10 to 15 year time span is not very reliable. It, however, is safe to say that today's high performance systems are easily outperformed by future systems.

[7] Please note that `USR_LOCATION_UPDATE` messages are also sent to the peers in the region. This message, however, contains less bytes then the others and, therefore, does not add to the worst case calculation.

Feasability: Based on DNS traffic statistics as seen by the Leibniz Supercomputing Center (LRZ) in 10/2011 [Lei11] (AS 12816; $1,000\,\mathrm{qps}$ from approx. 40,000 nodes), every node starts in average 0.025 DNS queries per second (qps). For the GA this means that 6 billion subscribers would send 150 million `USR_REGION_REQUEST` messages per second. Many DNS queries in the LRZ AS, however, do not reach an outside DNS server. They can be answered by the LRZ's own server due to the use of caching. Jung et al. [JSBM02] analyzed the caching behavior of DNS resolvers and experienced a 80 - 86% cache–hit–rate for an AS. Assuming a similar caching hit rate for a client's $\langle \mathrm{UID}, \mathrm{RP} \rangle$–cache, the GA faces a query rate of $30 \cdot 10^6/\mathrm{sec}$. This means that with today's database technology and a projected subscriber number of 6 billion, around 300 instances of the GA must be provided. This seems to be a feasible number, considering that DNS maintains 257 root servers [Ass11].

Performance

To evaluate the lookup performance of HiiMap, the prototype implementation was compared to today's DNS. The HiiMap prototype was deployed within the G-Lab experimental facility [TG09].

Domain Name System Usually DNS heavily relies on caching on each level of its hierarchy to reduce the lookup time. As discussed in Section 3.2.3, however, a DNS structure serving as mapping system cannot rely on caching as the entries are expected to outdate rather quickly. The following test, therefore, was based on trace queries. This means that the first query containing the queried domain was sent to one of the DNS' root servers. These servers answer with a referral to the next hierarchy, which again is queried by the test program. This is repeated down the hierarchy until the query can be resolved by one of DNS' leaf nodes.
The test routine ran $5,000$ queries, randomly choosing between 30 different domains for each query. The domains were all part of the G-Lab experimental facility and point to individual machines of the G-Lab cluster (`glab010.i4.tum.german-lab.de` through `glab200.i4.tum.german-lab.de` and `glab010.i4.kit.german-lab.de` through `glab100.i4.kit.german-lab.de`). To resolve the queries, three referrals were issued by DNS before the fourth level was able to answer the request. The authority chain, thereby, consisted of a root server, followed by a DENIC server (responsible for the `.de` TLD), then a DNS server from the University of Kaiserslautern (authority for the `german-lab.de` domain) and either a server from LRZ (`*.tum.german-lab.de` nodes) or from the Karlsruhe Institute of Technology (`*.kit.german-lab.de`).
As the root server locations are spread all over the world, only root servers located within Germany were selected for the tests in order to minimize the Round Trip Time (RTT). These servers are A, C, I and J hosted in Frankfurt, F in Munich and G in Stuttgart. For each query, a root server was randomly chosen from this list. Likewise, the server of the next level was randomly drawn from the referral list in the higher layer's response. The test was conducted on a Friday at approximately 10 a.m.. Re-

sults from test runs on other days, however, showed no significant difference. Figure 3.13 shows the results of the test.

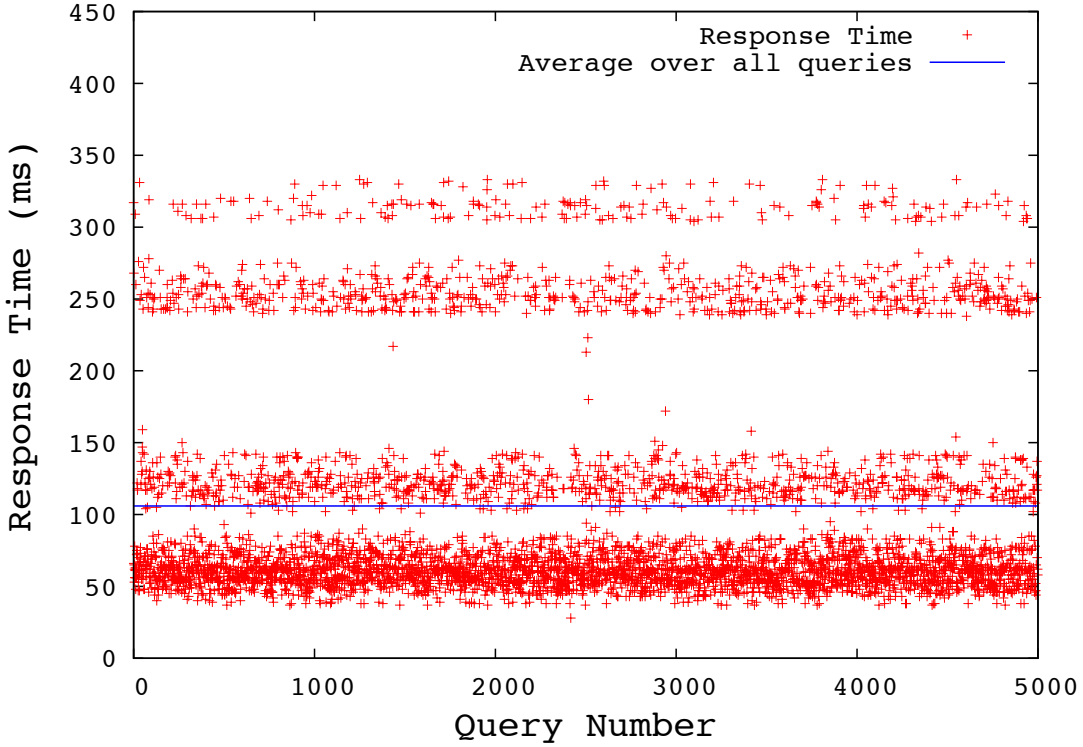


Figure 3.13.: Results of the DNS performance test.

Although the average lookup delay was $106\,\mathrm{ms}$ (blue line), the graph shows that different root servers introduced a different amount of lookup delay. At least three different layers can be distinguished. The upper most crosses in the graph do not represent another layer. They result from irregular delays across all queries. The fastest responses had a lookup delay of roughly between $50\,\mathrm{ms}$ and $80\,\mathrm{ms}$. For the comparison with HiiMap these values are taken from now on.

HiiMap Likewise to DNS, HiiMap was tested with $5,000$ queries. The setup for the test featured a single GA pre–filled with $50,000$ entries and hosted in Kaiserslautern's part of the experimental facility. Furthermore, two regions were instantiated, each hosting $25,000$ mapping entries. The regions equally consisted of a loadbalancer and five peers in the DHT. The replication factor within the DHT was set to two. This means that each peer stores two more datasets additionally to its own. One region was hosted on machines located at the Karlsruhe Institute of Technology and the other one at the Technische Universität München.
For the sake of comparability, caching was also disabled at the HiiMap client issuing the queries. This means that for each query the GA has to be queried first in order to retrieve the RP for the UID. Subsequently, the responsible region was queried to resolve the mapping request. The $5,000$ UIDs for the queries were randomly selected

from the $50,000$ entries stored in the mapping system. Figure 3.14 shows the result from the HiiMap test run.

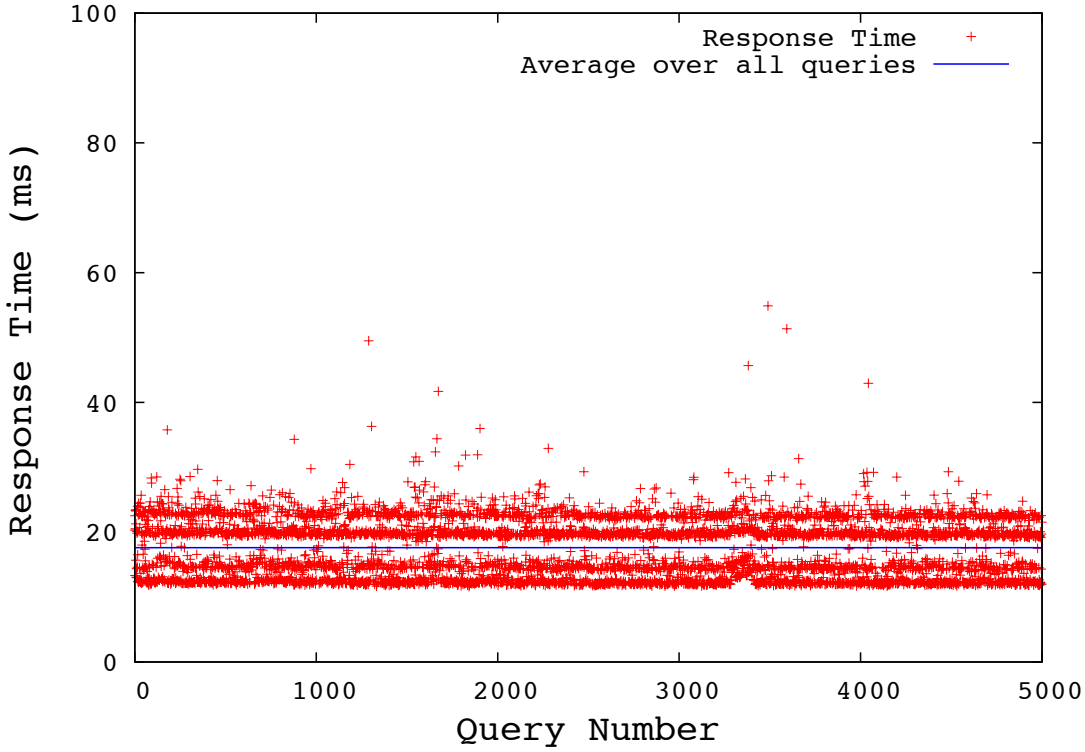


Figure 3.14.: Results of the HiiMap performance test.

The average lookup delay of HiiMap was $17.6\,\mathrm{ms}$ (blue line). The two different layers in the graph result from the different RTTs to the sites were the regions were hosted.

Comparison The results of the tests show greatly differing average lookup delay values ($106\,\mathrm{ms}$ vs. $17.6\,\mathrm{ms}$). A conclusion, however, cannot be drawn from these values alone. There are multiple aspects which influence the measured times for both systems. A handicap of DNS—increasing the lookup delay—is the productive use of the system during the test. This means that the DNS server had to handle multiple concurrent queries not related to the test whereas the HiiMap system was only processing the queries generated by the test client. On the other hand, the DNS servers were running a highly optimized software, implemented in native C code and executed on dedicated hardware. Contrary, the HiiMap mapping system prototype is implemented in Java and executed within a virtual machine. The nodes of the G-Lab experimental facility are running the PlanetLab virtualization software [TPU03], and multiple users run different experiments on each physical hardware. This means that not the full hardware resources are dedicated to the HiiMap prototype, and the load of the system can vary depending on other experiments conducted in paralell.
A further difference between both tests is the RTT between the test clients and the queried server. For the HiiMap test, all nodes (client and mapping system) were con-

nected via the *Deutsche Forschungsnetz (DFN)* which provides a dedicated network for Germany's academic and research institutes. Although the last two levels of the DNS authoritative server are reachable via the DFN, the root and `.de` authoritative server are outside the DFN, and RTTs are significantly higher. Especially the RTT between the test client and the root server disproportionately increases the lookup delay. Table 3.4 lists the RTTs between the client and the different sites:

Sites	RTT	
Technische Universität München	$0.5\,\mathrm{ms}$	DFN
Karlsruhe Institute of Technology	$8\,\mathrm{ms}$	
University of Kaiserslautern	$9.5\,\mathrm{ms}$	
DENIC	$10 - 20\,\mathrm{ms}$	external
root server	$10 - 165\,\mathrm{ms}$	

Table 3.4.: RTTs between the client and the queried server.

Comparing the measured results from Figure 3.13 and 3.14 with the RTT ping times reveals an interesting aspect. The major part of the lookup delay is caused by the RTT. The processing time at either the DNS server or HiiMap mapping nodes has only a limited impact. Therefore, the limitations of the prototype implementation or the load of the productive DNS servers do not play a major role and are rather insignificant to the test results. This can best be seen on the results in Figure 3.14. The fastest lookups in the HiiMap system took $11.5\,\mathrm{ms}$ to $12.5\,\mathrm{ms}$. These were the queries where the responsible region of the UID in the request was at the Technische Universität München. Before querying the responsible region, a request for the RP was sent to the GA, hosted at the University of Kaiserslautern. This means, the RTTs sum up to $10\,\mathrm{ms}$ ($9.5\,\mathrm{ms} + 0.5\,\mathrm{ms}$). Compared to the total lookup delay of $11.5\,\mathrm{ms}$, it shows the significant impact of the RTT.
Having identified the importance of the RTT for the lookup delay shows the advantage of the HiiMap architecture over DNS in this regard. HiiMap requires no more than two hops (levels) in order to resolve a query in case the RP was not cached or learned otherwise (GA & region). DNS, in contrast, requires at least three levels (root server, TLD server & domain server). Assuming that the DNS tree must be extended by several layers in case it should be used as a mapping system, additional RTTs add to the lookup delay.

Prototype Limitation Another important aspect is shown by the HiiMap measurement result. This is shown in Figure 3.15 which is a close–up of the data set shown in Figure 3.14.
The close–up reveals two different lookup speeds at each site. This is represented by the two *lines* which are formed by the measurement points. Below the average (blue line) all response times from the region hosted in Munich are located (except some spikes). The lower and somewhat *thicker* line has its median at approximately $12\,\mathrm{ms}$ to $12.5\,\mathrm{ms}$. The median of Munich's upper (a bit *thinner*) line is around $14.5\,\mathrm{ms}$. The

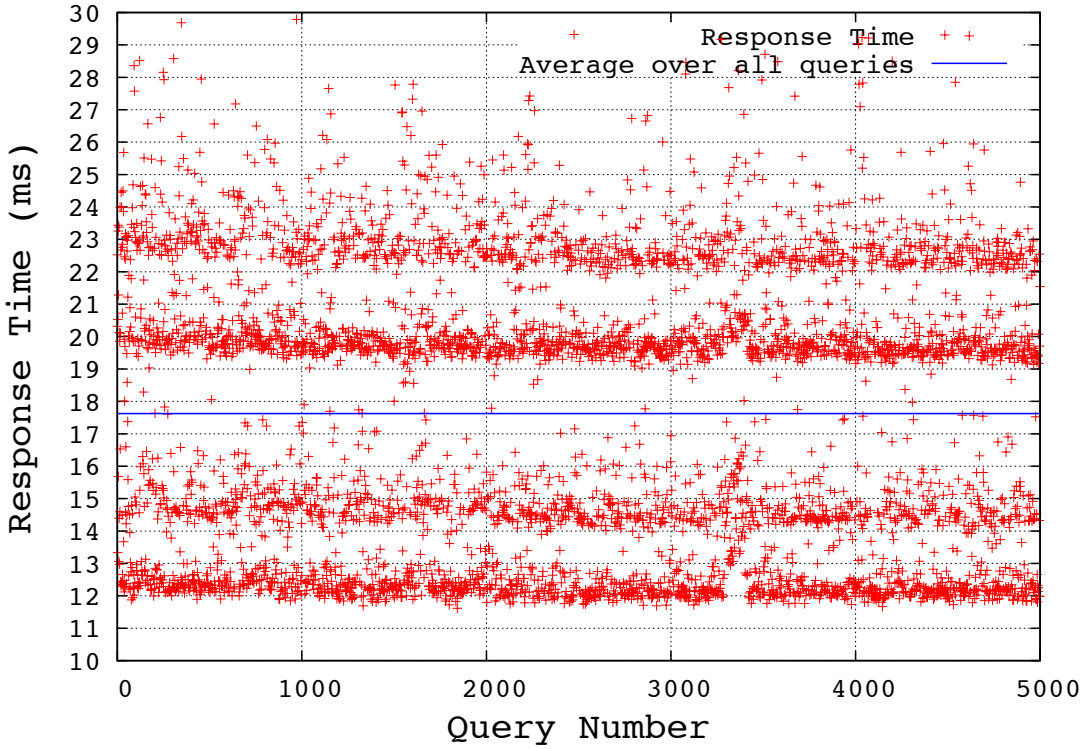


Figure 3.15.: Close up of the HiiMap performance test results.

cause of these two lines and their relative *thickness* roots from the DHT used to store the mapping entries. A query is sent to a region's load balancer and then forwarded to a random peer of the DHT. In case the peer is able to resolve the mapping request, the response is sent back to the querist via the load balancer[8]. If the peer, however, is not able to answer the query, it passes it on to the responsible one within the DHT. As mentioned earlier, the replication factor within the DHT was set to two. Before forwarding a query into the DHT, a peer checks its own dataset and also the replicas it stores. In case the query can be answered based on the stored replicas, the query is not forwarded and immediately resolved. The $2.5\,\mathrm{ms}$ gap between the two lines is the time it takes to forward the query within the DHT (RTT plus processing time at the peer). The load balancer as well adds another $2.5\,\mathrm{ms}$ to the total lookup delay—although this cannot be derived from the graph. The difference in thickness can be explained by the replication factor. In a five node and balanced DHT, each peer stores $3/5$ of all entries. This means that only $2/5$ of all queries must be forwarded to another peer in the DHT.
As a productive HiiMap mapping system would require more than five nodes in the DHT, the second line becomes the significant one. A goal for a real world implementation of the mapping system must be to shrink the gap between the first responder

[8] A direct response by the prototype implementation is not possible. The prototype HiiMap protocol is based on UDP/IPv4, and the test client was situated behind a NAT gateway. Therefore, the load balancer is required to answer the request in order for the response to be forwarded to the client (cf. Chap. 6.2).

and the responsible peer within the DHT as well as between the load balancer and the first responder. The hardware nodes at a single site of the G-Lab experimental facility are interconnected by $1\,\mathrm{GBit/s}$ Ethernet lines, and the RTT between two nodes is approximately $0.2\,\mathrm{ms}$. This means that the major part of the additional delay in the prototype setup is caused by the implementation and the virtualization of the nodes. A native C implementation on dedicated hardware should significantly reduce the additional delay.

3.4. Conclusion

In this chapter a novel addressing scheme for a NGI architecture was discussed. The so–called locator/identifier separation paradigm solves several challenging issues of today's Internet architecture. This addressing scheme, however, requires a new entity within the network, the so–called mapping system. After outlining the requirements for such a mapping system, various proposals for a mapping architecture, ranging from today's mechanisms to completely novel concepts, were briefly described and evaluated. The evaluation, however, showed that non of the proposals met all of the requirements.
The major contribution of this chapter is the introduction and evaluation of the Hierarchical Internet Mapping Architecture (HiiMap). This approach combines the strength of tree–based and DHT–based mapping systems and proved to meet the requirements expected from a NGI mapping system (*scalable*, *fast*, *trustful* and *persistent identifier*). The subsequent analytical and experimental evaluation of the concept showed the resource requirements of the mapping architecture and revealed the concept's performance advantage over today's Domain Name System. The practical evaluation was carried out in the G-Lab experimental facility. In Appendix B a migration strategy is described how to introduce the HiiMap architecture to the Internet.
Although this chapter did not focus on the security requirements of a mapping system, the foundation to incorporate such functionality was laid out (i.e., the trust aspect). As security plays a fundamental role for a NGI architecture, this topic is discussed in the next chapter.

4. Integrated Security Architecture Based on Distributed Public Key Infrastructure

Providing security functionality is an important aspect of any Next Generation Internet (NGI) architecture. Contrary to today's Internet design, the security functionality must already be embedded in the architecture and not provided as an add–on. In this chapter, the Hierarchical Internet Mapping Architecture (HiiMap) security framework is introduced. Even though the description of the HiiMap architecture and the security framework is split into two chapters, this does not mean that the architecture and the framework are two separate parts. The security framework is an integral component of the HiiMap architecture and is fully integrated.
Before introducing the framework, some state of the art fundamentals are summarized and other novel security concepts are briefly discussed. The limitations of these concepts are outlined before the elements of the HiiMap security framework are described and evaluated.

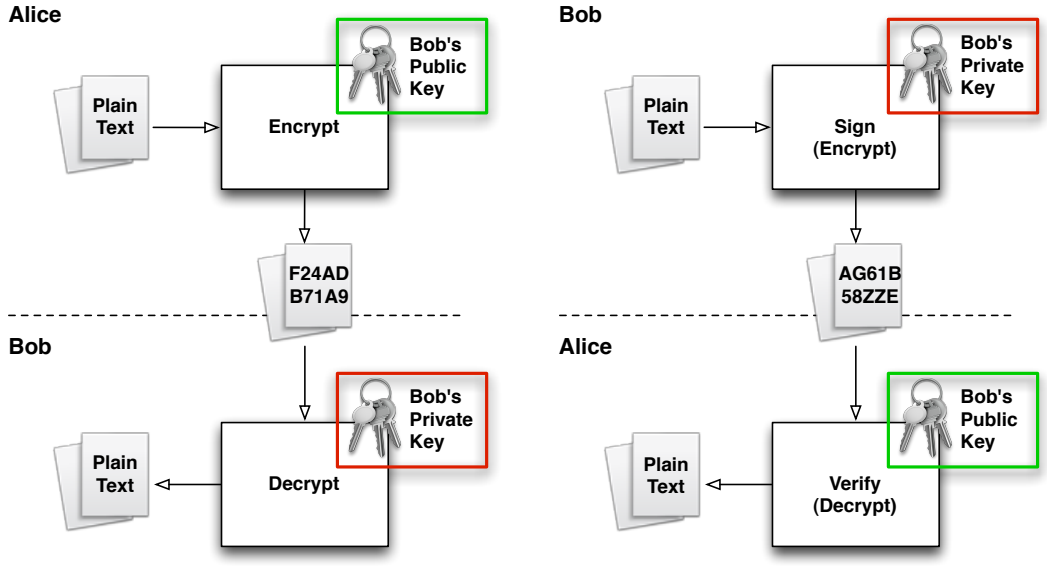
4.1. State of the Art

In this section fundamental cryptographic concepts are briefly summarized. These concepts are utilized by the novel security proposals and the HiiMap security framework.

4.1.1. Public Key Cryptography

Traditional symmetric cryptography requires a shared secret between communication partners. This secret is used to en- and decrypt a confidential message. Albeit its simplicity and elegance sharing the secret is not trivial. Prior to an encrypted communication both partners must agree upon the secret. The problem arises as on the Internet it cannot be assured that involved parties can meet physically or even know each other beforehand to exchange the secret [Eck06, LOP05].
Contrary to symmetric, asymmetric cryptography does not require a pre–shared secret. Asymmetric or public key cryptography is based on a two key mechanism. Two separate but linked keys, thereby, are required. One for encrypting a plain text and the other to decrypt the cypher text. The important aspect is that neither key will do both functions. Furthermore, with the proper algorithm it is impossible to derive one key from the other [Eck06].
The owner of a key pair creates a mathematically linked key pair (i.e., a public and a private key) and publishes the public key. The private key must be kept secret and not disclosed to anyone. By publishing the public key anyone holding a copy of it is able

to encrypt a plain text in a way that only the key owner is able to decrypt it (cf. Fig. 4.1(a)). The other way around is also possible. Encrypting a plain text with the private key results in a message only decipherable by the public one. Though this does not provide any confidentiality as the public key can usually be copied by anybody, it is a proof to a public key holder that the message was encrypted by the key owner (cf. Fig. 4.1(b)).



(a) Using Bob's public key to encrypt a message only he is able to decrypt

(b) Using Bob's public key to verify his signature

Figure 4.1.: Encrypting (a) and signing (b) a message with public key cryptography.

Several algorithms have been proposed to implement the public key cryptography scheme. The two most prominent are integer factorization cryptography (e.g., RSA by Riverst et al. [RSA83]) and elliptic curve cryptography (ECC) by Koblitz [Kob87]. Beside the mathematical principles, the difference between both mechanisms lies within the required key length. Whereas for RSA a key length of 2048 bit and above is suggested[1], 160 bit are sufficient for ECC. Furthermore, ECC is less computational power and memory consuming than RSA. It, therefore, is often used for low power devices or smart cards [GPW+04].

4.1.2. Public Key Infrastructure

Today, the exchange of public keys is done via a Public Key Infrastructure (PKI), e.g., defined by the ITU-T standard X.509 [RFC 2459, RFC 3280]. All approaches have in common that a particular user or node publishes his public key on a key server of

[1]This recommendation from the German Federal Network Agency (Bundesnetzagentur) is valid until the year 2017. The lowest allowed key length for the year 2011 is 1976 bit [Woh11].

some sort from which it can be downloaded by other peers. After that, encrypted and signed messages can be exchanged. The most common approach of PKIs is based on so–called *certificate authorities* (CA). These authorities have two purposes. On one hand, they distribute public keys and on the other, *certify* that a specific public key belongs to a specific legal person (e.g., person, company, etc.). This is done by storing a certificate on the key server instead of the public key alone. The certificate contains the identity of the key owner and the public key. To prove its integrity, the certificate is signed by the CA itself. This approach, however, implies that users have to trust the CA. In case a key pair ever gets lost, it can be revoked by including it in the so–called *Certificate Revocation List* (CRL) where all invalid keys and certificates are kept [RFC 3280].

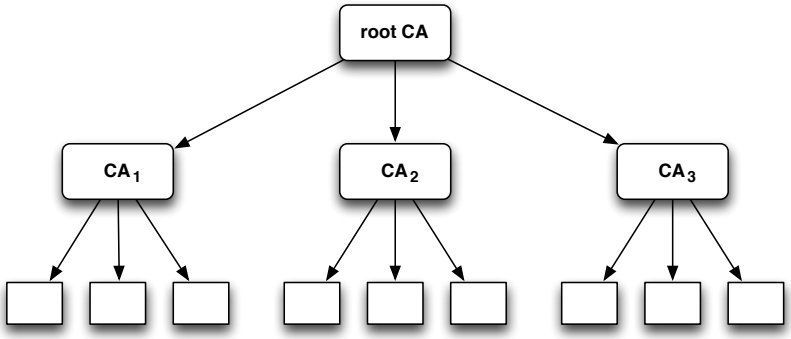


Figure 4.2.: Hierarchical trust model of certificate authorities (CA).

In order to cope with the load of serving and issuing certificates, multiple CAs are used and a trust hierarchy among them is built. Figure 4.2 illustrates such a trust tree. Except from the root level, all CAs are certified by their superordinate CA. The root CA is self–certified. This means that it issued and signed its own certificate. In order to verify an end system's certificate, stored at a leave of the CA tree, the complete certificate chain, beginning at the root, must be verified. By trusting the root and following down a tree branch of CA certificates, therefore, the end system certificate can be verified [RFC 2459, RFC 3280].
This approach of CA–based PKIs, however, has several problematic issues as identified by [ES00, LOP05, LB04]. These problems are in random order:

- As the root CA is self–certified, any end system is able to act as a root CA by issuing a certificate for itself (self-proclaimed authorities). A user, therefore, can not base the decision which root CA to trust on certificates. Other means to establish a trust relation between the CA and a user are required.

- Certificates are usually not signed by the root CA itself. This means a trust chain must be validated in order to verify a certificate. The complete chain, however, is only as strong as its weakest link. By compromising a single CA, an attacker can control the complete subbranch of the CA hierarchy.

- Multiple CAs exist, each having a different policy as to how certificate holders are required to identify themselves towards the CA before a certificate is issued. Methods are, for example, online trust schemes, hand written signatures or physically presenting a legal document to a CA's employee. Yet, no consistency over all CAs exists.

- CAs are usually owned and governed by open corporations. This means that the CAs are subject to financial transactions and can be bought by third parties. This can be a problematic issue for clients in case the CA they subscribed to is sold to a competitor, for example.

Other problematic issues are related to a unified global namespace, certificate complexity, liability and economical problems [ES00, LOP05, LB04].

4.1.3. Threshold Cryptography

Threshold cryptography is a mechanism to share a secret among different parties. The secret, thereby, is split into multiple *shares* and distributed by a dealer to n participants. The distinctive feature is that any group of at least k participants is able to reconstruct the secret. The scheme is perfect if any group of fewer than k participants cannot obtain any information about the secret. Threshold cryptography, therefore, is also often referred to as (k, n)–threshold scheme.
To explain threshold decryption it is helpful to take a look to the principles of Shamir's Secret Sharing [Sha79]. Shamir requires one dealer which is trusted by all participants as he knows every part of the key and has to distribute them to the participants. Shamir's scheme relies on the construction of a polynomial following the pattern $f(x) = s + a_1x + a_2x^2 + ... + a_{t-1}x^{t-1}$ where s represents the secret. Hence, all a_i values can be chosen unrestrictedly by the dealer. Afterwards, the dealer can generate n arbitrary value tuples $(x, f(x))$ where x can be disclosed. As $f(x)$ is a share of the key, however, it must be kept as a secret. At least k value tuples are required to calculate the polynomial by interpolation and consequently also the secret s.

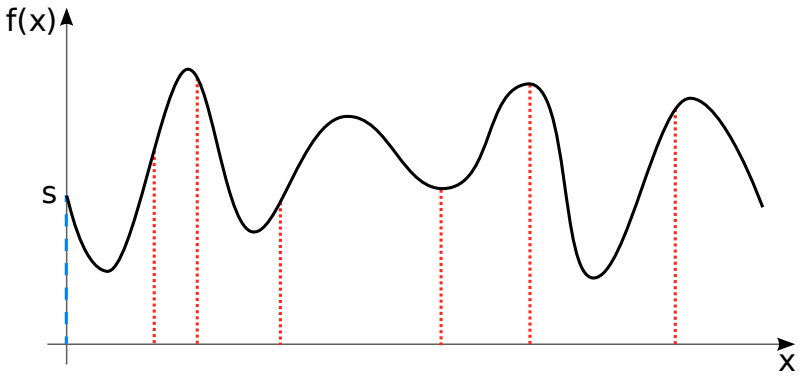


Figure 4.3.: Simplified geometric representation of Shamir's Secret Sharing.

In Figure 4.3 a polynomial is shown where the y–value is marked with s at $x = 0$. This is equivalent to the secret s in Shamir's principle. The red marked points are basic values which are distributed to every single party by the dealer. The x–value of the basic values are public, the corresponding $f(x)$–values are subject to non–disclosure and are known only by the respective party and by the dealer. They represent the shares. If there are enough $(x, f(x))$–pairs available, it is possible to solve the polynomial by interpolation and calculate the secret s accordingly.

A problem of Shamir's (k, n)–secret sharing scheme, however, is its computational complexity. The interpolation to recover the polynomial requires addition, subtraction, multiplication and devision with huge integer values (e.g., $4k\,\mathrm{bit}$ numbers). Other threshold cryptographic schemes based on the same (k, n) principle, therefore, have been proposed. These only require bit–concatenation and logical XOR operations (e.g., Kurihara et al. [KKFT08]).

4.1.4. Internet Protocol Security

Internet Protocol Security (IPsec) is a mechanism to provide security functionality on the network layer [RFC4301]. This extension to the IP header and subsequent encryption of the IP payload was standardized in 1995[2]. It provides end–to–end confidentiality, integrity and authentication between peers. IPsec can seamlessly be integrated into the layered architecture of the Internet as either extension to the IP layer or a so–called bump layer above IP. It, thereby, satisfies the design principals of *layering* and *end–to–end argument* (cf. Sec. 2.1.1).

During the connection setup phase certificates are exchanged between peers in order to mutually authenticate each other. The peer's public key acquisition, however, is not covered by IPsec, and external PKIs are required. This means that although a peer can be identified and authenticated, a link to a real or legal person cannot be established without other means.

Another problematic issue of IPsec is its positioning within the OSI model. While it integrates itself seamlessly into the layered architecture, non–transparent network middle boxes, like firewalls and NATs, are not able to pass through an IPsec stream without further add–ons [RFC3715]. The problem, thereby, is caused by the non–standard conformity of the middle boxes. These, however, have been deployed in enormous numbers before the large scale use of IPsec. In terms of the OSI model, all layers from the transportation layer and above are encrypted when using IPsec and can only be decrypted by the two end–points of a stream. NAT boxes, in order to function properly, require information from the transportation layer (e.g., port numbers). Likewise, firewalls need access to higher layer information to match incoming packets against filter rules.

[2] RFC4301 is the third revision of IPsec published in 2005.

4.2. Novel Security Concepts

Asymmetric cryptography is a key element in today's and future security concepts (e.g., cf. IPsec). Today's PKIs, however, are not a satisfying solution to the problem of how to trustfully distribute public keys [ES00]. Novel mechanisms, therefore, are required to tackle this problem. In the following two different solutions, not requiring a centralized authority, are briefly introduced and their strengths and weaknesses in regard to a NGI architecture are analyzed.

4.2.1. Cryptographic Namespace

In an asymmetric crypto–system, users have a key pair, consisting of a public key and an associated secret private key. PKIs use certificates to tie the public key to a specific user or identifier, respectively. The concept of a *cryptographic namespace* was initially developed by Shamir in 1985 [Sha85]. The idea is to replace certificates by *mathematically* coupling a user's address or identifier (from now on only referred to as identifier) with his public key. One could, for example, simply use the public key as an identifier which, however, is not practical as the key should be quite long for security reasons. Thus, most implementations apply a cryptographic hash function to the public key and use the output (hash) as an identifier[3]. The character of the identifier is thereby depending on the application and could describe a user, a host in a network or an object in a filesystem.
A cryptographic namespace allows for secure communication without the need of any additional infrastructure elements, like a PKI. Figure 4.4 illustrates the necessary steps to acquire the public key of a peer node.

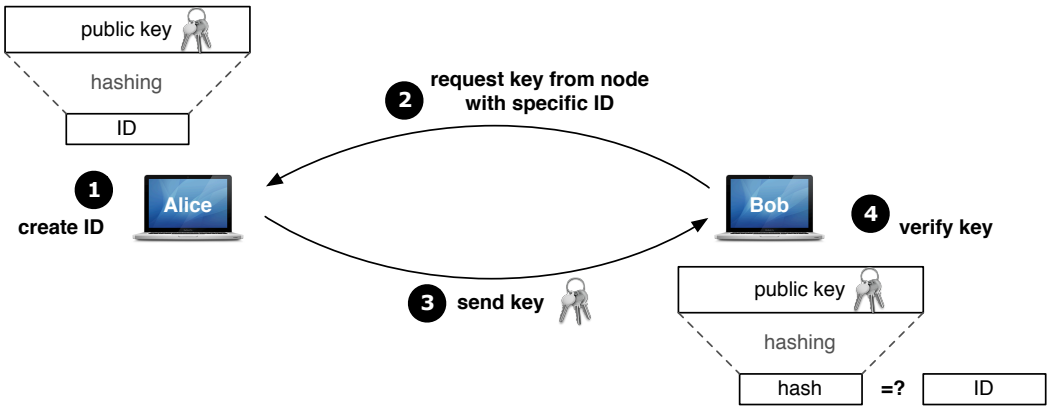


Figure 4.4.: Cryptographic namespace based host verification.

In a first step, Alice needs to hash her public key in order to generate her identifier. The hash value can either be part of the identifier or already serve as the identifier on

[3]Note that Shamir did it the other way round as he used the identifier as public key and thus had to apply some tricks to generate a matching private key.

its own. Afterwards, Alice is able to publish her new identifier (e.g., via a link on a website, DNS, out–of–band). After learning Alice's new identifier, Bob needs to send a key request to Alice in order to obtain her public key. This request is responded to by Alice by sending her public key back to Bob. Note that the communication between Alice and Bob is not required to be encrypted so far. To verify that the host behind the identifier Bob sent the key request message to is really in possession of the corresponding public/private key pair, he needs to hash the public key and compare the result to the identifier.
In the following, three addressing scheme proposals are briefly described which are based on the cryptographic namespace principle.

Host Identity Protocol

In [RFC4423] Moskowitz et al. propose a new protocol layer, the so–called *Host Identity Protocol (HIP)*. As already described in Chapter 3.1.1, HIP introduces a novel namespace on top of IP following the locator/identifier split principle.
HIP renews the current TCP/IP architecture by suggesting a new namespace, the Host Identity namespace. The new namespace consists of Host Identifiers (HI) and fulfills the role of the identifier in the locator/identifier separation paradigm. The HI represents a statistically globally unique name for any system with an IP stack. In [RFC4423], the authors state that "any name that can claim to be statistically globally unique may serve as a Host Identifier". They, however, strongly recommend to use a public key of a public/private key pair as the HI, thereby introducing a cryptographic namespace.
Since different public/private key algorithms use different key lengths, a Host Identity Tag (HIT) is introduced. The HIT is a fixed $128\,\mathrm{bit}$ representation of the Host Identity. It is created by cryptographically hashing the corresponding HI. The HIT, therefore, is a $128\,\mathrm{bit}$ hash of the host's public key. The size of $128\,\mathrm{bit}$ was chosen to match the length of an IPv6 address. In a HIP packet, the HITs identify the sender and recipient of a packet and are meant to look like IPv6 addresses to higher layers.

Accountable Internet Protocol

The *Accountable Internet Protocol (AIP)* by Andersen et al. [ABF+08] introduces a novel two–level hierarchical addressing scheme. The two hierarchies are represented by, first, the network part, the so–called *accountability domains* (ADs), and second, the unique endpoint identifier (EID). The AD is a $160\,\mathrm{bit}$ value representing the domain the node is currently attached to and globally visible in the wide–area routing protocol (such as a BGP prefix). In case the domain is internally organized hierarchically, multiple levels of the AD are allowed. Such an address would look like this: $AD_1 : AD_2 : AD_3 : EID$.
Both $160\,\mathrm{bit}$ values, the AD and EID, are computed by hashing the public key of either the domain or the end–node into a $144\,\mathrm{bit}$ value. The leftmost 8 bits of the address determine the crypto version of the used hash and/or key mechanism. This allows for new algorithms to be used in case an old one is considered to be not secure enough

anymore. The rightmost 8 bits are used to identify the interface of the node or domain in case it is attached via various ones.
The authors argue that a scalability issue may arise from the non–structured address space but state, this might become insignificant in the future as computational power increases over the years.

Cryptographically Generated Addresses

In his work, Aura introduced *Cryptographically Generated Addresses (CGA)* [Aur03] which are based on IPv6. He uses the fact that the $128\,\mathrm{bit}$ long addresses are separated into a subnet prefix and interface identifier. Both parts are $64\,\mathrm{bit}$ long and while the subnet prefix is predetermined by the network topology, the end–node can freely choose the interface identifier part.
Aura uses the $64\,\mathrm{bit}$ interface identifier to incorporate the hash of the node's public key into the IPv6 address. It contains three security parameters (3 leftmost bits) and the "u" and "g" bits which are ignored for this proposal. The remaining 59 bits represent the hash of the public key. The author, however, states that $59\,\mathrm{bit}$ may not be complex enough for an attacker to compute a hash collision. To increase the costs of a brute–force attack, a second hash is used. This hash is generated out of the public key and a modifier which must be chosen to result in a hash with a certain amount of zeros as leftmost bits. The number of zeros is indicated by the three security parameter bits in the IPv6 interface identifier part.

Limitations of Existing Proposals

The cryptographic namespace design aims to provide secure communication without the need of an additional infrastructure element. While there are some applications in which this statement holds true (e.g., local networks), this is not the case for a scenario where the cryptographic namespace is used as a novel addressing scheme in a NGI architecture. In the following, four limitations of the cryptographic namespace are outlined and discussed in regard to a NGI deployment. The work in this subsection has been published in [HL11].

Key Identifier Binding The first shortcoming of the cryptographic namespace design is the tight binding between the address or identifier and the cryptographic public key. As described in Section 4.2.1, the identifier of a node is computationally bound to the public key. This means, whenever either the identifier or the key changes, the other entity changes as well.
Normally, none of these two items is expected to change. There are, however, some cases where it makes sense to replace either the identifier or the public key. Changing the identifier, for example, could be useful for privacy reasons. A user might choose to hold multiple identifiers and swap them periodically to complicate the creation of a tracking record. Using self–certifying identifiers, the user is unable to use the same key pair with the set of identifiers.

Another reason could be the change in server structure or topology. Imagine an online shop or online banking service. In this scenario, trust and authentication should be based on certificates and not on identifiers. In case the shop provider or bank is forced to install new or additional hardware to cope with the load, the identifier would change. By binding the public keys to identifiers, the keys as well have to change, and customers are confronted with a completely new set of identifiers and public keys. This is contrary to still relying on the old certificate and solely being redirected to a new identifier, in case identifiers and public keys are not directly coupled.
The other way around is even more problematic. Today's de–facto standard for asymmetric encryption is RSA with a varying key length of 1024 to $4096\,\mathrm{bit}$. There is no mathematical proof, however, that the RSA algorithm cannot be cracked besides the always possible brute–force method [RSA78]. While we assume RSA to be secure now, this might not be the case at a future point in time. Shor has shown that prime factorization is theoretically possible in polynomial time on quantum computers [Sho99]. Even if RSA might still be considered to be safe in a couple of years, the recommendation for the minimum key length may change. Recently, the National Institute of Standards and Technology (NIST) advised that key lengths of $1024\,\mathrm{bit}$ should not be used after the end of 2010 [BBJ+09]. Whether a completely new algorithm or 'just' a longer key is required, being forced to change the public key immediately results in a different identifier as well. This means that beside all the trouble caused by the swap of algorithm or key length itself, we also have to deal with changing all identifiers worldwide.
One fundamental design goal of the locator/identifier split is the property of the identifier to never change as long as the user does not request a new one. The locator, instead, should be allowed to change frequently reflecting the different access points towards the topology while roaming. By computationally binding the public key to the identifier, however, this design goal is violated.

Lack of Key Revocation Another crucial point is that—by design—the cryptographic namespace does not allow for a key revocation mechanism. In current PKIs, trusted Certification Authorities (CA) approve the relationship between an identifier and a public key by signing a certificate [RFC 3280]. Revocation of a key (pair) is required in case a private key was destroyed, became public, was lost, compromised or a certificate was issued improperly. The CAs hold so–called Certification Revocation Lists (CRLs) which can either be checked by the clients on demand or are periodically published to them. This enables a participant to detect compromised or non–trustworthy parties.
As the previous section has outlined, the cryptographic namespace couples the identifier even tighter to its public key, meaning that any revocation scheme would revoke the identifier simultaneously and therefore interrupt communication until the partners have exchanged the new identifier. Consequently, none of the presented approaches integrates a revocation mechanism leading to severe security issues.
If one client looses his private key, no reliable communication is possible under the same identifier anymore. It might, however, be acceptable to change the identifier

under certain circumstances, but a more critical problem rises if the key is not only lost but also found or stolen by a (malicious) third party. The private key enables the attacker not only to decrypt messages dedicated to the righteous owner of the key pair, but also to more or less automatically spoof his identifier and act as if he was the legitimate addressee. This is possible until the previous communication partners are notified that a new identifier has been issued. Moreover, the identifier update has to be done *out of band* as another secure channel is required.
Furthermore, key pairs should have a limited lifetime and are to be renewed regularly. Reasons, therefore, can be cryptographic algorithms that have turned out to be insecure, increased computational power requiring longer keys[4], or simply to make a successful brute–force attack even more unlikely [BDR+96]. Currently, digital certificates have limited validity periods. Their lifetimes differ depending on their purpose, the issuing CA's guidelines, reputation of the client and other criteria. For example, the CA/Browser Forum which many leading CAs are member of, recommends to renew TLS/SSL certificates every 12 months [The10]. This would again result in a change of the host's identifier.

Additional Trust Entity Required The cryptographic namespace has been introduced to support secure (encrypted) communication based solely on the identifiers of the clients. No additional mapping between public key and identifier is required. Two nodes contact each other by their IDs, and they exchange their public keys which both parties can verify to match the IDs by hashing.
A problem arises in case the identifiers have not been transmitted over a secure channel in advance, e.g., if a client gets a faked identifier of his communication partner from an untrusted source. This is because there is no way to prove the association between the identifier, public key and the entity claiming to be its owner. This means, HIP and other NGI protocols will nonetheless require an (PKI–like) additional party that maps identifiers and keys to specific legal persons or entities. Today's mapping service, DNS, has suffered from recent attacks illustrating the need for a secure and trusted mapping service. Consequently, the migration to DNSSEC [RFC4033] is a step towards more security.
It should be mentioned that a cryptographic namespace based design can, from a functional point of view, work without trusted entities. But in order to provide full communication security (authentication and integrity), future concepts cannot get along without a trusted ID mapping service.

Key Guessing The last reason seems to be statistically insignificant at first. It, however, becomes an important issue as more identifiers out of the flat identifier space are used. This is relevant for any NGI architecture which uses globally unique identifiers. It is possible for an attacker to generate a random triple of a public key, a private key and an identifier. The attacker needs to generate a public/private key pair and hash the public key. As a result, it holds a valid private key for a specific identifier. In a

[4] As a rule of thumb, the lifetime of a key should be much shorter than the estimated time it would take to crack the key through brute–force.

last step, the attacker needs to check the mapping system whether this identifier is already in use or not. In case the identifier is already in use by a node, the attacker can either compromise any communication with that node or start new communication flows pretending to be that node.
While the first attack—compromising a communication—poses a serious problem for that random node, the second one questions any authentication supported by a cryptographic namespace based concept. Any communication or message fails verification and becomes deniable. This is because a signee can always argue that it was not him who electronically signed the document.
It is possible to argue that it is very unlikely to hit a random address within a $128\,\mathrm{bit}$ identifier space (e.g., used by HIP) by random key generation. This might be true for today's number of users currently roughly being around 4 billion nodes [Hus10]. In that case, it would take an attacker several years until finding an identifier which is already taken. First of all, he needs to generate 2^{96} identifiers in average which is an enormously huge number. Generating and hashing a $2048\,\mathrm{bit}$ RSA public key takes $270\,\mathrm{ms}$ on an Intel Xeon $2.5\,\mathrm{GHz}$ machine. This results in 7×10^{20} years for a single machine, neglecting that it takes time to query the mapping system (this should take less than $200\,\mathrm{ms}$ and can happen in parallel to key generation).
The interesting figure in this equation is the number of used identifiers. This number might increase significantly in the future. Proposals for Next Generation Internet architectures discuss to not only use identifiers for end nodes, but also for persons, content and services.

4.2.2. Public Key Distribution in Wireless Ad–Hoc Networks

Public key distribution in wireless ad–hoc networks is a challenging task. Ad–hoc networks are required to work without a centralized infrastructure and cannot rely on steady connections between nodes [MM04]. Depending on the use case of the ad–hoc network, it might be an isolated network without any connectivity to the Internet. Traditional resources (e.g., X.509 following PKIs), therefore, cannot be utilized to distribute public keys.
Wireless ad–hoc networks are formed as nodes discover other ones within their own radio range. Depending on the underlying protocol, they send discover messages and maintain connections to their direct neighbors. The ad–hoc network is extended as each node participates in routing by forwarding data for other nodes. The determination of which nodes forward data is made dynamically as well, based on the network connectivity [MM04].
Another property of wireless ad–hoc networks is that nodes might loose connectivity due to mobility or being switched off. This means that a wireless ad–hoc network rarely is in a static state and normally experiences a churn rate among its participating nodes. It is even possible that a single ad–hoc network is split into several subnetworks without connectivity among the subparts.
Due to a wireless ad–hoc network's attributes a traditional PKI following the X.509 standard [RFC 2459, RFC 3280] is not applicable. Another solution to distribute public

keys among a known group of nodes is to use threshold cryptography introduced in Section 4.1.3. Instead of providing a centralized PKI each node participates in the key distribution. Nodes, thereby, store public key certificates of other ones. To this extent, multiple copies of the same key certificate might exist within the network. Contrary to today's PKI, not a centralized instance issues the key certificates. Instead, nodes collaborate based on a trust model (e.g., [BSD10]), and collectively issue and sign public key certificates for other nodes. This means that a master key pair exists which is used to sign and verify the certificates issued for nodes' public keys. The public key of that master key pair is distributed among all participants (e.g., upon connecting to the network). The private key, however, is split into shares, and these are distributed to various nodes. Preferably these nodes proved to be rather stable and trustful in the past.
A new node joining the ad–hoc network is required to request a key certificate for its public key and upload the key as well as the certificate to other nodes. The request to issue the certificate will only be granted in case enough nodes trust the new one. This means a specific threshold of nodes must be met. In terms of the threshold cryptography scheme, at least k out of n nodes must trust the new node. In case enough trusting nodes can be found, a certificate is collectively issued, using the shares of the k nodes. The verification of the certificate is done using the free circulating public master key. Figure 4.5 illustrates a sample ad–hoc topology.

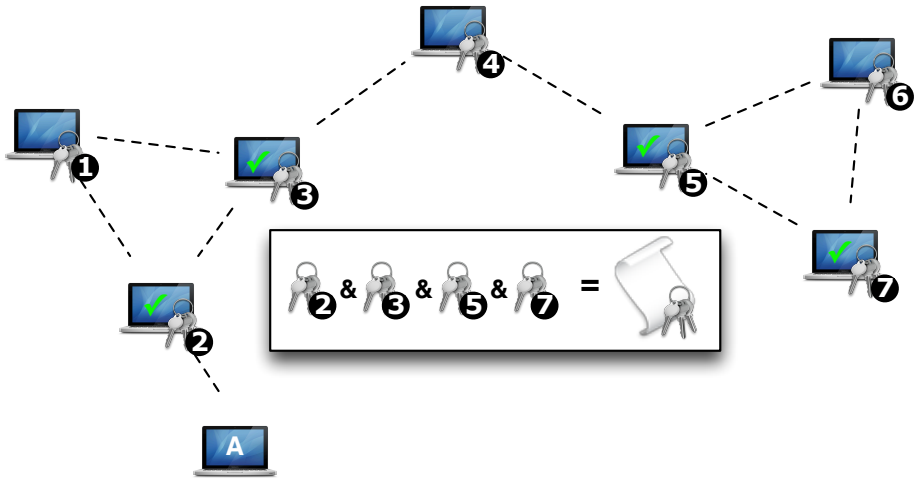


Figure 4.5.: Public key distribution in wireless ad–hoc networks.

In this example, node A wants to join the ad–hoc network and distribute its public key among the other nodes. Nodes 2, 3, 5 and 7 trust node A and collectively sign node A's public key certificate.
Implementations of this distributed certificate authority scheme are, e.g., [KZL+01, Kon11]. They mainly differ in the trust models used in the schemes. Further, the second one groups nodes into clusters in order to increase the availability of the shares.

Limitations for a NGI scenario

The introduced mechanism to distribute public keys based on threshold cryptography and trust models is an interesting approach for wireless ad–hoc networks. The parameters of ad–hoc networks and a NGI architecture, however, drastically differ, and the mechanisms cannot be applied one–to–one to the latter. Even though the threshold scheme would theoretically be applicable to billions of nodes, the trust models are not due to political and scalability reasons. It is not suitable that half the world must trust a single node in order to issue a key certificate. The consequence would be to build trust domains, reducing the amount of required trusting nodes. This, however, would raise the question about how the individual domains could trust each other, requiring an hierarchical sort of structure among the domains. The outcome would be similar to today's X.509 PKI architecture and, therefore, suffer from the same problematic issues (cf. [ES00]).
In the remainder of this chapter the HiiMap security framework is introduced. This framework is also based on threshold cryptography for public key distribution but does not require a master key to be shared.

4.3. HiiMap Security Framework

The HiiMap security framework is an integral element of the HiiMap NGI architecture. It provides the foundation for any security–related mechanism or application above the network layer. The framework is designed as a trust anchor to a chain of security mechanisms built on top of it. In that way it is open to future applications and protocols while at the same time securing the HiiMap protocol described in the previous chapter.
The framework is built around asymmetric cryptography, and its core element is a distributed PKI. Instead of providing an additional infrastructure, however, the PKI is integrated into the HiiMap mapping system. In that way resources can be shared among the functionalities, and overhead and maintenance can be kept lower compared to operating separate services.
The work in this chapter has been partly published in [HEP+11, FH10, HF11, Han12, LH10].

4.3.1. Public Key Infrastructure

In the HiiMap architecture each mapping entry consists of an identifier as the primary key and a set of locators by which the end system currently can be reached (c.f. Sec. 3.3). Further, a timestamp of the last update and a flag, indicating whether the location update was cryptographically signed by the end system or not, are stored. Normally any location update must be signed to prevent identifier hijacking [Han12]. Some devices, however, might not have enough computational power to handle long asymmetric cryptographic operations. To integrate these devices, the owner can request to send unsigned location updates. An unsigned location update is signaled by

a set flag, and peers can decide whether they want to start a communication with such a device or not.
To combine the PKI with the mapping system, the public key of each node must be additionally stored for each mapping system. The result is that no additional protocol or infrastructure has to be provided for querying and storing public keys. As the public key is a very static value and not expected to change frequently, the additional burden for the mapping system is limited, and the public key databases can be optimized for frequent read requests—contrary to frequent read and write requests for the locators.

Distributed Key Storage

By storing the public key at only one administrative domain (region) in the mapping system, the user heavily depends on the trustworthiness of that particular region. In case the mapping service provider collaborates with an attacker, it could send a wrong or manipulated public key to the client. Therefore, any security functionality based on the public/private key principle would be rendered useless. Even worse, the client considers the connection to be secure while in fact talking directly to the attacker (i.e., man–in–the–middle attack).
To address this issue, the public key is divided into n shares and distributed to various independent regions. To reconstruct the public key only k out of n shares are required, using the threshold cryptography scheme, as summarized in Section 4.1.3. Figure 4.6 illustrates an example with $n = 4$ and $k = 3$. The client is able to reconstruct the public key from a subset of the key shares. This means that not all shares must be obtained. Untrustworthy or topology wise far away regions can be omitted, and the client can choose which regions to query for the key shares. In that way, the key is not stored at only a single region, and at least k regions must be queried to reconstruct the key.
In comparison to distributing full key copies to several regions, this approach has an important benefit. In case a region stores the complete key and cooperates with an attacker, it could swap the key with another one. This is obviously a problem in case the requester relies only on a single region. Even if he queries multiple regions, the result will be a set of non–matching keys. This means that he needs to base his trust in the key on the majority principle by selecting one of them. Some requesters might choose to cancel the connection setup because of the irregularities of the obtained key set.

Determining the Storage Location

Another critical aspect of this approach is the mechanism how a client is able to determine the storage location of the shares. Being required to retrieve this information from the responsible region would allow a malicious region to manipulate this list. Entries in the storage location list could be modified to point to allied regions. As a result, a malicious key could be divided into shares and distributed to the allies. Together with a malicious locator information from the responsible region, a client would again be vulnerable to man–in–the–middle attacks. The goal for the mechanism, therefore, must be to determine the storage locations without any additional

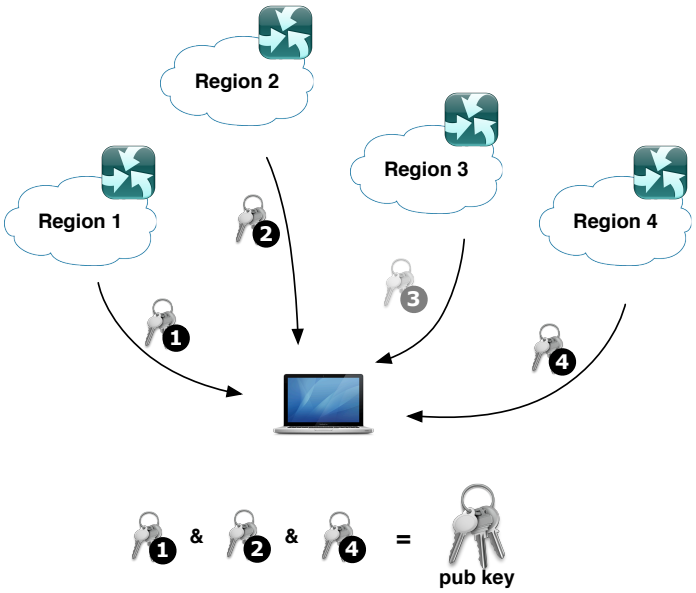


Figure 4.6.: Reconstructing the shared public key.

information from the responsible region.
The algorithm to retrieve the storage location requires two input parameters. These are the identifier the shares are associated with and a transformation directive from the Global Authority (GA). In the HiiMap architecture up to 256 regions exist. This is represented by the $8\,\mathrm{bit}$ Regional Prefix (RP) address space which plays a crucial role in the algorithm. The algorithm is formalized as follows:

1. The identifier is hashed into a $n \cdot 8\,\mathrm{bit}$ value ($\mathcal{H}$). For this operation, a cryptographic hash function, like SHA-1, is suggested.

2. $\mathcal{H}$ is then divided into n equal sized pieces of $8\,\mathrm{bit}$, resulting in a group $\mathbb{K} = \{k_1, k_2, ..., k_n\}$.

3. The group $\mathbb{K}$ is then transformed using a directive downloaded from the GA ($\mathbb{K} \rightarrow \mathbb{K}'$).

4. The resulting group $\mathbb{K}' = \{k'_1, k'_2, ..., k'_n\}$ reflects the storage locations of the shares wherein each k'_i represents a RP.

5. To ensure that a region only stores the maximum of one share for a given identifier, it must be guaranteed that $k'_i \neq k'_j$ is met[5]. To satisfy this requirement, k'_j of a found $k'_i = k'_j$ pair is increased by one. This process is repeated until $k'_i \neq k'_j$ is valid for the whole group $\mathbb{K}'$.

[5]This could be the case as the k_i pieces result from a truncation and transformation operation. The hash value $\mathcal{H} = 10011001\,10011001$ for $n = 2$, for example, would result in $k_1 = k_2 = 10011001$. The transformation might as well map two different k values to the same k' value.

Following these steps, a client is able to determine the storage locations of the key shares associated with a specific identifier. The same algorithm is applied for up- and downloading the shares. The transformation directive in step 3 is necessary as not all 256 regions might exist. The RP address space might not be full, and the directive ensures that shares are only stored at existing regions. Furthermore, the transformation can be used to implement statistical load balancing between regions. Larger regions might be capable of storing more shares (from different identifiers) than smaller ones.

Figure 4.7 illustrates the algorithm for $n = 4$. The identifier is hashed into a $32\,\mathrm{bit}$ value ($\mathcal{H}: 4 \cdot 8\,\mathrm{bit}$) and afterwards truncated into four pieces. The group of four pieces $\mathbb{K}$ is transformed into the group $\mathbb{K}'$ following the directive downloaded from the GA. As the directive is expected to change rarely, it can be cached at the client. To ensure each share is stored at a different region, the requirement from step 5 is enforced wherein the second k' value from a matching pair is increased by one until no other k' exists with the same value. Finally, the key shares can be up- or downloaded from the regions identified by the k' values.

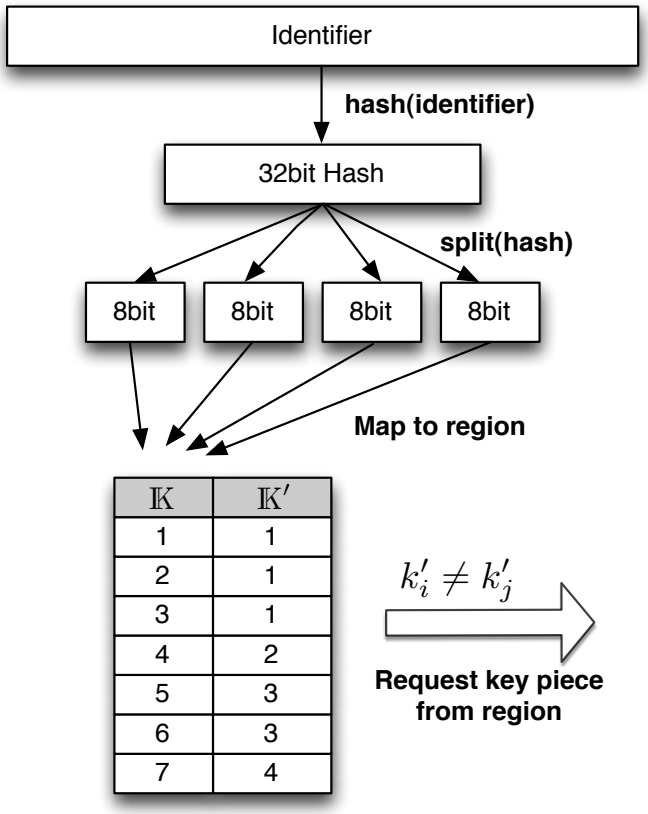


Figure 4.7.: Determining key share storage locations.

Retrieving a Public Key

To retrieve a public key from the HiiMap distributed PKI, several information is required. This information can be obtained from various entities in the mapping architecture. Table 4.1 lists an overview of the storage locations.

Entity	**Stored information**
Global authority	Transformation directive ID $\rightarrow$ RP
Responsible region	ID $\rightarrow$ locator ID $\rightarrow$ n, k ID $\rightarrow$ key length ID $\rightarrow$ key algorithm (e.g., ssh-ecc, ssh-rsa) ID $\rightarrow$ key timestamp
Region	ID $\rightarrow$ share
Client	cached(transformation directive)

Table 4.1.: Stored information on each entity in the HiiMap PKI structure.

In a first step, the transformation directive is downloaded from the GA. This directive is expected to change rarely and can be cached at the client site. Therefore, it can be omitted for subsequent public key queries. The second task is to determine the storage location of the shares. This is done by executing the previously described algorithm. After having determined the storage locations, at least k shares must be downloaded. Ideally the topology wise closest k regions are queried. Another strategy would be to query regions which previously provided reliable information to the client. With k shares the secret s (i.e., the public key) can be reconstructed. To be able to validate the public key, s does not only consist of the public key but also a checksum. In that way, a client is able to verify the correctness of the public key and can conclude that none of the shares was modified. In case the checksum does not match the public key, the client is aware that at least one of the shares is either corrupted or was intentionally modified. He, therefore, must request additional shares from the n regions he did not obtain a share from before. The process of additional share downloading, key reconstruction and verification is repeated until either the checksum matches the public key or all n shares have been obtained and any possible k out of n combination was used in the attempt to reconstruct the key. Please note, the case in which no key reconstruction is possible is very rare and expected to never occur. Not being able to reconstruct the key means that at least $n - (k + 1)$ shares are either corrupted or intentionally modified.
An addition to this scheme—not further investigated in this thesis—could be that a client determines which downloaded share was corrupted. This could be done by downloading $k + 1$ shares in case the initial reconstruction fails and rotate the individual shares until a reconstruction is possible. The region from which the corrupted share was downloaded could then be reported to the GA. In case the GA receives multiple complaints about a specific region, it could take measures to deal with the

problem. Another approach would be that the GA itself conducts periodic random tests to monitor all regions and the operation of the overall system. This again could be indicated to clients in order to give a hint on each region's credibility.

4.3.2. Client Key Management

Beside the distribution of the public key, the protection of the private one plays an equally crucial role in asymmetric cryptography. The private key must be kept secret at all times and must only be accessible by the rightful key owner. A disclosed private key breaks the security chain of asymmetric cryptography and any security functionality built on top. It is, therefore, the user's task to prevent unauthorized access to the private key. The problem arises as soon as unskilled or untrained users are to use the cryptographic scheme. In today's systems the handling of the private key is rather unintuitive, and users are often not aware of risks and ways a private key can be disclosed to third parties. A security framework in a NGI scenario, however, must be easily usable and not depend on the technical knowledge of the users.
The security framework of the HiiMap architecture, therefore, relies on cryptographic smart cards (from here on only referred to as *smart card*) to store a user's private key. In terms of usability, the abstract entity of the private key is coupled with the physical entity smart card, hence, providing the user with a more natural and familiar security token. The functionality of the smart card which is subsequently introduced could also be fulfilled by a Trusted Platform Module (TPM). As this module is usually soldered to the main board of an electronic device, it lacks the natural *key like* feature of the smart card and further provides a less flexible solution.

Smart Card

The smart card stores the values listed in Table 4.2. The values in the left column, thereby, can be retrieved from the card. The right column value is not accessible from outside the card's security architecture. Even the owner of the smart card has no access to it.

Accessible information	**Non–accessible information**
Public key Unique Identifier (UID) Card ID	Private key

Table 4.2.: Information stored on the smart card.

Once the card is produced, a unique card ID as well as the UID are transferred to the smart card in the initialization process. Afterwards, the card's method to generate a private/public key pair is triggered. After the key generation a fuse bit is automatically set, disabling the method. This bit is only writeable once. This has the benefit that the smart card generates the key pair itself, and it does not have to be transferred

onto the card. The latter case would mean that the private key once was outside the security environment of the smart card—a possibility to copy the key. The fuse bit is set in order to ensure that no new key pair is generated by the card. The initial pair should be valid for the remaining lifetime of the smart card.
The term *smart card* in this context refers to the generic device. The exact implementation is not specified for the HiiMap security framework, and multiple versions might coexist. The variations might range from today's mobile phone SIM cards via Mobile Security Cards compatible with the microSD standard up to Internet capable smart card tokens [Dev11]. Each version might be suitable for different use cases. The SIM or Mobile Security Card, for example, could be used for personal devices such as smart phones whereas the smart card token in the form factor of a USB flash drive might be preferred for multiuser terminal devices. In the last case, the smart card could be used to securely access a personal context and content stored in a cloud from a generic non–personalized device. These use cases are based on the introduced security framework but not further detailed in this thesis.
Although the physical aspect of the smart card adds to the usability of the security framework, it also represents a drawback as the card can be lost or stolen. To prevent unauthorized access to the smart card's security functionality (e.g., signing of messages), the card must be protected with a PIN/PUK mechanism[6]. Furthermore, a lost or stolen card must be reported to the user's provider. The provider issues a new card and the old one is marked in the mapping system as out of use. Any further `USR_LOCATION_UPDATE` sent from this card will be ignored. For this functionality, each smart card requires a unique identification number. The *card ID* of the smart card in use for a specific UID is stored in the mapping system. Each incoming message attempting to alter a mapping entry is, additionally to the cryptographic signature, checked whether it contains the correct smart card ID or not.

Bootstrap

An important aspect of the client key management is the initial bootstrap, i.e., the first time a new end system, respectively a new UID, connects to the network. The challenge is how the public key associated with the UID is entered and updated in the mapping system. Sending the public key without any protection along with the first `USR_LOCATION_UPDATE` message would allow an attacker to switch the key. The mechanism of splitting the public key into shares and distributing them over several regions would be rendered useless. A bootstrap mechanism, therefore, is required which safely registers the UID and its associated public key with the mapping system. Figure 4.8 illustrates the necessary steps specified by the HiiMap security framework. Before being able to connect to the network, a user is required to request a new UID for his end node. To do so, he authorizes his provider to request a new smart card from the GA. The GA's task is to provide the user with a smart card which is initialized with an unused UID and on–card generated key pair. The manufacturing, initializing

[6]This mechanism can be considered as statistically safe. Assuming an implementation with a four digit PIN and three attempts, the likelihood to find the correct PIN is $\frac{1}{3333}$ (Digit range: 0-9, $P \approx \frac{3}{10 \cdot 10 \cdot 10 \cdot 10}$).

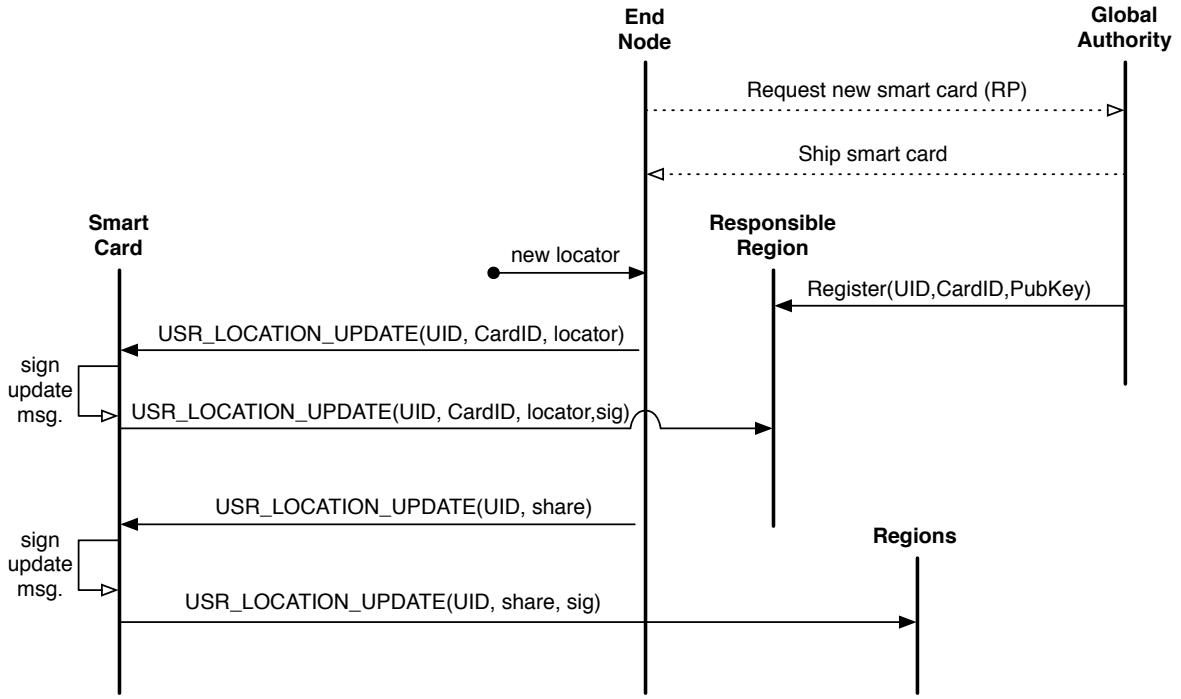


Figure 4.8.: Initial end node bootstrap process in the HiiMap security framework.

and shipping of a smart card is logically in the responsibility of the GA. It, however, might choose to delegate these tasks to trusted partners in order to cope with the load. It is also conceivable that the GA commissions the provider to produce and initialize a certain amount of smart cards for future usage. These smart cards are then shipped to customers upon request. The process of requesting, initializing and shipping the smart card serves two important aspects.

- With requesting a new UID/smart card, the user is required to identify himself. This is similar to requesting a new SIM card for mobile phones. By providing some sort of identification during the order process, the key pair on the smart card is directly linked to a legal person. Thus, the problem of *additional trust entity required* of the cryptographic namespace (cf. Sec. 4.2.1) and the concern raised by Ellison et al. [ES00] about inconsistent identification process of today's PKIs are overcome.
- Before shipping the smart card to the user, the GA (or one of its delegates) transfers the public key into the mapping system. The GA guarantees that a secure channel is used (e.g., integer and encrypted connection between GA and the responsible region). In this way, the key is not initially transferred directly from the unauthenticated user to the mapping system, eliminating the possibility of a key swap by an attacker.

As the registration of the UID/key pair coupled with the requirement for legal identification may raise privacy concerns, the usage of unregistered UIDs and keys is also

possible. Not all users wish to link their network identity with their real one. Unregistered network participants, however, are marked as such and might be excluded from certain services (e.g., e–commerce applications).
After the smart card is shipped to the user, the GA registers the new UID with the responsible region for this UID. The RP of the responsible region the user wants to join was included in the request of the smart card. Along with the UID, the user's legal information, the smart card ID (from here on only referred to as card ID) and the public key are transferred. These values are stored by the responsible region although not all of them are accessible to clients of the mapping system. The public key, for instance, is stored for the region's own usage. It requires the key to verify signatures of location updates concerning that specific UID. In that way, the public key does not have to be reconstructed from shares for each signature verification process by the responsible region.
Once the user received the smart card, it can be used on any Internet–capable device. Upon connecting to the network, the end node receives a new locator from its network attachment point. The end node then constructs a `USR_LOCATION_UPDATE` message containing the locator. Instead of sending this message to the mapping system, it is transferred to the smart card. The smart card internally adds its own card ID, the UID and cryptographically signs the update message. This signed message is then sent to the responsible region. Not shown in Figure 4.8, this message also contains the `OPTION_FIELD` (cf. App. A). This option field contains the configuration for the threshold cryptography scheme. The configuration determines how many shares (n) exist and how many are needed to reconstruct the key (k). In addition, the length and used algorithm for the public key are sent along in the option field.
The region first verifies the signature and checks whether the included card ID is listed as the current valid one. Afterwards, the mapping entry is updated. Subsequent to the initial `USR_LOCATION_UPDATE` message, the end node constructs the public key shares, and after being signed by the smart card these shares are sent to the regions identified by the algorithm described in Section 4.3.1.
The regions storing shares of the public key query the responsible region for the public key associated with the UID the update messages are from. In that way they are able to verify the signature in the update message and the integrity of the share.

Location Updates

Location updates are straight forward, and the end system's part is already depicted in Figure 4.8. Whenever a new locator is assigned to an end system, it needs to notify the mapping system of the change. The end system constructs the `USR_LOCATON_UPDATE` message containing its own UID, the card ID, the new locator and (not mandatory) some options. This message is cryptographically signed by the smart card and then sent to the end system's responsible region.
A different approach would be to issue location update messages by the access network instead of the end system. This mechanism, however, has two drawbacks. Firstly, it increases the computational burden of the access routers and, secondly, an access network cooperating with an attacker could send forged update messages.

Upon receiving the update, the responsible region must verify the message. This process is depicted in the flow chart in Figure 4.9. The first task is to fetch the corresponding mapping entry for the UID listed in the message from the own database. In case no such entry is found, an ERROR_MSG is returned to the end system. The next step is to verify the signature enclosed in the message. In case the signature check fails, the update process is aborted and again an error message returned. Afterwards, the card ID included in the message is checked against the stored card ID in the mapping entry. In that way the mapping system is able to verify that the update request was sent from a legit smart card and not from a stolen one. In case all checks succeed, the mapping entry is altered according to the information of the USR_LOCATON_UPDATE message.

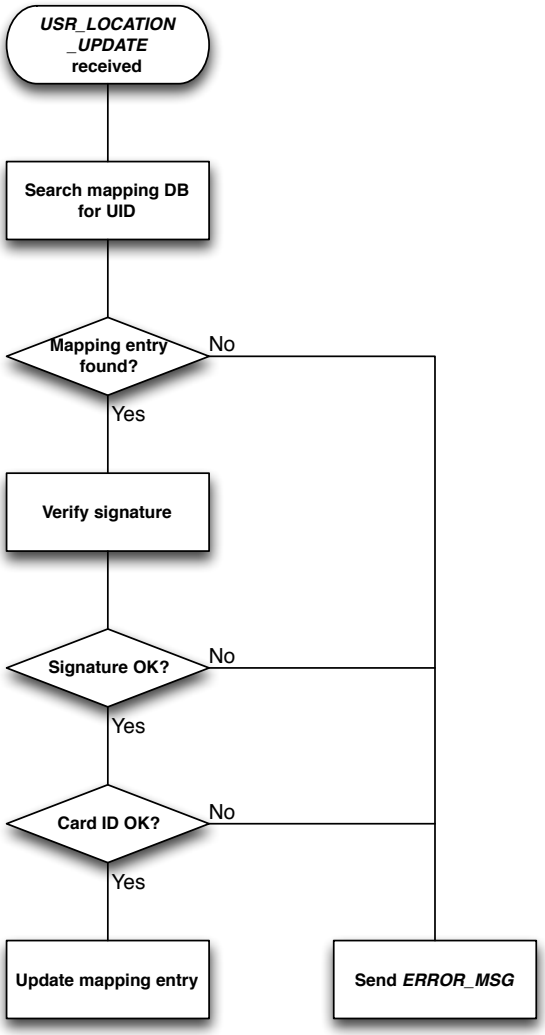


Figure 4.9.: Flow chart for USR_LOCATION_UPDATE message processing.

Unauthenticated Location Updates

Some Internet–capable devices might not be able to compute crypto challenges (e.g., sensors) or it might not be possible to interface them with a smart card (e.g., physically remote or inaccessible devices like satellites). The security framework, therefore, must provide a mechanism to connect these devices without compromising the entire security architecture. Devices which are required to send unauthenticated `USR_LOCATION_UPDATE` messages are marked as such and their mapping entry contains a flag, signaling peers that the last location update might have come from an untrusted source. It is the peer's decision to decide whether to trust the mapping entry or not. A researcher querying sensor information, for example, might choose to trust the unauthenticated location update from his device whereas a bank probably requires its online customers to only connect with authenticated devices.
In order to disable authentication, the following process as depicted in Figure 4.10 is required.

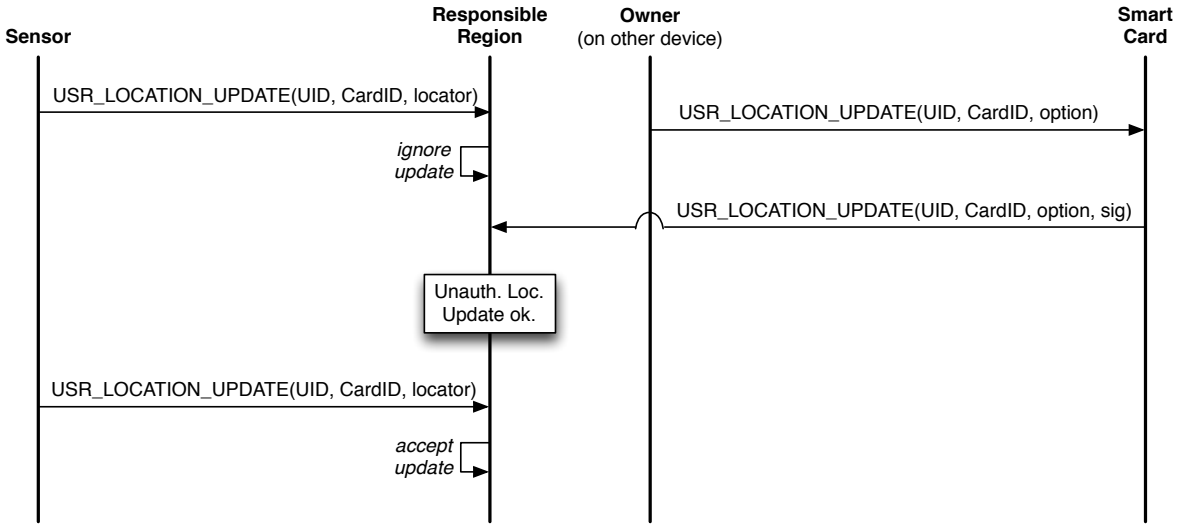


Figure 4.10.: Enabling unauthenticated `USR_LOCATION_UPDATE` messages.

The owner of the device requests a UID and smart card as usual—this might be in addition to his own personal smart card and UID. He uses any Internet–capable device which is equipped with a smart card interface and starts the regular bootstrap process. Afterwards he sends another `USR_LOCATION_UPDATE` message which contains the request to disable authentication in its options field. This message is signed by the smart card as usual, thus, authenticating the request for this specific UID. Upon receiving the message, the mapping system sets the flag in the corresponding mapping entry. After the user configured the device (the one which is not capable of interfacing a smart card) to use the correct UID, the device can send `USR_LOCATION_UPDATE`s which do not include a signature. The smart card remains with the owner of the device. To reenable authentication for that specific UID, the user uses the smart card to send another (signed!) `USR_LOCATION_UPDATE` message, requesting to clear the flag in the mapping entry.

Key Revocation

Today's PKI provides a means of key revocation. This means that a user is able to declare a compromised or lost key pair. Before using a peer's public key to either verify a signature or encrypt a message for the recipient, the user is able to check whether the key is still valid or not. A similar mechanism is implemented in the HiiMap security framework. Only the key which is stored in the mapping system is the current valid one for a specific UID. This means, in case the downloaded and a cached public key differ, the cached one is most likely revoked and not valid any more. The challenge is how to revoke a public key in case the smart card was lost or stolen. Without the card, the user is not able to sign the `USR_LOCATION_UPDATE` messages which are required to upload new shares into the mapping system. Furthermore, the user cannot authenticate himself to his provider in order to report the loss of his card. To circumvent this problem, two possibilities exist:

- As with today's PKI, the first solution is based on revocation certificates. These certificates state that the legitimate owner of the smart card—and, therefore, also the UID and key pair—wants to revoke the key and delete the card ID from the mapping entry. This revocation certificate must be signed with the smart card's private key in order for the mapping system to be able to verify its legitimacy. As the certificate cannot be signed after the card is lost or stolen, it is advised to sign the certificate shortly after receiving the smart card and storing it in a safe place. In that way, the signed certificate can be sent to the responsible region and a new smart card requested from the user's provider.

- The second way is similar to reporting a lost credit card. The user is required to phone a call center of the responsible region and authenticate himself. This could be done by providing parts of a secret pass code, personal details about the user (i.e., street address, mothers maiden name) or else. After successfully identifying himself, the user is able to report the loss or theft of his smart card. Afterwards the process is the same as with the revocation certificate. The old card ID is deleted from the corresponding UID's mapping entry, and any further messages sent from the old smart card are ignored by the mapping system. A new smart card is issued, and the new card ID and public key are transferred into the mapping system (see *Bootstrap*). This second mechanism is also the fallback solution—and overrules the revocation certificate—in case the certificate itself is lost or disclosed to third parties.

An end system is required to query the mapping system in order to check whether the cached public key of a peer is still valid or not. As downloading and reconstructing the public key from shares each time a key is to be used is cumbersome, a timestamp is included in each mapping entry. This key timestamp (cf. Tab. 4.1) is set to the last time the public key was uploaded. In that way only the cached timestamp and the one stored in the mapping system must be compared to determine whether the key is still valid or not.

4.3.3. Attack Scenarios

The HiiMap security framework builds the foundation for higher layer security functionality (e.g., encryption, authentication, etc.). The framework itself, however, must be resistant against attacks as well. The following feasible attacks concerning the framework and the HiiMap architecture are considered and their countermeasures described.

Private Key Attacks

To undermine the security framework based on asymmetric cryptography, an attacker's main goal is to compromise the private key of its target. Controlling the private key means that the attacker is able to decrypt private messages, sign documents in the name of the target as well as authenticating itself as the target towards third parties. Employing man–in–the–middle tactics the target might even not be aware of the attack. Gaining control over the private key, thereby, can be achieved in two ways. The first and *direct* way is to gain possession of a target's actual private key. The other method is to manipulate the public key stored in the PKI. This *indirect* method aims at exchanging the public key to match a private one the attacker is in possession of. Peers of the target would use the altered public key to encrypt and verify messages from and to the target. From here on, the various keys are distinguished as follows:

- $K_{\mathrm{D}}^{\mathrm{t}}$: The original private key of the target.
- $K_{\mathrm{E}}^{\mathrm{t}}$: The original public key of the target.
- $K_{\mathrm{D}}^{\mathrm{a}}$: A private key controlled by the attacker.
- $K_{\mathrm{E}}^{\mathrm{a}}$: A public key matching the private key controlled by the attacker.

Direct Private Key Control To control a target's private key ($K_{\mathrm{E}}^{\mathrm{t}}$), the attacker requires access to it by either copying or stealing it. As a result, the attacker is able to compromise any secrets or authentication of the target until the target generates a new key pair[7]. In the HiiMap security framework, the cryptographic smart card ensures that no direct access to the public key is possible. $K_{\mathrm{E}}^{\mathrm{t}}$ remains on the smart card and cannot be copied. Any cryptographic operation involving the public key is computed by the smart card's own processor. The hardware of the smart card ensures that the private key can not be copied even if the card is stolen, disassembled and challenged in a laboratory (e.g., side channel attack, power analysis, etching). For this architecture it is assumed that smart cards exist which do not disclose the private key even if manipulated physically or chemically [RE03].
In case the attacker is able to steal the target's smart card, the card is protected by the PIN/PUK mechanism. Depending on the realization, an attacker has, e.g., three tries to guess the four digit PIN code. The statistical chance to find the right combination is

[7] An attacker probably would ensure that it gains access to the newly generated key pair as well. This, however, is not further considered as the goal is to initially deny access to the private key.

$P \approx \frac{1}{3333}$ (cf. Sec. 4.3.2). After three failed attempts to enter the correct PIN, the card is disabled, and the correct personal unlocking code (PUK) is required. This multi digit code may also only be entered up to three times. Failing to provide the correct code initiates a self–destruct procedure of the card, including the deletion of the private key.
Before stealing the card an attacker could try to gain knowledge of the PIN, e.g., with the help of a key logger on the Internet capable device interfacing the smart card. A countermeasure could be to equip the card with a keyboard to enter the PIN directly or a display which provides the user with a random challenge. This challenge would be based on the PIN but does not require all digits. In that way a key logger would not be able to record the complete PIN. These mechanisms to protect the PIN, however, are not further considered in this thesis[8].

Indirect Private Key Control For the indirect approach, an attacker needs to alter the public key in the mapping system. It needs to exchange the target's public key ($K_{\mathrm{D}}^{\mathrm{t}}$) with its own one ($K_{\mathrm{D}}^{\mathrm{a}}$). The first possibility would be to send a manipulated `USR_LOCATION_UPDATE` message to the mapping system (either an intercepted message from the target or a newly constructed one). This, however, will be recognized as the signature attached to the message does not match its content. Sending a message without a valid signature also fails verification as long as unauthenticated location updates were not previously enabled. Only UIDs for which authentication has been disabled are prone to these attacks. The mapping entries of these UIDs, however, are marked as such (cf. Sec. 4.3.2).
The alternative to sending a forged update message is to collaborate with the mapping system. A region being or allying with the attacker could manipulate the mapping entries under its own authority. The goal would be to exchange $K_{\mathrm{D}}^{\mathrm{t}}$ with $K_{\mathrm{D}}^{\mathrm{a}}$ as well as modifying the locator entry to point to an end system controlled by the attacker. Having stored $K_{\mathrm{D}}^{\mathrm{t}}$ beforehand the attacker can pose to the target's peers as the target itself while at the same time passing on messages to the target. This man–in–the–middle attack requires the attacker to use both key pairs—its own and the target's one. A peer would download $K_{\mathrm{D}}^{\mathrm{a}}$ and, for example, encrypt a message intended for the target. Due to the forged locator in the target's mapping entry the peer sends this encrypted message to the attacker. The attacker is able to decrypt and read or modify this message using $K_{\mathrm{E}}^{\mathrm{a}}$. Before forwarding the message, the attacker re–encrypts it using $K_{\mathrm{D}}^{\mathrm{t}}$ this time. Upon receiving the message, the target decrypts it using $K_{\mathrm{E}}^{\mathrm{t}}$.
This attack scenario is countered by using threshold cryptography. An end system's public key is stored at multiple regions, each being under a different authority. At least k shares of the public key are required to be able to reconstruct it. This means, for a successful attack, the attacker must collaborate with at least k regions. The parameters k, n can be determined by the end system. The minimum configuration, however, is $k = 3$ and $n = 5$. As the storage location of the shares is determined by the algorithm introduced in Section 4.3.1, an attacker must collaborate with a minimum of three semi–random regional authorities.

[8]For further reference on existing technologies see [RE03].

As mentioned earlier, the bootstrap is a very critical process. An attacker could aim at exchanging public keys before they are entered into the mapping system. This, however, is prevented by copying the public key into the mapping system out of band and signing subsequent location update messages (cf. Sec. 4.3.2). Simultaneously with shipping the smart card, the UID, card ID and public key are transferred to the responsible region. Other regions receiving a `USR_LOCATION_UPDATE` message to store a share of the public key query the responsible region for the public key in order to verify the message's signature. A responsible region collaborating with an attacker could send a malicious public key to the other regions upon request. This would result in a failed signature check and no shares being stored at other regions. The result would be that peers of the target are unable to retrieve the target's public key from the mapping system. However, it is not possible to distribute a manipulated public key. By trying to download its own shares an end system would recognize—in case an error can be ruled out—that its own region is or cooperates with the attacker.

Foreclosing Communication

Another goal of the attacker could be to not manipulate the private key but to foreclose any secure communication between the target and its peers. Again, the attacker's possibilities can be split into stand alone and cooperative attacks.

Distributed Denial of Service (DDoS) A common way to disturb communication is a Distributed Denial of Service attack. For this attack, a significant amount of clients send requests to one target. The latter one then collapses from the sheer amount of (unreasonable) requests. In the HiiMap architecture two goals for that attack are conceivable. The first is the end system itself. This does not differ from today's attacks and is not addressed by the HiiMap security framework. The second target would be the mapping system. By attacking a region, all mappings for the UIDs stored in that region could not be resolved any more. This threat is addressed by storing all vital information on multiple physical machines. The suggested architecture for the regional mapping resolvers is a Distributed Hash Table (DHT) with replication (cf. Chap. 3.3.1). This means that not a single but a complete set of nodes must be attacked in order to foreclose the mapping resolution for a specific UID. By increasing the replication factor, the effort for the DDoS is increased as well.

Cooperative Attacks The second scenario to foreclose communication is based on the cooperation between regions and the attacker. To prevent a peer from retrieving a target's public key requires the same efforts as discussed in the before mentioned Indirect Private Key Control paragraph (i.e., multiple semi–random regions must collaborate with the attacker). The responsible region, however, has more possibilities. It can either respond to a `USR_LOCATION_REQUEST` with either a wrong locator or with a UID not found message (`ERROR_MSG(NX_IDENT)`). The first one can be detected by the peer by starting a mutable authentication with the target based on the public key

retrieved from the mapping system. The peer is aware that the locator was incorrect in case the authentication fails.
The second possibility—to respond with `NX_IDENT`—cannot be countered with any mechanism. This means that the responsible region is able to foreclose any communication with a target under its authority. From a technical point of view, mechanisms could be introduced to also address this attack scenario (e.g., end system encrypted mapping entries, storing mapping entries at multiple regions). These mechanism, however, would only complicate the architecture without providing any real world benefit. It is suggested that the regional mapping authorities are controlled by provider–funded non–profit organizations. These organizations are supposed to be non–government ones as well. Some countries, however, might use these organizations to control the Internet connectivity of their citizens. Although this is obviously a bad thing, it does not add to today's capabilities of governments to limit and filter Internet connectivity. A government or force determined to isolate a certain set of users can take other measurements than altering mapping entries in order to reach their goals (e.g., filter mechanisms, IP table manipulations). The important aspect, however, is that no encrypted communication between two peers can be compromised by third parties[9].

4.4. Evaluation

The HiiMap security framework was evaluated in two ways. One part was a prototype implementation which extended the existing HiiMap demonstrator. Again, the prototype of the mapping system was deployed in the G-Lab experimental facility, and a modified client, capable of handling shares, was run on various clients. The outcome of this prototypical evaluation was the overall feasibility of the concept. A showcase was constructed in which an external tool (e.g., *ssh-keygen*) generated a key pair, and a text file was encrypted using the private key from this pair. Afterwards the public key was loaded into the HiiMap client software which calculated the shares and uploaded them to the mapping system. Another device was used to query the mapping system for the first client's UID and to download the shares. After reconstructing the public key an external tool was used to decrypt the text file which was copied to the second device via USB stick. This showcase proved that the HiiMap security framework is operational sound and can be used as intended. More details about the prototype can be found in Chapter 6.3.
The prototypical evaluation, however, does not allow to draw conclusions about the security level provided by the framework. In order to assess the approach in regard to its security strength, a qualitative metric was used. The framework was matched against the requirements from the Authentication, Authorization and Accounting (AAA) protocol evaluation guidelines [RFC2989].
These guidelines, assembled by the IETF, are considered to be suitable to assess an architecture's security level due to its broad requirement spectrum. The AAA principles

[9] This requires the implementation of the algorithms to be correct and no back–door implementation enforced by law.

interfere with all security dimensions. This means that the aspects listed in Chapter 2.1.4, i.e., *authentication, integrity, confidentiality, availability, non repudiation and privacy*, are covered. Fulfilling all requirements is a strong hint that mechanisms to provide these aspects are integrated[10]. Furthermore, the AAA protocol evaluation guidelines do not specify a certain set of security protocols. They only outline a collection of conditions to be followed in order to progressively customize the security level of one particular architecture. This means that NGI architectures can be assessed as well although the guidelines were specified with today's protocols in mind.
In the following the proposed HiiMap security framework is evaluated against the requirements of the AAA protocol evaluation guidelines. Four levels of coverage are taken as a metric to describe how the framework performs: The qualification levels are "T" for total coverage, "P" for partial coverage, "F" for failed coverage and "OS" for out of scope of this framework.

4.4.1. General Requirements

Table 4.3 lists all general requirements from the IETF AAA recommendation which will be briefly discussed point by point in the following. The first two issues, scalability and fail–over, are covered by the DHT–based architecture of the mapping system (c.f. Chap. 3.3.3). A mutual authentication is done between the client and the mapping system based on the used asymmetric cryptography principle. Transmission level security is generally supported if it is required by one party. It, however, has to be noted that the proposed framework resides on layers below the transmission layer. This means, it is transparent to eventually employed transmission layer security mechanisms. Nevertheless, the functionality provided by the framework can be used by higher layer mechanisms. Data object confidentiality is only partially covered as transmission can be secured but the mapping system is open to everyone. This is because any client is able to initiate lookup requests. As signatures are used to sign mapping responses within the framework, integrity of the data objects can be regarded as fully covered (cf. Chap. 3.3.1). The transport of certificates is not required because the framework is not relying on certificates. This aspect, therefore, is regarded as out of scope. All security–related data can be transmitted in a trustworthy manner—encrypted and signed. The system, therefore, fulfills all AAA reliability requirements. The HiiMap architecture is not limited to a single addressing scheme used for the locators. IPv4 and IPv6 can be used as well as possible future network layer protocols. Correspondingly, proxies are higher layer network elements and are, therefore, fully supported. The requests and replies from and to the mapping system are not supposed to be altered on their way. This is ensured by the integrity checks. The auditability aspect, therefore, is considered to be out of scope. The framework does not require a shared secret in terms of a common token—note the difference between shared credentials and Shamir's shared secret. It solely relies on asymmetric

[10] Please note that no system or architecture can provide a 100% security level. This is why even fulfilling all requirements of the AAA guidelines does not *guarantee* a failsafe and invulnerable architecture.

cryptography. Similar to the support of future protocols (network layer and higher), service specific attributes can seamlessly be included into the communication pattern as this does not affect the proposed architecture.

Type	Requirement	Rank
General	Scalability	T
	Fail-over	T
	Mutual authentication	T
	Transmission level security	T*
	Data object confidentiality	P
	Data object integrity	T
	Certificate transport	OS
	Reliable AAA transport mechanism	T
	Run over IPv4	T**
	Run over IPv6	T**
	Support proxy and routing brokers	T
	Auditability	OS
	Shared secret not required	T
	Ability to carry service specific attributes	T

*If required **Not limited to

Table 4.3.: AAA general requirements qualifications.

4.4.2. Authentication Requirements

The first requirement of the authentication section is the support for network access identifiers (NAI). These are inherently supported by the HiiMap architecture as it is based on the locator/identifier separation paradigm. Thus, the identifier can be utilized as NAI. As an authentication is not required to be authorized to resolve endpoints from the mapping system, this aspect is regarded as fully covered. Re–authentication is out of scope for the framework as connections to the mapping system only consist of a single request and response message scheme. Higher layers, however, may implement this mechanism. All other requirements concerning the authentication are transparently supported as they reside on higher layers.

4.4.3. Authorization Requirements

A summary of all authorization requirements is given in Table 4.5. Support for static and dynamic IP address assignment is possible if required. This as well as the support for Remote Authentication Dial-In User Service (RADIUS) is covered correspondingly to the IPv4/v6 support outlined in the general requirements Section 4.4.1. The framework covers the capability to reject single participants, e.g., if they behave in a suspicious way, due to the integration of signatures. Hence, also access rules could be applied. As this is not meaningful in the given scenario (everybody is entitled to retrieve

Type	Requirement	Qualification
Authentication	Network Access Identifier (NAI) support	T
	Authorization without authentication	T
	Re–authentication on demand	OS
	Challenge-handshake authentication protocol support	T*
	Extensible authentication protocol support	T*
	Password authentication protocol/clear-text passwords	T*

*If required

Table 4.4.: AAA authentication requirements qualifications.

mapping entries), this criteria is regarded as out of scope. Layer 2 Tunneling cannot be prohibited. The mapping system, however, cannot be bypassed and, therefore, Layer 2 Tunneling has no benefit to an attacker in regard to the framework. Reauthorization on demand is not required and, therefore, neglected. Queries from clients do not have a significant influence on the system's state, nor do they change them (as far as it is not a malicious DDoS attack). Furthermore, the distributed self–organized nature of the DHT–based mapping system guarantees a steady operable state. Unsolicited disconnects, i.e., a disconnect initiated by the mapping system, is regarded as not fulfilled (= failed). The framework handles requests on an event–basis. To this extent, there is no way for a server to disconnect an ongoing transmission.

Type	Requirement	Qualification
Authorization	Static and dynamic IP address assignment	T*
	RADIUS gateway capability	T*
	Reject capability	T
	Preclude layer 2 tunneling	OS
	Reauthorization on demand	OS
	Support for access rules and filters	OS
	State reconciliation	T
	Unsolicited disconnect	F

*If required

Table 4.5.: AAA authorization requirements qualifications.

4.4.4. Accounting Requirements

Accounting, in general, is not within the scope of this framework as an evidence of who has requested which address at a given time is not practical nor applicable to a global mapping system. The system puts a focus on integrity of the requests/responses in terms of validity rather than on monitoring each of them. Basic requirements, however, within the accounting section (Table 4.6) can be regarded as fulfilled

which is guaranteed delivery and accounting timestamps (for location and public key updates, cf. Tab 4.1). Dynamic accounting which is regarded as out of scope, denotes multiple recordings of timestamps for a single session or transmission respectively. The framework does not integrate sessioning, therefore, this has to be included in higher layer services. Other requirements as real–time accounting, an obligatory compact encoding, the possibility to extend the accounting records as well as a batch accounting are not incorporated here.

Type	Requirement	Qualification
Accounting	Real–time accounting	OS
	Mandatory compact encoding	OS
	Accounting record extensibility	OS
	Batch accounting	OS
	Guaranteed delivery	T*
	Accounting timestamps	T
	Dynamic accounting	OS

*If required

Table 4.6.: AAA accounting requirements qualifications.

4.4.5. Result

Evaluating the overall results including general requirements, authentication, authorization, and accounting aspects, the framework fulfills 92% of the requirements, partially covers 4% and misses 4%. These results are obtained neglecting out of scope aspects.
The interpretation of these results, however, is not straight forward. The percentage of fulfilled requirements has only a limited significance. As the AAA protocol evaluation guidelines are kept general and must be interpreted individually for each architecture, two architectures cannot simply be compared based on the number of fulfilled requirements. It is more important to look at the partially fulfilled and failed requirements. As the HiiMap security framework fails to fully cover two requirements, these two are discussed in more detail in the following.

- *Unsolicited Disconnect:* In the AAA protocol evaluation guidelines this requirement reflects an entity's ability to disconnect a session upon request of an authorization authority. The HiiMap architecture fails this requirement as no such authorization authority exists. Although the GA is a logical entity supervising the regional authorities, it has no ruling power as to which mapping request to answer and which not. Furthermore, sessions between end systems and the mapping system are limited to single request and respond cycles. These cycles are not even broken in case authentication in the form of signature checks fails (i.e., the session is completed by sending an error message where a response is due).

Although the requirement is not covered by the architecture, it has no impact on the overall integrity of the system. Any Internet participant is authorized to query the mapping system, and the signature checks verify the legitimacy of write request (i.e., location updates). Once the signature check passed, there is no reason to abort the subsequent transaction in the database.

- *Data Object Confidentiality:* The HiiMap security framework only partially covers this requirement. Data objects sent between two end systems can be encrypted, and peers are enabled by the framework to authenticate each other. This means that data object confidentiality is supported between end systems. All data objects stored in the mapping system (i.e., mapping entries), however, are not confidential. Any Internet capable device can be used to access all world wide stored mapping entries. It is even one of the fundamental tasks of a mapping system to provide means of resolving mapping requests.

  The lack of full data object confidentiality, however, reveals a weakness in regard to privacy of the locator/identifier separation paradigm. This problematic issue is described in the next chapter, and a solution for the HiiMap architecture is presented.

4.5. Conclusion

In this chapter, the HiiMap security framework was introduced. Its core is the threshold cryptography based public key infrastructure which uses the HiiMap mapping system as a distributed resource. Contrary to other NGI security proposals extending the identifier address space with a cryptographic namespace, the HiiMap security framework has several advantages. It does not suffer from the static key binding and key guessing problem and inherently provides a distributed trust entity. This is achieved by only loosely coupling an identifier and its security token (e.g., UID and public key) whereas the cryptographic namespace is based on a static computational coupling.
The HiiMap security framework provides a strong trust anchor within the architecture. It secures the HiiMap mapping protocol and builds a foundation for other security protocols and applications above the network layer. The client key management, thereby, is based on cryptographic smart cards to increase security and usability. In this chapter the operation of the framework was discussed, ranging from the crucial initial bootstrap up to unauthenticated location updates for devices not strong enough to compute cryptographic puzzles.
In the last section of this chapter the framework was evaluated after describing typical attack scenarios and the framework's countermeasures. The evaluation showed by means of a prototype the overall feasibility of the framework. The theoretical evaluation against the *AAA protocol evaluation guidelines* showed that the framework covers all important aspects except for one. This not yet covered aspect concerning data object confidentiality, however, is dealt with in the next chapter.

5. Location Privacy in Locator/Identifier Split Architectures

One of the requirements of a Next Generation Internet (NGI) architecture, as listed in Chapter 2.2, is mobility. By using the locator/identifier split principle, combined with the introduced HiiMap mapping approach, this requirement is satisfied. Beside its benefits, however, the locator/identifier separation paradigm has a grave disadvantage (cf. *data object confidentiality*, Chap. 4.4.1). The topological location of each end device is disclosed within the architecture, enabling an attacker to localize any node. Furthermore, by recording the whereabouts of a target over time, a tracking profile can be established. The ability to trace a mobile node by an untrusted third party is a direct violation of the mobile user's privacy. It can cause serious damage to his personal, social, and professional life. In this chapter the location privacy problem is outlined and possible solutions are discussed. After analyzing the existing approaches a novel concept is introduced. Finally, the various approaches are compared based on a Capital Expenses (CAPEX) and Operational Expenses (OPEX) supported cost model. As with the previous chapter, the terms *attacker* and *target* are used. The term target is used to specify an entity whose privacy is threatened by an untrusted third party, the attacker.

5.1. Problem Statement

Location privacy is the ability to prevent other parties from learning one's current or past location. In order to get such ability, the mobile node must conceal any relation between its location and the personal identifier [BS03]. This, however, heavily contradicts the locator/identifier split principle. Each node is identified by a unique address, and the mapping system establishes the link between the location information and the identifier. Note that in this context the location information normally refers to the topological location and not the geographical one. Quite often, however, the geographical location can be extracted from the topological one up to a certain granularity. A given mobile node's address, out of an access point's address space, combined with the knowledge of its geographical location, limits the possible whereabouts of the node to the coverage area of that specific access point.
The threat to location privacy can be divided into three steps:

- Identifying
- Locating
- Tracing

In a locator/identifier split architecture, an attacker can identify a target by its unique address, the identifier. This step is an inherent feature of any communication protocol which is not solely based on broadcast mechanisms. It requires that communication partners specifically address a message to a certain node they want to communicate with.
The next logical step after identifying a target is to localize it with maximum accuracy. By querying the mapping system, an attacker is able to obtain the current valid locator for any identifier currently in use by a connected end system. It is, therefore, using a basic and fundamental feature of the locator/identifier split architecture to obtain the target's locator. To finally locate the target, knowledge of the address space structure is required. Again, at this point, the locator/identifier split is in favor of the attacker. To counter scalability issues, the structure of the locator address space is organized in a strictly hierarchical manner (see Chapter 3.1.1). This simplifies the task to estimate the geographical location out of the topological one as hierarchies will reflect geographical, political and economical structures.
The third step consists of tracing the target while it is moving around the Internet. This can be done by periodically locating the target. An attacker needs to query the mapping system for a specific identifier and record the returned locator combined with extracted geographical information and a time–stamp. Over time a movement profile of the target can be created. Thereby, at least the recording of the locators must be performed in real–time as the mapping system stores only currently valid and no historical data. Figure 5.1 depicts the tracing process.

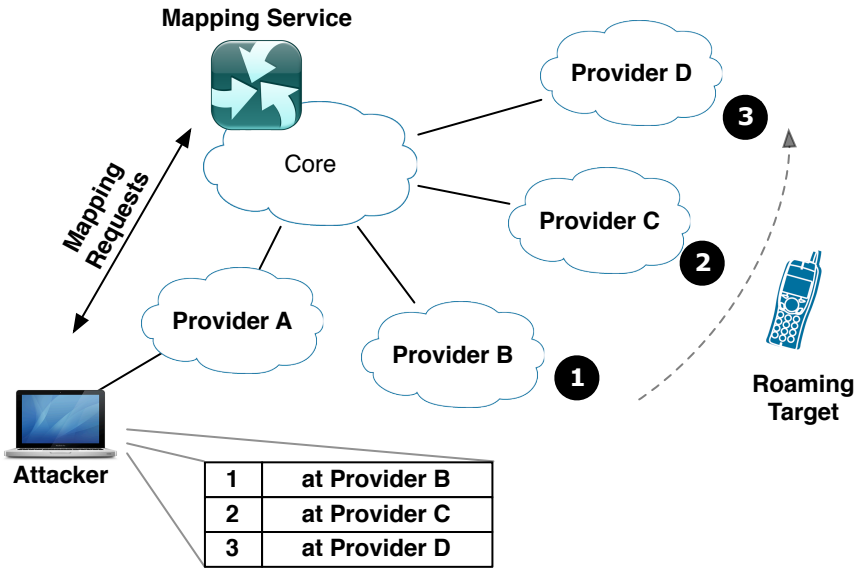


Figure 5.1.: Tracing a target.

The gained information can be used for various interests. This ranges from legal applications, such as personalized advertisements, up to illegal actions, like stalking a person. Depending on the application, the whole movement record can be relevant

or only a single whereabout to prove that the target was present at a specific place at a specific time. Furthermore, studies showed that a node's future location can be predicted with a high accuracy based on historical data [SQBB10]. Rather than on mathematical functions, this is based on the social behavior and acquired habits of users. The daily routine of a person tends to be same for each day, e.g., going to work, shopping and leisure activities.

5.2. NGI Location Privacy Proposals

Privacy protection is not a novel challenge and has been addressed prior to the upcoming of the locator/identifier separation paradigm. So–called *mix networks* [Cha81] are a common approach to allow nodes to anonymously send messages to peers. Location privacy protection, however, is not about anonymity. While mix networks might also provide location privacy, the following proposals do not try to hide a node's identity from its peers. They rather try to hide a node's location information while still allowing peers to contact the protected node. This is not possible with mix networks. A node protected by a mix network is only able to receive data from a correspondent node in case it previously addressed a packet to it to set up the path through the mixes.
The proposals introduced in the following are all based on or are extensions to the Host Identity Protocol described in Chapter 3.1.1.

5.2.1. HIP Location Privacy Framework

The HIP Location Privacy Framework [MSS+06] provides location privacy by introducing so–called Rendezvous Agents (RVA). A local network area accompanies a RVA and is called protected area. These protected areas have similarities to Autonomous System (AS) but are not limited to these in terms of size. A protected area, for example, could span over several ASs.
RVAs are middle boxes which are located at the border of a protected and a public network and forward packets in both directions. They, thereby, shield the nodes within the protected area from the rest of the Internet and conceal actual locators of nodes from outer ones. Similar to a NAT, a RVA administrates a pool of global locators and leases these to nodes within its protected area. In case a node from within the protected area wants to send a packet to a correspondent node on the outside, the packet is required to go through the RVA before leaving the protected area. The RVA swaps the original source locator to the global one currently leased to the node before forwarding the packet. It is, therefore, impossible to extract the exact location of the node by means of the network layer. A packet sent to a node within the protected area lists the leased global locator as the destination locator. The agent then looks up the internal locator of the destination node by querying an internal mapping. After replacing the global locator with the destination node's one in the packet, the agent forwards it into the protected area. Figure 5.2 shows an example topology.

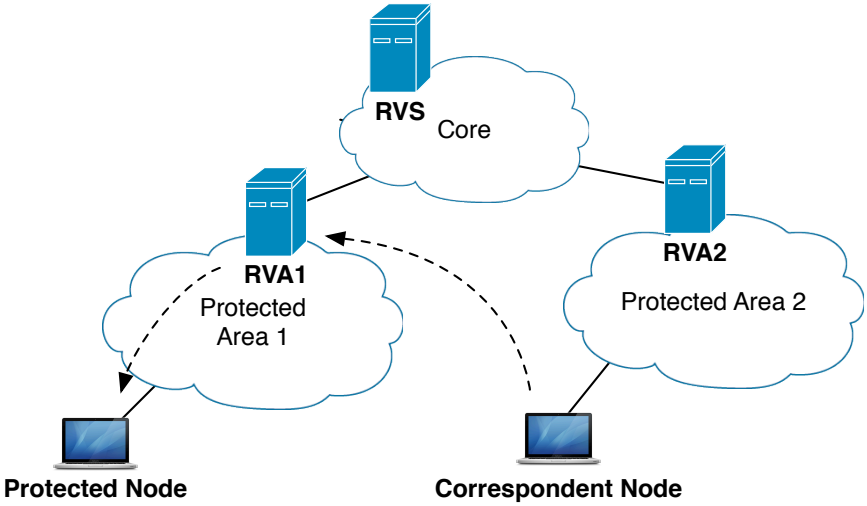


Figure 5.2.: Sample topology of the HIP Location Privacy Framework.

In terms of mobility, a RVA supports two types of handovers. The internal handover occurs whenever a mobile node roams within a protected area. In this case the RVA is informed of the mobile node's newly acquired internal locator. Correspondent nodes on the outside do not recognize the handover event other then a potential change in the round trip time. The second type of handover is required in case a mobile node roams between two protected areas. In this case, the mobile node needs to inform the old RVA (RVA_1) about the change and register itself with the new RVA (RVA_2). RVA_2 then updates RVA_1 about the successful registration in order to complete the handover. To reflect the new global address of the mobile node, HIP's rendezvous server (RVS) is updated by RVA_2. Packets still addressed to RVA_1 are forwarded to RVA_2 and then relayed to the mobile node. Contrary to the internal handover, correspondent nodes are aware of the locator change and recognize the handover event.

5.2.2. BLIND

The BLIND Framework [YN06] was originally designed to provide identifier privacy against third parties. This is done by using an enhanced two–round–trip Diffie–Hellman key exchange mechanism. It substitutes the real identity of two peers with so–called Blinded Host Identity Tags (BHIT). During the key exchange the two nodes identify themselves to each other but onlookers only see the non–encrypted BHITs. By using different BHITs for each communication, no conclusion can be drawn from recorded traffic patterns.
To provide location privacy, BLIND introduces Forwarding Agents (FA) which, similar to the HIP Location Privacy Framework's RVAs, lease public locators to mobile nodes. Two types of FAs, thereby, exist. Untrustworthy ones not requiring authentication from the client and trustworthy ones. The downside of untrustworthy FAs is that they do not support node mobility. A node using the service of an untrustwor-

thy FA does not authenticate itself towards the FA. The FA distinguishes nodes based on their locators. A roaming node changes its locator and, therefore, the FA is not able to safely track a mobile node. Trustworthy FAs have other means to identify a mobile node. They support node mobility, however, a mobile nodes identity must be disclosed to the FA. This contradicts the initial BLIND objective.
The Enhanced Location Privacy Framework by Maekawa et al. [MO09] is based on BLIND. It extends the BLIND framework with Temporary Host Identity Tags (THIT) in order to fully support node mobility. The goal of the enhanced framework is to fulfill the objective of identity privacy in combination with location privacy and node mobility. They, therefore, introduce THITs which a mobile node creates. This THIT is based on a node's Host Identity Tag (HIT) but it is not possible to conclude from a THIT to the node's HIT. The THIT is used to authenticate a mobile node against a FA. The FA stores the following values per registered mobile node: $\langle$THIT, $\mathrm{IP}_{\mathrm{real}}$, $\mathrm{IP}_{\mathrm{lease}}$, lease time$\rangle$. By authenticating, using the THIT, a mobile node is able to update its locator (i.e., $\mathrm{IP}_{\mathrm{real}}$) in the FA's forwarding database. Untrustworthy FAs do not exist in the enhanced BLIND framework.

5.2.3. Limitations of Existing Privacy Proposals

HIP Location Privacy Framework The HIP Location Privacy Framework does not offer full location privacy. While a mobile node's exact location is concealed by the RVAs, it still can be tracked on a protected area granularity. Whenever a node roams from one protected area to another, it needs to lease a new global routable locator out of the new RVA's address pool. This change of global locator is advertised to the RVS and correspondent nodes. Depending on the size of the protected areas, it is, therefore, still possible to determine the cities or countries in which a specific mobile node currently resides. A mobile node is also not able to use a foreign RVA (i.e., from a different protected area) as its relay to obscure its actual location. This is due to the locator assigned to the mobile node as it is not routable outside of the protected area. A foreign RVA has no means to forward packets to an alien locator.
To address this problem, the authors in [MSS+06] suggest to establish rather large protected areas which cover a large amount of ASs. As a result, the leaked information with each global handover event is very coarsely grained, and a majority of roaming occurs within the protected area. To achieve this, they position the RVAs very close to or, respectively within the Internet's core. The placement of the RVA, therefore, is very inflexible, which introduces another downside in regard to transit costs. This aspect will be covered in Section 5.4.

BLIND In the HIP architecture, the mapping between identifiers (i.e., HIT) and locators for mobile nodes is done by so–called rendezvous servers (RVS). A node may register its HIT and current locator with a RVS in order to be reachable by other nodes [RFC5204]. This RVS acts like a MobileIP home agent [RFC5944]. Upon receiving a packet destined for a registered HIT, the RVS forwards the packet to the enlisted locator of that node. BLIND, however, breaks this mechanism [MO09]. Peers use different

BHITs for each correspondent node and communication flow. This means that the RVS is not able to track a node using BLIND. It requires unblinded HITs to base its forwarding decision on. Even enhanced BLIND does not solves this issue. The temporary HITs enable FAs to track a mobile node. In the same way RVS could track nodes based on THITs. Correspondent nodes, however, only know the HIT of a node they want to reach. The RVS for nodes using BLIND, therefore, is useless as no mapping between THITs and HITS exists. Providing such a mapping would obsolete BLIND's objective to provide identity privacy. As a result, mobile nodes are not reachable by correspondent ones.
The initial BLIND approach while providing full location privacy does not support node mobility. This is due to the FA's inability to track a node. Only trustworthy FAs support this feature which on the other side, require a node to identify itself. Enhanced BLIND, however, addresses this problem and introduces a mechanism to support node mobility. Not using FAs for the initial BLIND approach would allow node mobility at the cost of no location privacy between the mobile and the correspondent node.
Another downside of BLIND is that the correspondent node is required to know the mobile node's identity beforehand. During the so–called base exchange between the correspondent and the mobile node, BHITs are established for the communication flow, and mutual authentication is performed. As nodes using the BLIND framework are not listed in RVSs or other global databases (due to identity privacy reasons), a correspondent node must have learned the mobile node's identity (e.g., public key, fingerprint of the key) before the connection request. In a real world example this would mean that an online shop must know all possible customers beforehand. An out–of–band registration is required.
The placement of the FA is not covered in either BLIND or enhanced BLIND. While the framework does not limit the flexibility in FA placement, the aspect is not discussed.

Importance of node reachability Several applications do exist, in which a node other than a server needs to be contacted at an arbitrary point in time. These are, for example, any kinds of push services (e.g., e–mail, instant messenger) or telephony and video conferencing services. Even though mechanisms do exist in today's Internet architecture to overcome no reachability barriers like NAT, these are only bug–fixes to the problematic side effects of the NAT or firewall solution. Designing a novel and improved architecture should not incorporate old limitations and be required to rely on the established add–ons to fix these. By providing full end–to–end reachability, future applications might benefit from this without having to design a novel bug–fix to overcome the old obstacles.

Conclusion Table 5.1 summarizes the attributes of the location privacy frameworks. The HIP location privacy framework fulfills all aspects except for full location privacy and RVA placement flexibility. Especially the first aspect renders the approach partially obsolete. BLIND and its enhanced version support full location privacy. They, however, fail to enable node reachability and do not allow for arbitrary connections.

This means that a protected node must be known to correspondent nodes a priori. This limits the flexibility in communication possibilities as long as authenticated peers are required. FA placement flexibility is an open topic in these frameworks.

	HIP Location Privacy	BLIND	Enhanced BLIND
Full location privacy	-	+	+
Node mobility	+	-	+
Protected node reachability	+	-	-
Arbitrary connections	+	-	-
RVA/FA placement flexibility	-	o	o

Table 5.1.: Comparison of existing location privacy mechanisms.

5.3. HiiMap Privacy Service

In order to support location privacy in the Hierarchical Internet Mapping Architecture (HiiMap), a novel privacy service is introduced. It provides full location privacy while overcoming the limitations of the discussed frameworks. A strength of the concept, beside providing location privacy, is its possibility to organize its components to limit the service's costs. This is evaluated after the service is outlined. The work in this chapter has been partially published in [Han10a, Han10b, Han11].

5.3.1. Privacy Service

The HiiMap privacy service is based on proxies which hide a protected node's location. The service is offered by third party providers, and a node is required to register for an individual privacy service.
The presented approach, thereby, does not offer location privacy against the privacy service provider. The service provider is aware of the node's topological location at any time. Contrary to MIX networks [Cha81], for example, the objective of the privacy service is to conceal location information only towards third parties and not to all network elements (including itself). The reason behind this design decision is that a node's whereabouts are always disclosed to some parts of the network as well as the node's internet service provider (e.g., for billing reasons). The efforts required to conceal location information from the privacy service, therefore, are not justifiable. It is the user's task, however, to select a privacy service provider he trusts.

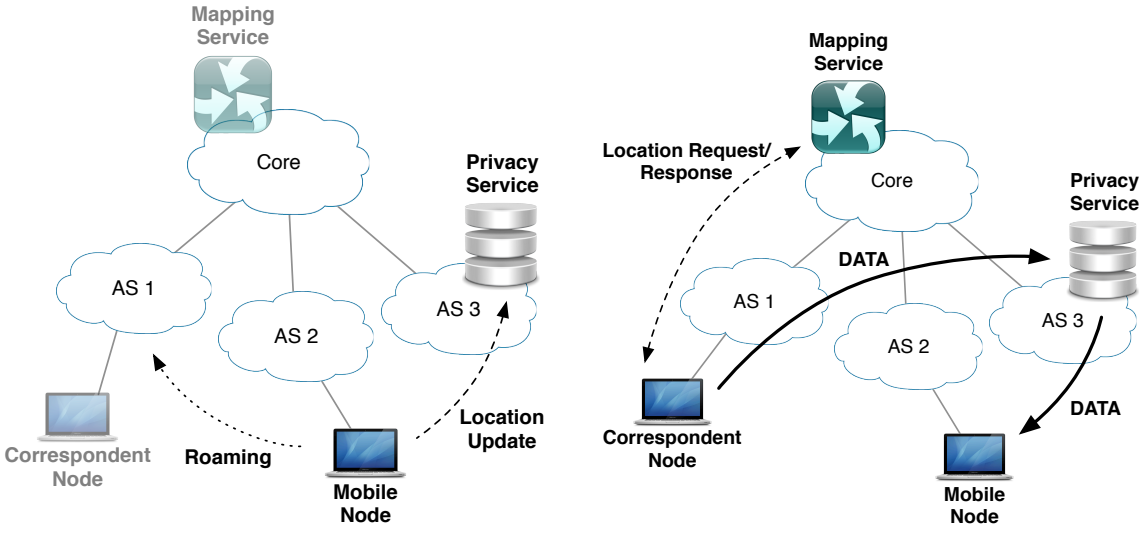
Components and Abbreviations

Before outlining the details of the HiiMap privacy service, a few components and abbreviations are introduced in addition to the ones from Chapter 3.3.1 (*Unique Identifier (UID), Regional Prefix (RP), locator and mapping system*). The interaction between these parts to provide location privacy will be explained subsequently.

- **Care–of locator:** The care–of locator is used to hide the real locator of a node. It is, however, a valid locator which belongs to the address space of the privacy service (defined below). It cannot be distinguished from any other locator.
- **Mobile node:** A mobile node is an end system which can change its point of attachment to the Internet. It possesses at least one UID and locator. It wants to roam freely without other nodes being able to track its location.
- **Correspondent node:** At least one partner is needed for communication. The correspondent node can either be a fixed or a mobile node.
- **Privacy service:** A privacy service helps the mobile node to protect its location privacy. It is located somewhere within the network, and several competing services may exist.

Location Update

After registering for a privacy service, the service's care–of locator is entered in the mobile node's mapping entry. All subsequent `USR_LOCATION_UPDATES` from the mobile node are sent to the privacy service. Contrary to the operation of the HiiMap mapping system without the privacy service, the service is now in charge of keeping track of the mobile nodes' topological location. In that way only the mobile node and the privacy service are in possession of the mobile node's location information. The mapping system is not aware of the mobile node's actual whereabouts. Figure 5.3(a) illustrates the location update in case of a roaming event.



(a) Location update of the roaming mobile node. (b) Data flow between the corresponding and mobile node.

Figure 5.3.: Interaction between the proxy service, corresponding and mobile node.

Packet delivery

Figure 5.3(b) illustrates packet delivery to and from the mobile node. In order for a correspondent node to send packets to the mobile one, it needs the mobile node's UID. The mapping system is as usual queried for the locator currently associated with this UID. Upon receiving the request, the mapping system returns a `USR_LOCATION_RESPONSE` message, containing the care–of locator. As this care–of locator does not differ from other locators, the correspondent node addresses its data packets to this locator. The privacy service, therefore, acts like a proxy and is transparent to the correspondent node.
Upon receiving the correspondent node's data packet, the privacy service retrieves the destination UID from the packet header. It initiates a lookup in its internal mapping database—which in its structure is similar to the one of the mapping system—and forwards the data packet to the mobile node's actual locator. This is done by encapsulating the original packet with a new header. This header contains the privacy service's locator (i.e., the care–of locator) as source and the mobile node's current locator as destination address. This packet is then routed to the mobile node which decapsulates the original packet and processes it as usual.
As the correspondent node is not aware of the privacy service[1], it is not required to follow another protocol when communicating with one of the service's subscribers. Even mutual authentication based on the HiiMap security framework is possible without any extension as the privacy service only forwards packets based on network layer information.

Return path

The response from the mobile to the corresponding node could be routed on two different paths. The first possibility for the return path would be to send packets on the direct path between the mobile node and the corresponding node. This approach has two problematic issues as already identified for Mobile IP [RFC3775, Sch03].

- In order to conceal its topological location, the mobile node would be required to use its care–of locator as the source address in the packet header. A network middle box (e.g., firewall), however, might discard this packet. A packet filter located at the edge of the AS the mobile node is currently residing in could recognize that the source address does not belong to the address space of the AS and drop the packet.

- Using its actual locator as source address, however, would reveal the mobile node's location. Furthermore, an ingress packet filter at the corresponding node's network might drop the packet. This is because no outgoing connection was established with this locator.

[1] Neglecting the detection of the privacy service due to the added delay and possible well–known locator for the moment.

The second possibility for the return path is to tunnel the response packet back to the privacy service. The privacy service decapsulates the response and forwards it to the corresponding node. This has the advantage that the response passes all packet filters without the risk of being dropped (in case no other irregularity is detected by the filters). Figure 5.4 summarizes the data flow between the corresponding and the mobile node.

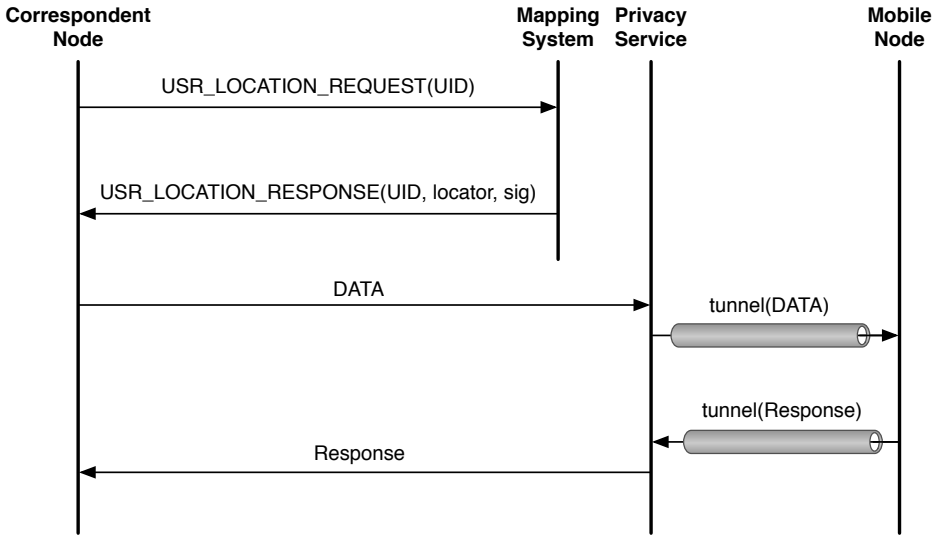


Figure 5.4.: Message flow using the HiiMap privacy service.

Triangular Routing

Although the data flow in Figure 5.4 might suggest otherwise, the detour of the packets, required to pass the privacy service, represent an unwanted side effect. This effect is called *triangular routing*. It is a known negative effect and has, for example, also been identified as a problematic issue for Mobile IP [Sch03].
Packets between the correspondent and the mobile node are not forwarded on the direct path between the two nodes. They are required to be forwarded to another network entity, the privacy service, first. This effect is called triangular routing as the privacy service forms the tip of an imaginary triangular while the direct path between the correspondent and the mobile node forms the base. Triangular routing has three unwanted properties:

- Delay due to processing time in the additional network element.
- Delay due to the extended path.
- Traffic in parts of the network other than the direct path.

Contrary to Mobile IP's home agent, however, the privacy service is not required to reside at the mobiles node's home network. Instead, it can be positioned anywhere

within the network. Combined with a privacy service provider's possibility to operate multiple proxies provide a possibility to lessen the impact of triangular routing. It is called *proxy selection* and introduced in the following subsection.

5.3.2. Proxy Selection

Figure 5.5 illustrates a sample topology in order to explain the benefits of proxy selection. There are two paths between the correspondent and the mobile node. The upper path includes two hops (in terms of ASs) and the lower only one. On each path a proxy is present which belongs to the privacy provider. Assuming that the metric for a better connection between two end points is the amount of hops (the less the better), the lower path is the preferable one. This means, by selecting proxy A instead of proxy B, the negative impact of triangular routing can be lessened for the communication between the correspondent and the mobile node. In this example, triangular routing is even non–existent for the lower path as it is the direct one.

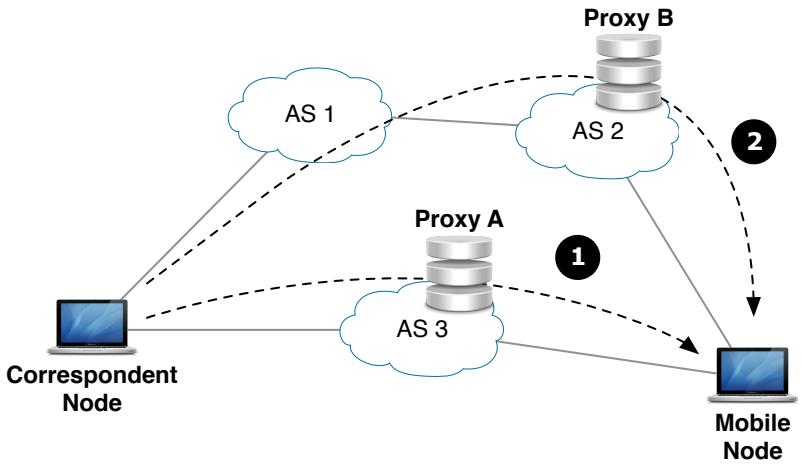


Figure 5.5.: Lessening the impact of triangular routing by proxy selection.

The goal for a privacy service provider, in order to provide the best possible service, therefore, must be to select a proxy which is as close to the direct path between two peers as possible. The task of selecting the best proxy possible, however, is not trivial. In the following, two mechanisms (one with a variant) are introduced and subsequently evaluated.

Privacy Service Migration

The first method uses the principle of service migration. A privacy service provider operates multiple world wide distributed proxies. A mobile node which has a subscription with the privacy service provider is assigned a conveniently located proxy out of the provider's pool. The proxy's locator is entered in the mobile node's mapping entry as the care–of locator. Please note, for the sake of comprehensibility the

authentication aspect towards the privacy service provider, the proxies and the mapping system is omitted in the description of this and the following mechanism. The details concerning security, however, will be given subsequent to the evaluation of the methods.
In case the mobile node roams, a new proxy is selected which better reflects the node's location. The node's context is transferred from the old proxy to the new one, and the care–of locator in the mapping system is updated. To a correspondent node the process appears to be a regular roaming event. By *following* the mobile node with the proxy, it is ensured that the proxy is relatively close to the direct communication path between the mobile node and its peers[2].
As the newly assigned proxy, at least partially, reflects the mobile node's location, an attacker will still be able to track the mobile node. A policy, therefore, must be introduced which allows the owner of a mobile node to select a location privacy granularity. The granularity determines as to which extent a proxy is allowed to follow the mobile node. Different levels could be "non–following", "continent–based", "country–based" or "city–based". The levels range from full to less location privacy and in reverse order from heavy to less impact by triangular routing. A privacy service provider might also charge differently for each level. A sample service migration process is shown in Figure 5.6.

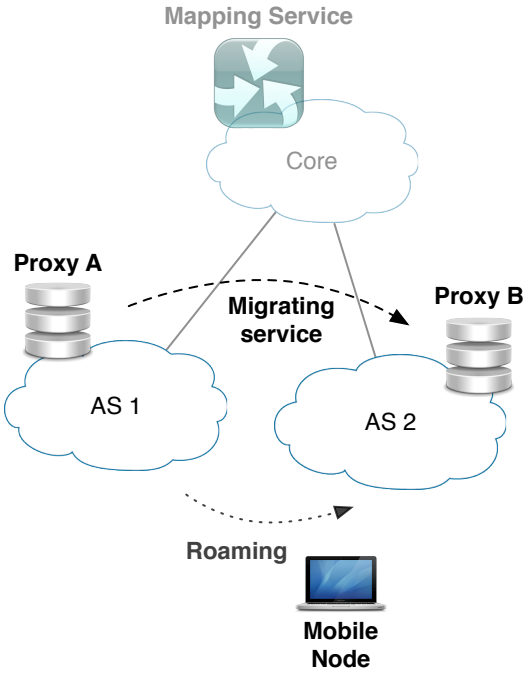


Figure 5.6.: Privacy service migration.

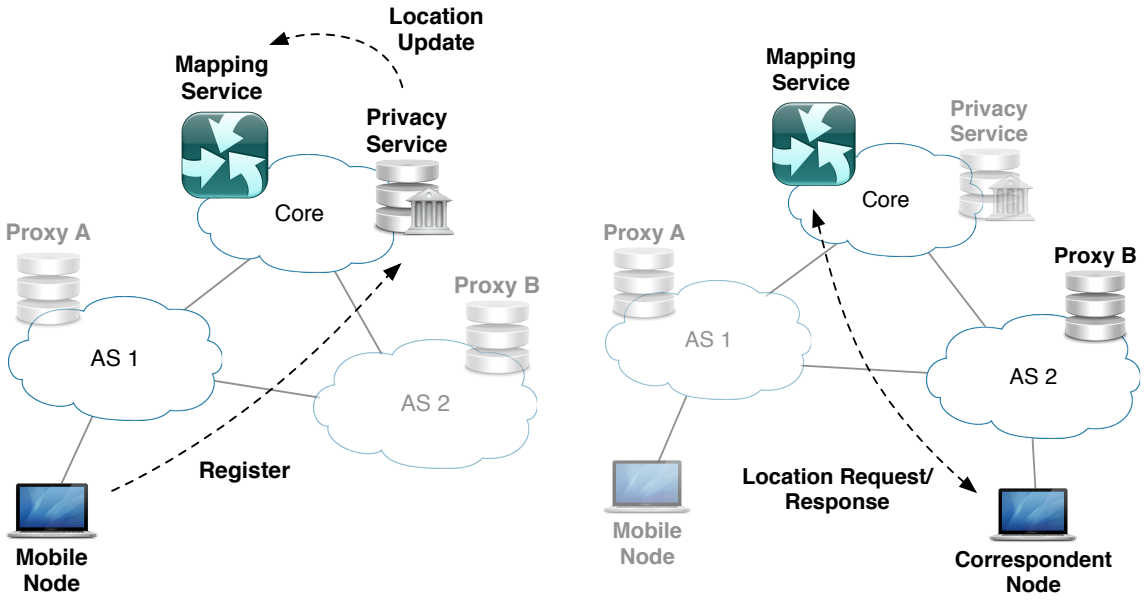
In this example the mobile node holding the subscription to the privacy service roams from AS 1 to AS 2. The previously assigned proxy was proxy A. Due to the roaming

[2]This, of course, heavily depends on the distribution of proxies.

event the mobile node's context is transferred to proxy B. To reflect this change, a location update message is sent to the mapping system, containing the locator of proxy B.

Proxy Selection

This mechanism takes a different approach. Instead of selecting a proxy which is as close as possible (in terms of location and policy) to the mobile node, a proxy is selected which reflects the location of the correspondent node. This means, however, that the proxy must be chosen individually for each of the mobile node's communication flows. The advantage is that the location of the proxy does not reveal any information about the mobile node's whereabouts.



(a) Mobile node registers with the privacy service. (b) Correspondent node queries the mapping system, and proxy B is selected for the communication flow.

Figure 5.7.: Overview of the proxy selection mechanism with registration (a) and query (b).

Similar to the privacy service migration approach, a mobile node has to register with the privacy service first (cf. Fig. 5.7(a)). Instead of selecting a proxy and pushing its care–of locator into the mapping system, however, a list of all proxy locators is sent to the mapping system. Each mapping entry consists of a single UID and one to multiple locators. This is intended to support multihoming, for example. In case a mobile node subscribed to the privacy service, all available care–of locators of this service are enlisted in the mapping entry. Whenever a correspondent node queries the mapping system for the mobile node's locator, the proxy, topology–wise closest to it, is selected. For the corresponding node the care–of locator belonging to proxy

B is selected in the example shown in Figure 5.7(b). To this end, two variants of this mechanism exist:

- *Selection by the mapping system:* In this variant, the mapping system chooses a care–of locator which best matches the correspondent node's one. This locator is then returned as the only one in the `USR_LOCATION_RESPONSE` message.
- *Selection by the end system:* Contrary to the first variant, all care–of locators are returned to the correspondent node in the location response message. It is then the correspondent node's task to select an appropriate proxy.

For both variants, the topological proximity between the mobile and the correspondent node is calculated based on the locator. One of the benefits of the locator/identifier separation paradigm is the possibility to strictly structure the locator addressing scheme (cf. Chap. 3.1.1). This eases the distance calculation between two locators.

Evaluation of Proxy Selection Methods

Both methods and the variants have their advantages and downsides. Table 5.2 summarizes the most important aspects. The first method, privacy service migration, does not provide full privacy due to its following character. The only policy supporting full location privacy disables the selection of a suitable proxy and, therefore, does not help to lessen the impact caused by triangular routing. The proxy selection methods, on the other hand, provide full location privacy as an attacker is not able to conclude the mobile node's location from the selected proxy. The selection is always based on the correspondent node and, therefore, no information of the mobile node is revealed.
Each method puts a different burden on the mapping system. The approach where proxy selection is done by the mapping system introduces the highest amount of complexity for it. Additional logic is required in the mapping system to compute topology–wise proximities and to select the best matching proxy. The privacy service migration method does not require any additional logic on the mapping system's side. Any proxy migration, however, results in a location update which needs to be processed by the mapping system. The least burden on the mapping system is caused by the approach in which the end system does the proxy selection. The mapping system is only required to store the care–of locators. Mobile node roaming is handled by the privacy service (this mechanism will be detailed in the next subsection), and no location updates are sent to the mapping system.
The burden on the end system, however, is inverse to the mapping system's one. The end system proxy selection approach requires the algorithm of proximity calculation to be executed on the end system. Both other approaches do not require any additional functionality on the end system side other the HiiMap mapping protocol. The same evaluation holds true for the integrity of the system. While the migration algorithm and the mapping system are expected to select the best possible proxy, it cannot be guaranteed for the end system. An attacker could willingly choose a different proxy in order to harm the system. This could be a distributed denial of service attack against a single proxy or intentional network clogging. The end system based

approach has another disadvantage compared to the other ones. A change in the proximity calculation algorithm is difficult to role out to each end system whereas only a few network elements require a software update for the migration and mapping system based methods.
The last characteristic is the introduced overhead in terms of additional network traffic. The privacy service migration approach requires user contexts to be transferred between proxies. These contexts, however, are expected to be of limited size. The end system proxy selection method requires the full care–of locator list to be sent upon each location request. This sums up to a significant amount of data for the complete network. No additional network overhead is introduced by the mapping system based proxy selection.

	Migration	Selection end system	Selection mapping system
Full location privacy	-	+	+
Burden of the mapping system	o	+	-
Burden of the end system	+	-	+
Integrity of the system	+	-	+
Algorithm change	+	-	+
Network traffic	o	-	+

Table 5.2.: Comparison between the three different proxy selection methods.

As a result of the evaluation (cf. Tab. 5.2) of the three different methods, the mapping system based proxy selection is considered the most suitable. The gravest disadvantage of the privacy service migration method is its lack of full location privacy support. The downside of the end system proxy selection is that it is not transparent to correspondent nodes. A standarized protocol, therefore, would be required which needs to be supported by any end system. The mapping system proxy selection approach is completely transparent to the user while providing full location privacy, even though it puts the most burden on the mapping system. In the following, the operation of the mapping system based proxy selection method is detailed.

Mapping System Based Proxy Selection

After having registered for the privacy service with a specific provider, the mobile node sends its `USR_LOCATION_UPDATE` messages to this privacy service instead of the mapping system. The current locator of the mobile node, however, is not pushed to all proxy locations of the service provider. It is stored at a logical central instance[3]. This instance has a similar functionality like the mapping system. Yet, only proxies are allowed to send encrypted requests.

[3]The implementation, however, could be Distributed Hash Table (DHT)–based or distributed in a Domain Name System (DNS) root server alike way.

Inbound Connection Whenever a correspondent node wants to send packets to the mobile node—inbound to the mobile node—, it queries the mapping system for the mobile node's locator. The mapping system looks up the UID in its database and retrieves the mapping entry. Due to a set flag the mapping system is aware that the mobile node is using a privacy service. Based on the correspondent node's locator, it selects a care–of locator from the mapping entry which has the smallest topological distance to the correspondent node. This care–of locator is returned in the location response message.

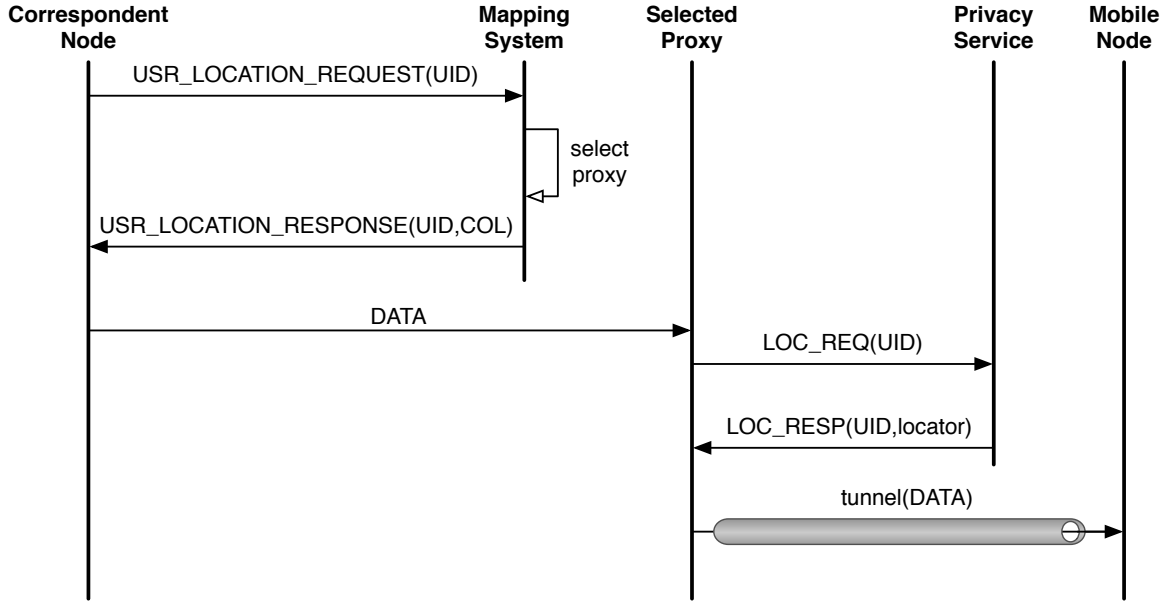


Figure 5.8.: Message flow of the mapping system based proxy selection.

As the privacy service is transparent to the correspondent node, the node sends the data packet towards this care–of locator. The proxy receives the packet and checks the enclosed destination UID against its table of open connections. In case no such connection is found, the packet is cached, and the proxy queries the privacy service's central instance. This message and the central instance's one are encrypted. Signatures assure that only the service's own proxies are allowed to request location information. After having received the mobile node's locator, the proxy forwards the data packet to the mobile node. Figure 5.8 shows the complete message flow.

Outbound Connection Whenever the mobile node wants to contact a peer, it has two possibilities. It can either send direct messages, thereby concealing its location or again use the privacy service. As the latter is the more interesting (although possibly the rarer) case, it is described in the following.
As usual, the mobile node is required to query the mapping system for its correspondent node's locator. In a next step, the mobile node requires a list of all the proxies belonging to its privacy service. This list can be downloaded and cached from the

privacy service. Similar to the mapping system, the mobile node calculates the proximity between the correspondent node's locator and the care–of locators from the list. The closest one to the correspondent node is selected and a service request sent. Afterwards, data packets are tunneled to this proxy. The service request includes the mobile node's legitimacy to use the privacy service and its peer's locator. The legitimacy is proved by a token signed by the privacy service.

In case the correspondent node is using a privacy service itself, the data packets need to pass two proxies. Furthermore, proxy selection must be done intelligently in order to not increase triangular routing. The proposed mechanism, however, handles this issues quite elegantly. A mobile node querying the mapping system for the locator of a correspondent node, which as well uses a privacy service, is returned a care–of locator as close as possible to its own location. At this point, it is important that the mapping system based privacy service mechanism is transparent to the querying end system. Hence, the mobile node assumes that the returned care–of locator is the correspondent node's real location and selects a proxy accordingly.

5.3.3. Feasibility and Performance of the Proxy Service

In order to prove the feasibility of the privacy service, a prototype was implemented. This prototype extends the HiiMap mapping system with the selection functionality, provides the central privacy service and the proxies (cf. Chap. 6.4). The client software was extended to be able to enter the UID of the privacy service. Again, the extended mapping system, the privacy service and several proxies were deployed to the G-Lab experimental facility. The proximity calculation, thereby, was based on the first octet of the client's and proxy's locator (i.e., IPv4 address). This simplification was necessary as today's Internet Protocol (IP) address space is de–aggregated, and the proximity between two complete IP addresses cannot easily be computed. Each site of the G-Lab experimental facility, however, shares the same network address per location. The prototype proved that the mapping system proxy selection based privacy service is a practical approach. To evaluate the added latency due to the mechanism a measurement was carried out. Figure 5.9 shows the measurement results in comparison to the plain HiiMap and DNS ones from Chapter 3.3.3. Please note that the chart has a logarithmic y–axis as the upper bound of the DNS measurements tops $400\,\mathrm{ms}$.

The measurement of the privacy service started with requesting a locator and stopped as soon as the data packet arrived at the mobile node. For this measurement, the correspondent and the mobile node were on the same end system. The values, therefore, include at least the delay from two complete roundtrips to the G-Lab experimental facility and one internal roundtrip (cf. Fig. 5.8). These results, however, heavily depend on the network topology, the roundtrip times between the network elements and the location of the clients and proxies. This means that an overall performance wise conclusion cannot be drawn from these measurement results. In order to gain realistic and statistical average values, a globally operating privacy service and globally distributed clients would be required.

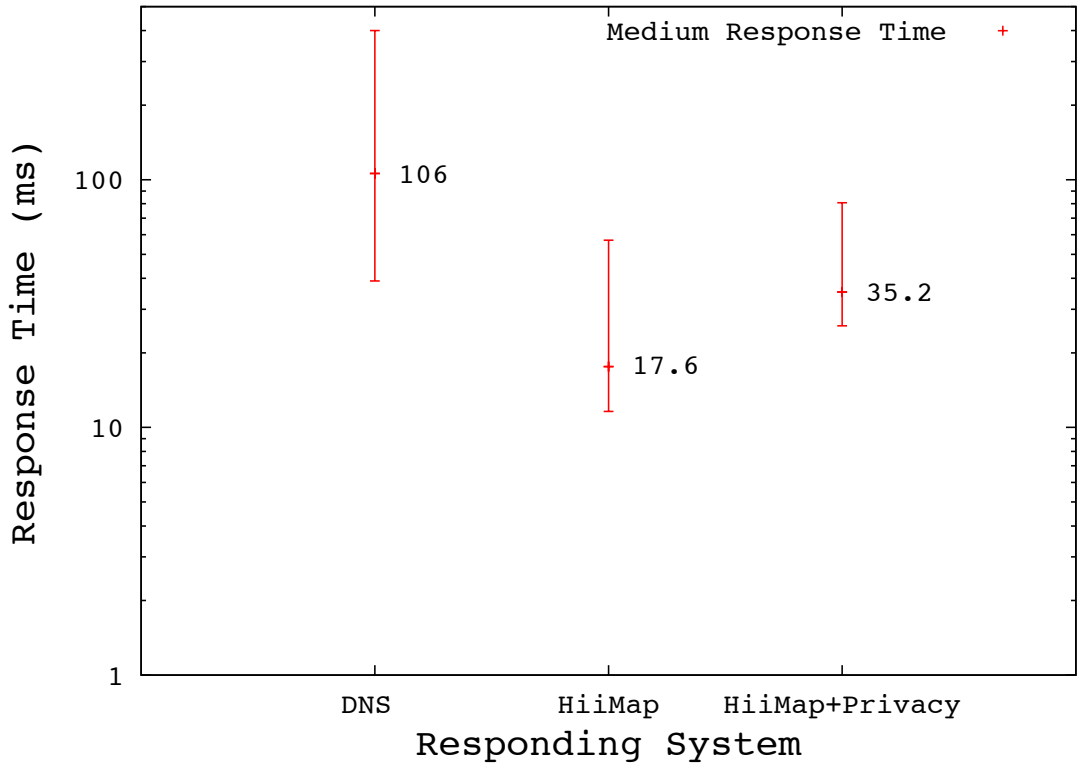


Figure 5.9.: Comparison of medium response times (DNS, HiiMap and HiiMap + privacy service).

5.4. Comparison of Location Privacy Frameworks

All of the introduced location privacy frameworks add latency to a protected connection. The amount of delay, however, depends on various parameters of the implementation. It might even be possible that the delay is not significant to a specific usecase. Comparing different approaches based on the introduced delay, therefore, is not a very good criteria.
In this section, the frameworks are compared based on the overall costs they cause. In that way, not only a single service provider is considered but the costs for the entire network (including the privacy service and internet service providers).

5.4.1. Cost Model

Before evaluating the different frameworks, the cost model for the comparison is described in this subsection. The costs for each framework can be split into two categories—the capital expenses (CAPEX) and operational expenses (OPEX). CAPEX, thereby, are costs which only occur once. In a simplified model the expenses are spent at the beginning, e.g., hardware and network equipment. Contrary, OPEX identifies any costs which accumulate over time. Examples are the electricity bill and maintenance costs. For the sake of simplicity any financial considerations, such as interest

rates of held back CAPEX funds or required loans for one–time spendings, are neglected. Table 5.3 lists all considered expenses for this model. In the following, RVA and FA are simply referred to as *agent*.

CAPEX	**OPEX**
• Proxy/agent servers	• Location/building rental
• Network equipment	• Rack rental
	• Internet connectivity
	• Electricity
	• Transit fees
	• Staff
	• Spare parts

Table 5.3.: Expenses considered in the cost model of the location privacy frameworks comparison.

The aim for the cost model is to evaluate and compare the total costs of the frameworks. The costs are calculated based on a monthly billing period. As CAPEX are a one time investment, a linear project of the expenses over a three year amortization period is used. The rest value of the equipment after that period is $\$0$. To calculate the total cost (C_{total}), a monthly share from that three year CAPEX projection is taken. Analogically, all items of OPEX are monthly fees. All parameters required for the cost model are listed in Table 5.4.

Name	Description	Range
C_{total}	The overall cost for the system	$0 - \infty$
C_{OPEX}	Operational Expenses	$0 - \infty$
C_{CAPEX}	Capital Expenses	$0 - \infty$
N	Total number of nodes	$0 - \infty$
α	Percentage of nodes using the privacy func.	$0 - 1$
β	Subset of N requiring full privacy (penalty)	$0 - 1$
ω	Transit cost multiplier	$0 - \infty$

Table 5.4.: Summary of the parameters for the cost model.

The capital expenses C_{CAPEX} can be calculated as follows (Equation 5.1).

$$C_{CAPEX} = C_{gateway}(\alpha \cdot N) + C_{net.\,equip.}(\alpha \cdot N) \tag{5.1}$$

$C_{gateway}$ in the equation reflects the costs of the server hardware on which the agent or proxy software is running. The costs are dependent on the number of users requiring privacy protection. Equivalently, $C_{net.\,equip.}$ reflects the costs of the required network equipment. For this calculation, the deployment of open–source software is

assumed, thus, causing no CAPEX and only maintenance staff costs (which is reflected by OPEX).
As shown in Table 5.3, electricity, rack rental, Internet connectivity and building/location rental are parts of the operational expenses. To be able to acquire accurate numbers for these costs, these entries are subsumed into a single value. Server hosting can be booked on a monthly basis and covers all of these operational expenses. $C_{co-location}$, dependent on the number of users, reflects this value. Furthermore, OPEX consists of additional transit fees ($C_{transit}$) caused by the triangular routing problem. This value is again dependent on the number of users requiring privacy protection and the transit cost multiplier ω which determines how many times transit fees have to be paid additionally for a single connection flow. C_{staff} and C_{spare} cover the costs of the maintenance staff and spare parts, respectively.
The HIP location privacy framework does not provide full location privacy. Nodes can still be tracked based on protected areas. To compensate for the lack of location privacy in terms of costs, a penalty would be applied. The penalty would be calculated by determining the additional costs which are generated by providing full privacy for a subset of nodes (β). For this thesis, however, the penalty is not factored in. Authentic numbers for β and the caused costs are quite difficult to determine. Instead, based on the author's suggestion [MSS+06], the location information leakage is minimized for this approach by assuming the protected areas to be very large. The consequence is that the FAs need to be placed in the core network. Equation 5.2 shows the OPEX calculation.

$$\begin{aligned} C_{OPEX} = {} & C_{co-location}(\alpha \cdot N) + C_{transit}(\alpha \cdot N, \omega) \\ & + C_{staff}(\alpha \cdot N) + C_{spare}(\alpha \cdot N) \\ & + C_{penalty}(\beta \cdot N) \end{aligned} \tag{5.2}$$

By adding up OPEX and CAPEX, the total cost (C_{total}) as shown in Equation 5.3 can be calculated.

$$C_{total} = C_{CAPEX}(\alpha \cdot N) + C_{OPEX}(\alpha \cdot N, \omega, \beta) \tag{5.3}$$

5.4.2. Case Study for the Comparison

The numbers taken for the comparison represent the average prices as of mid 2011 and not any forecast. Even though the privacy solutions aim at Next Generation Internet architectures, the numbers might change drastically, but the relations can be expected to stay the same.

Capital Expenses

For all frameworks, the functionality—and, therefore, the costs—of the agent and proxy servers are basically the same. The agent/proxy receives packets, initiates a lookup in its internal database and forwards the packet to its destination. It, therefore, can be compared to the functionality of today's network address translation (NAT)

boxes. To calculate the required computational power for the agents and proxies, the capabilities of today's NAT boxes are taken.
The Leibniz Supercomputing Centre in Munich, Germany [Lei12], provides the network infrastructure for three universities and affiliated student housings. The network is used by around 40,000 users which tend to generate more traffic compared to the country–wide average. Most of the traffic has to pass the NAT boxes before being forwarded to the Deutsches Forschungsnetz (DFN). A single machine serves up to 2,000 users. The servers are standard 2U systems with the following specs: 2x DualCore Opteron $2,5\,\mathrm{GHz}$ and $16\,\mathrm{GB}$ RAM. Each machine is serving a one gigabit connection (utilization $500 - 600\,\mathrm{Mbps}$). The servers are running a Linux system and cost about $2,000.
To calculate the capital expenses, it is assumed that the servers are hosted in a 42U rack where 2U are reserved for the network equipment. This means that 20 servers can be hosted in a single rack serving up to 40,000 users. The network equipment consists of a single switch per rack and the estimated costs are $2,000 as well. According to the cost model, projecting CAPEX over a three year amortization period is required. The costs of the servers and network equipment for a single rack ($C_{gateway} + C_{net.equip}$), therefore, total at $1,167 per month per 40,000 users.

Gateway Operational Expenses

Co–location offers for a single 42U rack including electricity, A/C and Internet connectivity are around $1,500 per month [EGI10] ($C_{co-location}$). For the maintenance staff, the fully allocated costs of an experienced single Full Time Equivalent (FTE), which is $180,000 per year [Pay10], are assumed. To maintain and service a full rack, 75% of a FTE is required. This results in $11,250 per month and 40,000 users (C_{staff}). To estimate the costs for spare and service parts, 15% of the monthly CAPEX are taken ($C_{spare} = \$175.05/40,000\,\mathrm{user}$).

Transit Costs

To obtain realistic numbers for the transit fees is quite complicated. The current (mid 2011) rate charged, based on the 95 percentile, is $3.25 per Mbps [Nor10]. The challenge, however, is to estimate the average traffic caused by each user and how often additional transit has to be paid because of the location privacy protection mechanism (i.e., side effect of triangular routing).
The average traffic caused by a single user is the same for all frameworks. To obtain real world numbers for the traffic per user, the LRZ's Eduroam subnet was monitored for a week in April 2011. The wireless Eduroam access network is used by students and researchers with mainly mobile devices like laptops, tablets and smart phones [TER12]. The traffic pattern, therefore, suits the use case of mobile nodes requiring location privacy protection. During the week in April, the out- and inbound traffic of the Eduroam subnet and the number of users as seen by the access points was recorded. The recordings are shown in Figure 5.10.

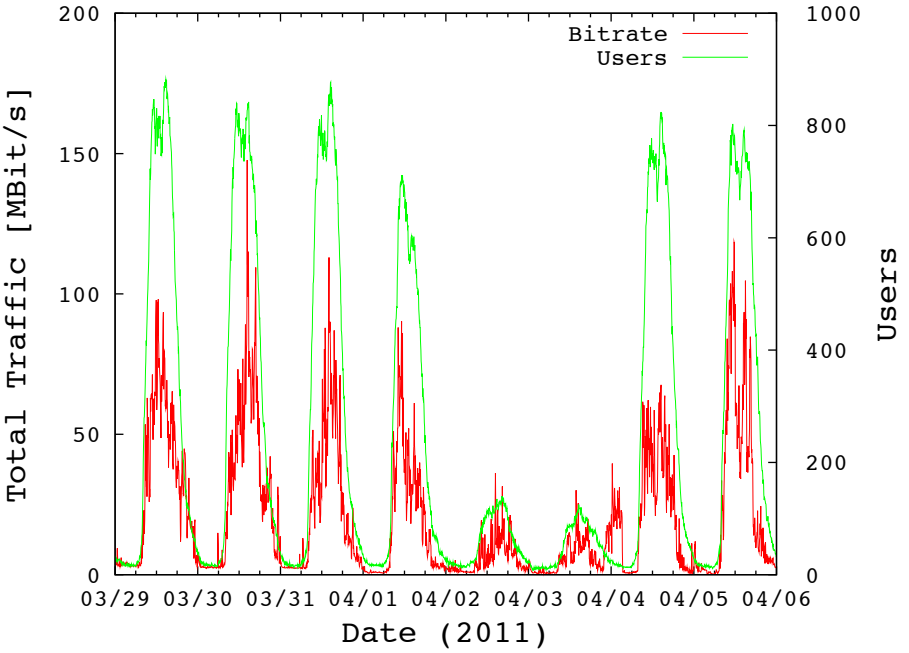


Figure 5.10.: Total amount of used bandwidth (green) and number of users (red) in the LRZ Eduroam access network.

Transit costs are usually charged based on the 95 percentile[4] [Nor10]. Figure 5.11 shows the average used bandwidth per user over the week. The green line indicates the 95 percentile which is at $0.244\,\mathrm{Mbps}$ per user.
At this point, the transit costs per user are as shown in Equation 5.4.

$$C_{transit} = \$3.25 \cdot 0.244\,\mathrm{Mpbs} \cdot \omega \tag{5.4}$$

ω, thereby, reflects how many times additional transit fees have to be paid. This value differs for each framework and is dependent on several aspects. These aspects are the average path length (in terms of AS), the proximity between the correspondent and mobile node and the positioning of the agent or proxy, respectively. The challenge, thereby, is to estimate how many connections, rerouted via the agent/proxy, cause transit fees in addition to the ones already caused by the direct path between the two peers. As no serious number can be given, the result shown in the next subsection varies this parameter. In order to get an understanding of the range, however, a simplified model is used. This model takes two frameworks into account—the HIP location privacy framework and the proxy–based privacy service. For BLIND (including enhanced BLIND) no information is given as where to position the FAs. Additionally, the authors of the framework do not specify whether a single FA per mobile node must be used or the usage of multiple ones is possible. For the remainder of this comparison, therefore, BLIND is treated equally to the HIP location privacy framework. The location of the RVAs and FAs is from here on only referred to as location of the

[4]The 95 percentile states that 95% of the time (e.g., during a billing period) the traffic was below this amount. It is taken to cut off bursts like the one shown in Figure 5.11 for the 4th of April.

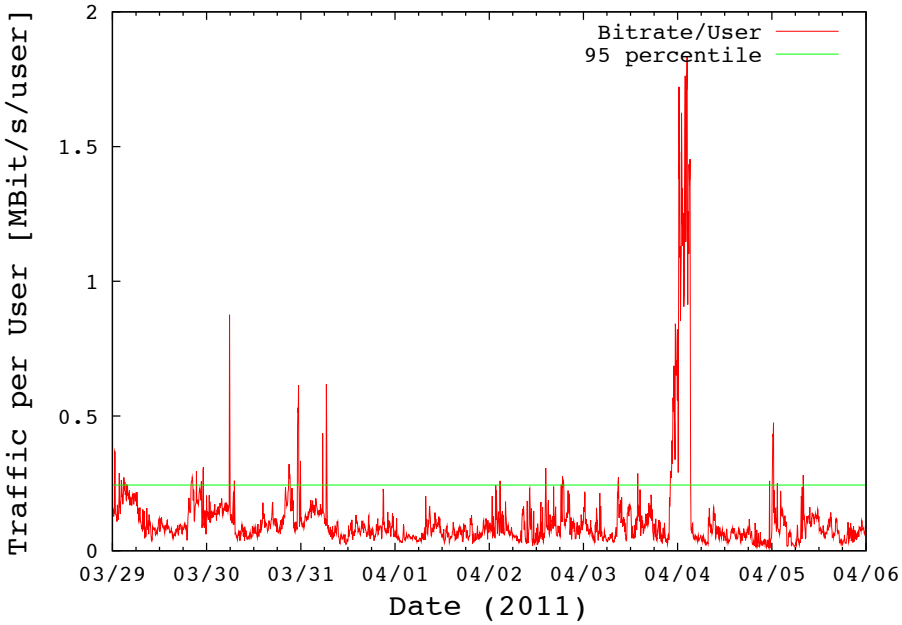


Figure 5.11.: Bandwidth per user in the LRZ Eduroam access network.

agent. It should be noted, however, that enhanced BLIND might be extendable with an intelligent proxy selection as proposed for the HiiMap privacy service. In that case, the positioning of the FAs would compare to the ones of the proxies from the privacy service framework.

The simplified network model is depicted in Figure 5.12. The network in the model consists of three tier levels. For the sake of simplification and to estimate worst case scenarios, the source and destination of a communication flow are always located at the tier 3 level. In case a direct peering between two tier 3 providers is present (e.g., ISP D and E), no transit fees are caused by the communication flow. If no peering exists, transit links to higher tier providers must be used and fees are charged. This simplified model is not too far fetched as Ripe NCC states in their statistic report [NCC10] that the average BGP path length includes 5.02 ASs.

The HIP location privacy framework requires all traffic to pass the core of the Internet. This is regardless of the location of the source and destination of a communication flow. As a result, each connection requires to pass four transit links, even if both communication partners reside within the same AS. This is due to the requirement to design large protected areas to keep location information leakage as small as possible (even though it still occurs). In a best case, the source and destination are already connected to a tier 1 provider, and no additional transit fees need to be taken into account. The worst case, however, requires four additional transits to be paid for ($\omega = 4$). This is the case whenever the source and the destination are connected to the same or different but interconnected tier 3 providers. As any traffic needs to pass the core (which means a tier 1 provider), no direct free–of–charge peerings between two providers can be utilized, and additional charges need to be paid.

For the privacy service in a single proxy scenario, the best and worst case in terms

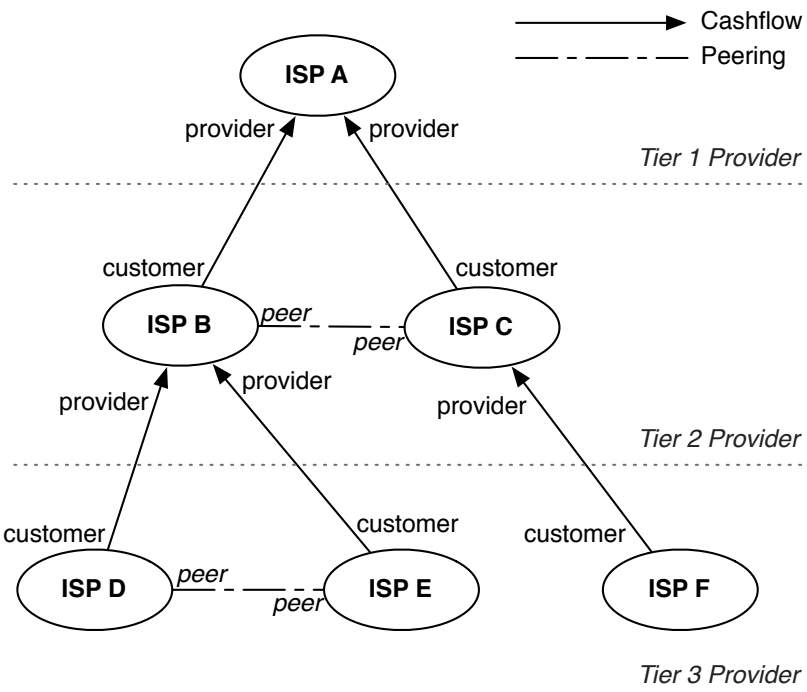


Figure 5.12.: Transit fee cashflow. Fees need to be paid between customers and providers. No fees are charged for peerings within the same tier level. Peerings in the tier 1 level are always free of charge [The11].

of required transits is the same compared to the HIP location privacy framework. By providing multiple proxies and employing intelligent proxy selection as introduced in Section 5.3.2, however, the number of required transits (ω) can be decreased. Figure 5.13 illustrates an example in which the selection of a proxy nearest to the direct path between the two peers [5] avoids any additional transit costs. The agent–based approach in this example includes four transit links.

In case no direct peering between ISP D and E in Figure 5.13 would exist, the peering between ISP B and C would be used. The selected proxy would remain number 3 as it is closest to the correspondent node. To evaluate the cost model with a realistic number ω for the proxy approach, detailed knowledge about how much traffic remains in the local AS, is destined for a neighbor AS (with direct peering) or is addressed to a far away AS would be necessary. This information, however, is not publicly available for either confidentiality reasons or because no widespread measurements have been carried out so far. In the cost evaluation, therefore, a range for ω from zero to four is used to evaluate the decrease in cost with each less required transit connection. Zero required transit links, thereby, are not unrealistic as content provider try to push the content as close as possible to users via mechanisms like content distribution networks or content caches.

[5] For privacy reasons located as close as possible to the source.

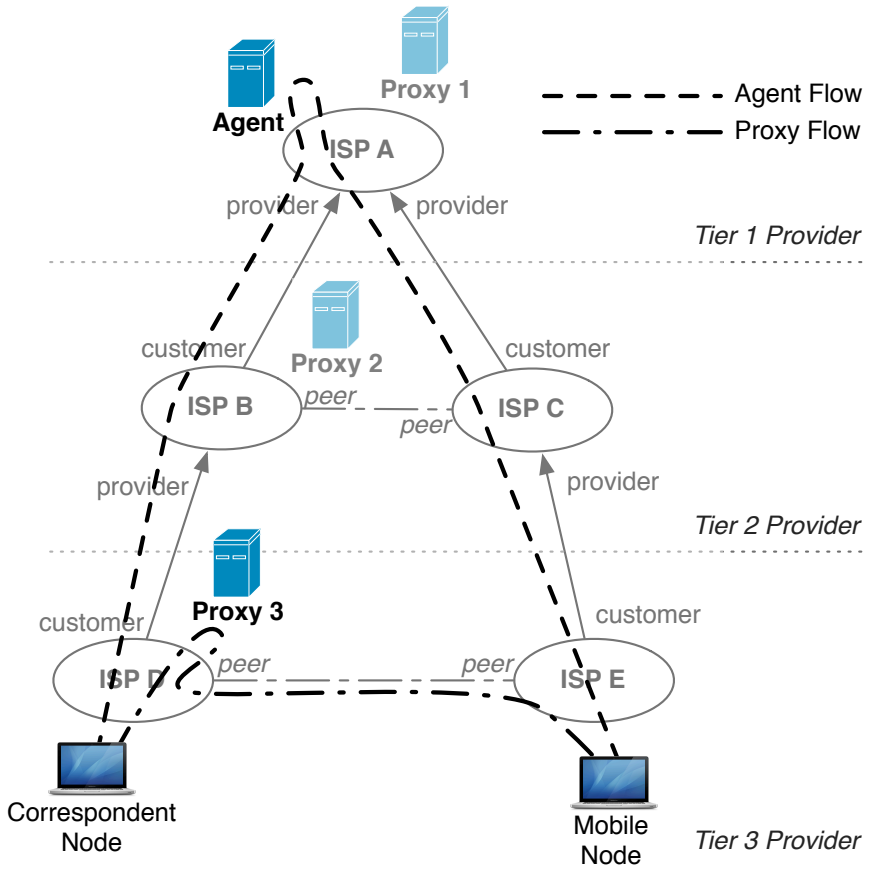


Figure 5.13.: Communication path for location privacy mechanisms.

Cost Evaluation

By substituting the given values in the cost model (equation 5.1, 5.2 and 5.3) the results as shown in Figure 5.14 can be obtained. For the number of total users (N) four billion was chosen. This roughly represents the amount of nodes as seen of today [Hus10]. The parameter α on the x–axis reflects the percentage of all nodes using the privacy protection function. The red line shows the overall costs for the agent–based framework as well as the privacy service with a single proxy (this means four transit links). The grey lines represent the privacy service with proxy selection enabled (i.e., less additional transit links).

The graph shows that the overall costs of the HIP location privacy framework sum up to $\$7.05\,\text{billion}$ per month for a ratio of 50% of mobiles nodes. By only decreasing the number of required transit links by one (and thereby the fees), this monthly bill goes down by $23.66\,\%$ to $\$5.46\,\text{billion}$. Requiring no additional transit at all, the overall costs decrease to $\$0.7\,\text{billion}$ per month. This shows the significant impact of the transit fees on the overall costs. Figure 5.15 shows the costs itemized for a single privacy service

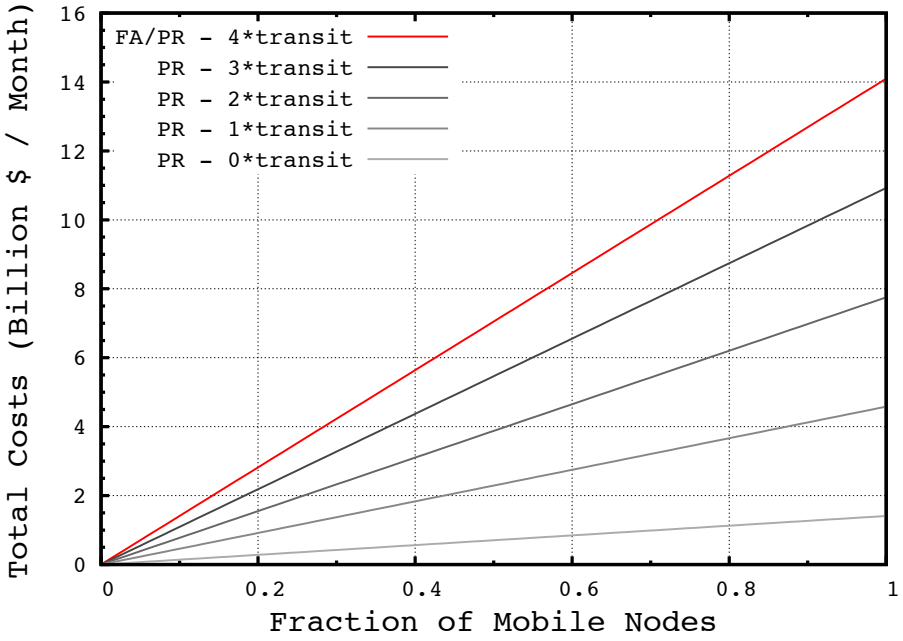


Figure 5.14.: Total costs of the location privacy mechanisms based on the number of users and additional required transit connections.

subscriber depending on the required transit links, to further illustrate this relation[6].
The comparison shows that the transit fees largely outgrow the gateway costs. In case the average amount of generated traffic increases faster than the transit fees decrease, this misbalance even worsens in the future. The conclusion which can be drawn from these results is that an intelligent proxy selection is very important. From a financial point of view it is beneficial to provide a lot of proxies in order to keep the caused transit fees to a minimum.
The HIP location privacy framework does not offer a possibility to select proxies. It, therefore, causes the maximum amount of additional transit fees. For the simplified network model used in this comparison the worst case is four additional transit links in the path from and to the FA. For this model, the monthly costs caused by the HIP location privacy framework sum up to $3.52 per month and user (cf. left bar in Fig. 5.15).
Contrary to the HIP location privacy framework, the privacy service introduced in this thesis provides full location privacy and allows to intelligently select a proxy individually for each communication flow. This means that the costs can be kept lower while at the same time providing more flexibility and privacy. Beside the financial advantage of this approach the impact of other triangular routing related side effects (e.g., delay) can also be lessened by the ability to select proxies.
Enhanced BLIND might also be extendable with a proxy selection mechanism as introduced in this thesis. Similar to the privacy service, it also guarantees full location

[6]Please note that these costs only include the values from the introduced cost model and do not factor in additional economic figures like marketing costs, desired asset gain, etc.

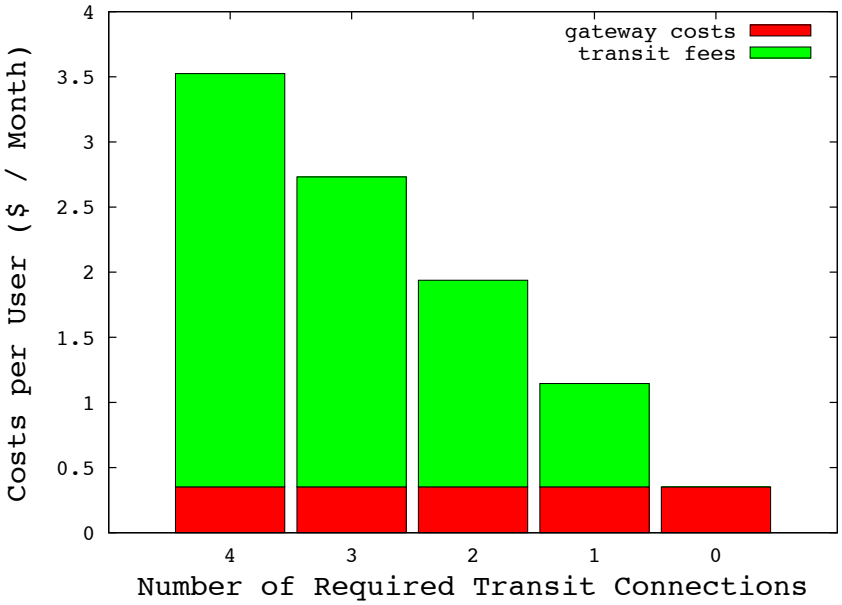


Figure 5.15.: Influence of the required transit connections on the total costs per user.

privacy. It, however, has the disadvantage that mobile nodes, respectively any node which requires privacy protection, is not reachable by other peers. It is only possible for these nodes to initiate communication flows but not be the destination of a connection request.

5.5. Conclusion

The locator/identifier split addressing scheme solves many challenging aspects of today's IP–based Internet architecture. As a downside, however, it is possible to track a node which uses this addressing scheme. This location privacy problem is an important aspect which must be solved in order for any locator/identifier split based architecture to be considered as a successor to today's Internet.
In this chapter, the location privacy problem in regard to the locator/identifier separation paradigm was outlined and existing proposals to overcome this problem briefly discussed. These proposals, however, do either not provide full location privacy or their mechanisms contradict the flexibility and connectivity desired of a NGI architecture. Subsequently the HiiMap privacy service was introduced. This framework is based on proxies which enable mobile nodes to conceal their topological location while at the same time being reachable by correspondent nodes. As any of the described approaches (including the HiiMap privacy service) introduce negative side effects to a communication flow—caused by triangular routing—, a mechanism was presented to lessen the impact of these side effects. The so–called intelligent proxy selection allows to select a proxy individually for each communication flow and, thereby, to choose one which is located as close as possible to the direct communication path between two peers.

Three proxy selection mechanisms for the HiiMap privacy service were discussed and compared with each other. The best one, i.e., mapping system based proxy selection, was chosen and compared with the other location privacy frameworks. Rather then focusing on delay, the comparison was based on financial costs of each mechanism. The result underlined the significance of transit fees for the overall costs of the location privacy solutions. This showed that an intelligent proxy selection mechanisms is highly advised to be employed in a location privacy service.

6. Realization Within the G-Lab Distributed Experimental Facility

In this chapter, the prototypical implementations of the Hierarchical Internet Mapping Architecture (HiiMap) are briefly discussed and their elements described. An extensive description of all software components and details, however, would exceed the page limit of this thesis. For each component, therefore, only a functional description and notable aspects are discussed. A prototype of the HiiMap mapping system was shown as part of a joint demo at the Euroview Future Internet Workshop 2011[1].
First, the G-Lab experimental facility, to which the most components were deployed, is introduced. Afterwards, the prototype of the mapping system and its security and privacy extension are described.

6.1. G-Lab Experimental Facility

The G-Lab project is a research project funded by the Bundesministerium für Forschung und Bildung (BMBF) [TG09]. Its main objective is to research and develop Next Generation Internet (NGI) concepts and architectures. One part of the G-Lab project is a distributed experimental facility. This facility provides the researchers with the ability to test and evaluate novel protocols and architectural elements.
The initial testbed consisted of 175 physical nodes distributed over six sites in Germany. Each site, thereby, hosts 25 nodes except the management site which operates 50 ones. Those six locations are the initial G-Lab partners and consist of the following universities: Julius-Maximilians-Universität Würzburg, Technische Universität Kaiserslautern, Technische Universität Berlin, Technische Universität Darmstadt, Technische Universität München and Karlsruher Insitut for Technology. In a second phase, the testbed was extended to 190 nodes, spreading over 13 locations. The first version of the experimental facility was initiated in January 2009.
All phase one sites (and several of the second phase sites) are interconnected by the Deutsches Forschungsnetz (DFN) which provides a backbone for academic and research facilities within Germany. The backbone only carries traffic from these institutes and does not face a normal load as commercial internet service provider. As a result, the roundtrip time between the G-Lab sites ranges from seven to ten milliseconds (cf. Chap. 3.3.3). The DFN supports IPv4 and IPv6 routing in its backbone. Not all G-Lab sites, however, offer IPv6 connectivity.
Each physical machine is virtualized in order to provide more flexibility and increase the amount of available nodes. The initial virtualization software, thereby, was a mod-

[1] *Deployment of Application–Tailored Protocols in Future Networks*, Proceedings of the 11th Euroview Future Internet Workshop, August 2001

ified PlanetLab [TPU03] version. PlanetLab does only provide a semi–complete virtualization which means that network resources are shared and layer 3 protocols cannot be modified. In order to support experiments with different layer 3 protocols, Proxmox virtualization software was deployed. For the HiiMap prototype, however, the PlanetLab installation was sufficient and, therefore, a description of the Proxmox virtualization is omitted (more details can be found under [TG09]).

6.2. Mapping Service

The HiiMap NGI architecture is based on the locator/identifier separation paradigm. This novel addressing scheme is intended to replace the current layer 3 protocol (IPv4 and IPv6 respectively). The Planetlab virtualization software, however, does not provide support for other layer 3 protocols beside the Internet Protocol (IP) without the use of tunneling. To overcome this obstacle, the network stack of each HiiMap component, except the topology monitor, was extended with a shim layer. This shim layer is located between layer 3 and 4 of the OSI reference model. It contains addressing based on identifiers while layer 3 provides network connectivity based on IP as usual. To this end, the IP addresses represent the locator of the locator/identifier separation paradigm. In that way, a locator/identifier split based addressing scheme is emulated towards higher layers. Figure 6.1 illustrates the extended network architecture.

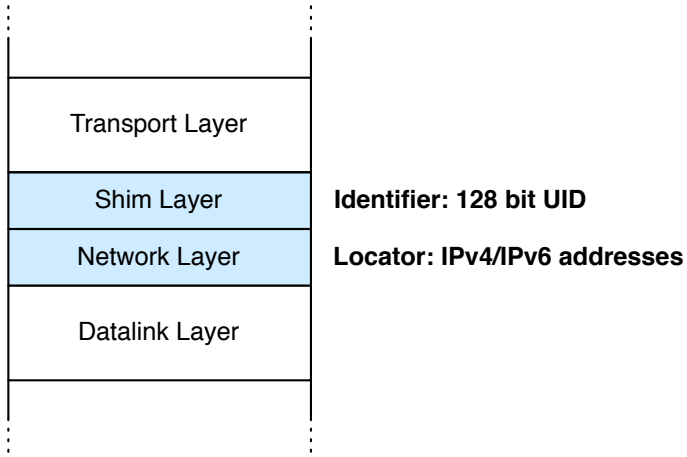


Figure 6.1.: Shim layer employed in the HiiMap component's network stack to emulate a locator/identifier split addressing scheme.

The prototype of the HiiMap mapping system consists of four components. These components are the Global Authority (GA), the mapping regions (consisting of Distributed Hash Table (DHT) peers and the load balancer), a client software and a topology monitor. These components will be described subsequently.
All of the components, except the topology monitor, are implemented in Java. The reason behind the programming language choice is the flexibility it provides. Com-

ponents of the mapping system can be executed on various machines and operating systems without the need to recompile them. This aspect was especially important as in the beginning of the G-Lab experimental facility it was not certain that the PlanetLab software would be used during the complete project lifetime. Java, therefore, was preferred over C++ even though C++ might have resulted in a prototype with faster execution times. To measure execution times, however, was not the main objective of the prototype.

6.2.1. HiiMap Protocol

Each component of the HiiMap mapping system contains a full HiiMap protocol stack. The protocol stack is designed as an independent Java package which can be linked to the program code of each component. The core of this protocol package is built by the HiiMap header class. It defines the protocol header which is equal for each protocol message. Its structure is depicted in Figure 6.2. The *ver*–field defines the protocol version of the packet and *cmd* the payload type. *length* states the payload length and *cs* is a one byte checksum to verify the packet.

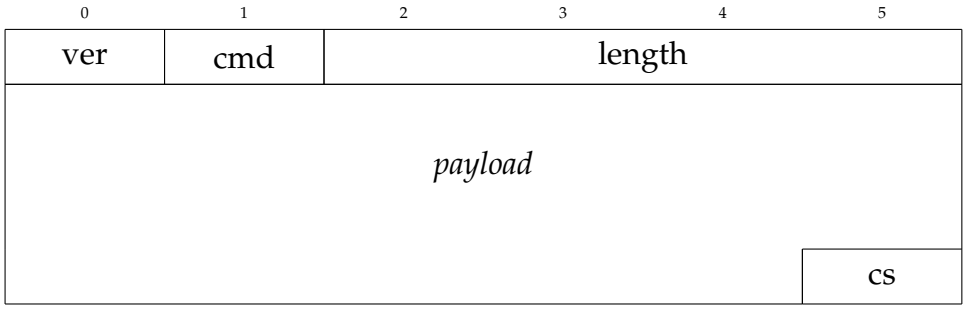


Figure 6.2.: Structure of the HiiMap protocol header.

The HiiMap header class provides a getter and setter method for the payload. Each payload class, thereby, implements a *payload_interface*. In that way, the source code of the packet handling modules are not required to know the specifics about the payload types. The payload classes can be handled in a generic way up to the point where information from the payload is processed. To cast the generic payload class to a specific one, the command code byte in the HiiMap header class is interpreted. This can be seen in the sample client implementation in Listing A.1. To this end, the HiiMap network components follow the design pattern of coding against an interface rather than an implementation. The result is a very flexible protocol stack which can easily be extended. This property of the prototype was a great asset when it was extended with the security and privacy functionality.

6.2.2. Global Authority

The GA is implemented as a single instance although in the concept of HiiMap a redundant design is suggested. This could either be realized by sharing resources of

multiple machines or replicating and synchronizing an instance similar to the Domain Name System (DNS) root servers. The prototype, however, was never expected to face an enormous load and to provide high reliability as required from a productive system.
One of the initial design decisions, therefore, was to use SQLite as database. During stress tests of the mapping system, however, this database system turned out to be unable to cope with even a moderate load. Update processing times became increasingly high whenever the database grew to store several 10,000 entries. After a while it clogged completely, and update as well as retrieval requests were dropped.
During stress tests, multiple badges of 5,000, 10,000 or 20,000 entries were pushed into the mapping system, requiring the GA to store the Unique Identifier (UID) to region mapping. Whenever a new UID was registered with a region, the region would send an update to the GA in order to store the $\langle UID\, region \rangle$–tuple. Over time, multiples of 10,000 entries accumulated in the GA as it was not restarted between the tests. As a consequence, the database system was changed to H2. Similar to SQLite, it is an inline database were the database management functionality is linked into the program code of the GA. A stand–alone database system (e.g., MySQL) was not chosen in order to keep the prototype flexible. In that way, initiating a new instance of the GA on a different machine does not require to set up a database system.

6.2.3. Region

The region is defined by two components. These are the load balancer and the DHT peers. The core of the peers is formed by the Event Detection and Reporting Algorithm (EDRA) of the D1HT protocol [MA06]. This algorithm distributes information about the DHT ring structure among the peers and ensures its stability. Whenever an event is detected (e.g., a peer joins or leaves), this information is distributed to the other peers of the DHT. Such an event is detected by sending *join* and *hello* messages to a peer's successor. A hello timer is started after a *hello* message is received. In case no such message is received before the timer expires, the predecessor is assumed to have left the DHT. In order to factor in packet loss and delay, the timer is set to three times the interval of the periodic *hello* messages.
Beside keeping the DHT ring structure intact, a peer must although store its share of mapping entries. A peer, thereby, is required to store entries which fall into the key subspace it is responsible for. In addition, it also stores entries of its predecessors in the key space. The reason for this is to achieve resilience within the DHT. When disconnecting from the DHT, a peer hands over its entries to its successor which then extends its responsible subpart of the key space. In case, however, the peer fails and is not able to transfer its mapping entries, these entries would be lost. Therefore, backup copies are stored at multiple successors.
In the prototype implementation, each peer maintains two data tables. The first one is used to store its own mapping entries. The second one stores the backups from other nodes. An own algorithm is used to keep backups synchronized and organize responsibilities of key subspaces in case a node joins or leaves the DHT. Whenever a

peer receives a mapping request (i.e., `USR_LOCATION_REQUEST`), it first determines whether the enclosed UID falls into its own key subspace or not. In case it does, the query is resolved from the data table of own mapping entries (assuming an entry for the UID exists). In case, however, the UID does not map to the own subspace, it usually is forwarded to the peer in the DHT which is responsible. In order to speed up the lookup process, it is checked whether the UID maps to the backup key subspace or not before forwarding the request. In case the query can be answered from the backup data table, it is not forwarded and the query is responded to directly. This, however, requires the backup entries to be up–to–date at any time. The impact of this respond from backup mechanism can be seen in the performance evaluation (cf. Chap. 3.3.3). Figure 3.15 shows that approximately 3/5 of location requests could be answered without the need to forward the request within the DHT. This is because the replication factor—the amount of predecessor key subspaces are stored as backup at a single peer—was set to two during the performance tests. This means that each peer stored three datasets of mapping entries (its own one, the one from its direct predecessor and the ones from its predecessor's predecessor). The replication factor within the DHT, however, can be configured.

The core of the peer's software implementation is a message flow architecture. Each information unit (e.g., network packet, database request, database object) passes the main messages queues. These queues are also coded against interfaces and, therefore, able to handle new information units not known to the system during its design. Each information processing module is able to register for specific information units (e.g., network stack for network packets). Once an information unit appears in one of the main message queues, all modules which registered for this specific information type are notified—the architecture, thereby, follows the observer design pattern. In that way the software can be extended with additional modules in order to fulfill new tasks. The resilience/backup module, for example, was not included in the initial version and added later one.

6.2.4. Client

Although there is only a single mapping system version, three different types of clients were developed. Each one serving a different purpose. A fourth version was developed at the Karlsruhe Institute of Technology and used in a joint demo held at the Euroview Future Internet Workshop 2011 (based on [WHSM10]). This fourth client, however, is not further detailed in this section.

Performance Test

The first set of clients are command line based tools to test and verify the mapping system. Among protocol tests these clients were used to run stress tests against the mapping system. The performance measurements presented in Chapter 3.3.3 and 5.3.3 were carried out by these clients.

Graphical User Interface

The second type of client has a graphical user interface (GUI) and was used as a demonstrator. Figure 6.3 shows the GUI of the client. It allows a user to enter or randomly generate a UID. After setting a region number, the UID is registered with the selected region of the mapping system. Please note that region selection is done by the client for demonstrator purposes. In a real world deployment, the region would be determined by the user's provider.
Beside registering a UID, the user is able to initiate UID lookups. This can be done by the text box and button at the top of the GUI. In Figure 6.3, a lookup for the UID beginning with 30:c1 was sent. This UID was previously registered by the same client (text boxes in the middle section) and, therefore, the locator (in this case an IPv4 address) of the machine the client software was executed on was returned.

Figure 6.3.: Graphical user interface of the HiiMap client demo software.

The three text fields on the bottom of GUI show the clients cache states and a log window. The upper most text field of these three shows which UID→locator mappings are currently cached by the client. The second one displays the region table downloaded from the GA. It includes all region numbers, the locator of the region's load balancer and a timestamp of the last update. It can be seen that the addressing scheme

used for the locator is not limited to IPv4 for the prototype. The GA stores domain names as region locators, for example.
The client can operate as webserver and -client in order to illustrate the locator/identifier separation paradigm. By pushing the "Open Server Window" button a http–server is started. Another client on another end system can request a website from the first client. To request the site, the "Open Client Window" button is clicked and the UID of the first client entered. The returned website contains the first client's UID and actual locator.

Real World Mobility Pattern

The third client version is developed in the Phyton scripting language and executed in the Seattle distributed testbed [Cap12]. The Seattle testbed offers virtual slices to developers on various end systems (stationary or mobile) in order to execute network applications. Those end systems are real world devices and distributed world wide. They can either be workstations in a university/company, privately owned laptops or mobile phones/handheld devices. The HiiMap Seattle client is used to emulate real world mobility patterns. Each client monitors the network connection(s) of its host system. In case an IP change event occurs, a `USR_LOCATION_UPDATE` message is sent to the HiiMap mapping system. During the test phase 150 randomly selected slices on different end systems were assigned to the HiiMap experiment. These clients were all registered with a single region.
Additionally to sending location update messages to the mapping system each client provides a website displaying its UID, current locator and the number of locator changes (cf. Fig. 6.4). During the initial test phase which lasted one week in December 2011, the number of locator changes varied drastically over all clients. It ranged from zero up to 328 changes. The average, thereby, being around 11 changes per device.

Figure 6.4.: Website screenshot of the HiiMap Seattle client.

6.2.5. Topology Monitor

The topology monitor shows all active regions of the mapping system on a map. The user is able to zoom into a region and check the status of its region's peers. The

topology monitor was developed for the joint demo at the Euroview Future Internet Workshop 2011. Contrary to all other components of the prototype, it is developed in Objective-C and executed on a tablet device. A tablet solution was chosen for its flexibility. The joint demo showed a video use case based on application–tailored protocols and virtual networks. These protocols were individually selected and downloaded for each connection. Each protocol block and virtual network, thereby, was registered with an own UID, and storage information could be retrieved from the mapping system. Furthermore, the communication between the different nodes of the showcase was based on a locator/identifier–split addressing scheme. As the showcase required three different stations (protocol design, deployment and usage), the tablet–based topology monitor could be brought along with each station, showing the activity in the mapping system for each one. Figure 6.5 shows the GUI displaying a map of Germany.

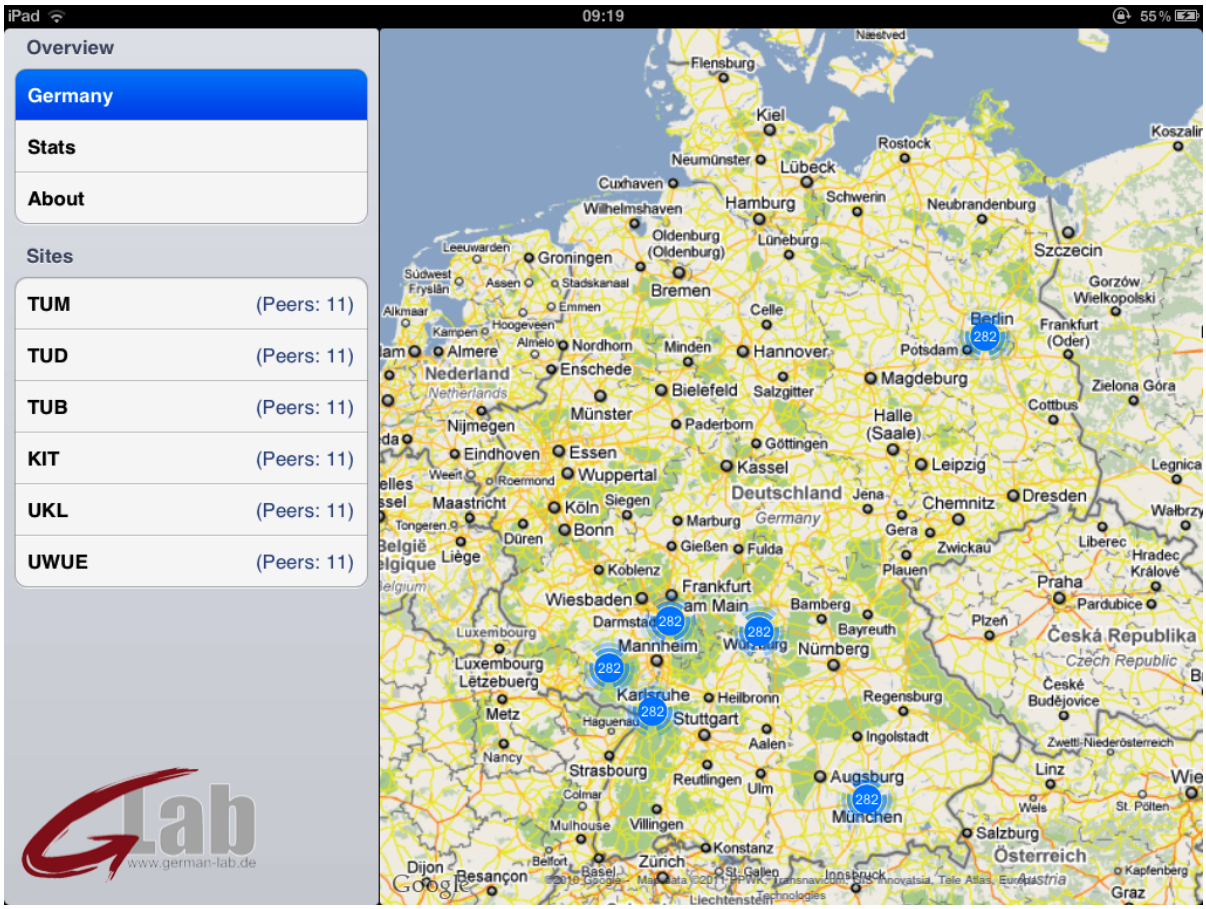


Figure 6.5.: Screenshot of the HiiMap topology monitor in overview mode.

For the demo six regions were initiated whereby each site of the G-Lab phase 1 partners hosted one. Each region consisted of eleven peers in addition to the region's load balancer. The blue circles in the screenshot in Figure 6.5 represent the location of a region. The number on the circle reflects the amount of stored mapping entries per region. The GA is not shown in the overview.

The topology monitor is connected with the load balancers of each region. They provide the monitor with the amount of peers within the region, the number of mapping entries and the activity status of each peer. Figure 6.6 shows the close–up view of a single region.

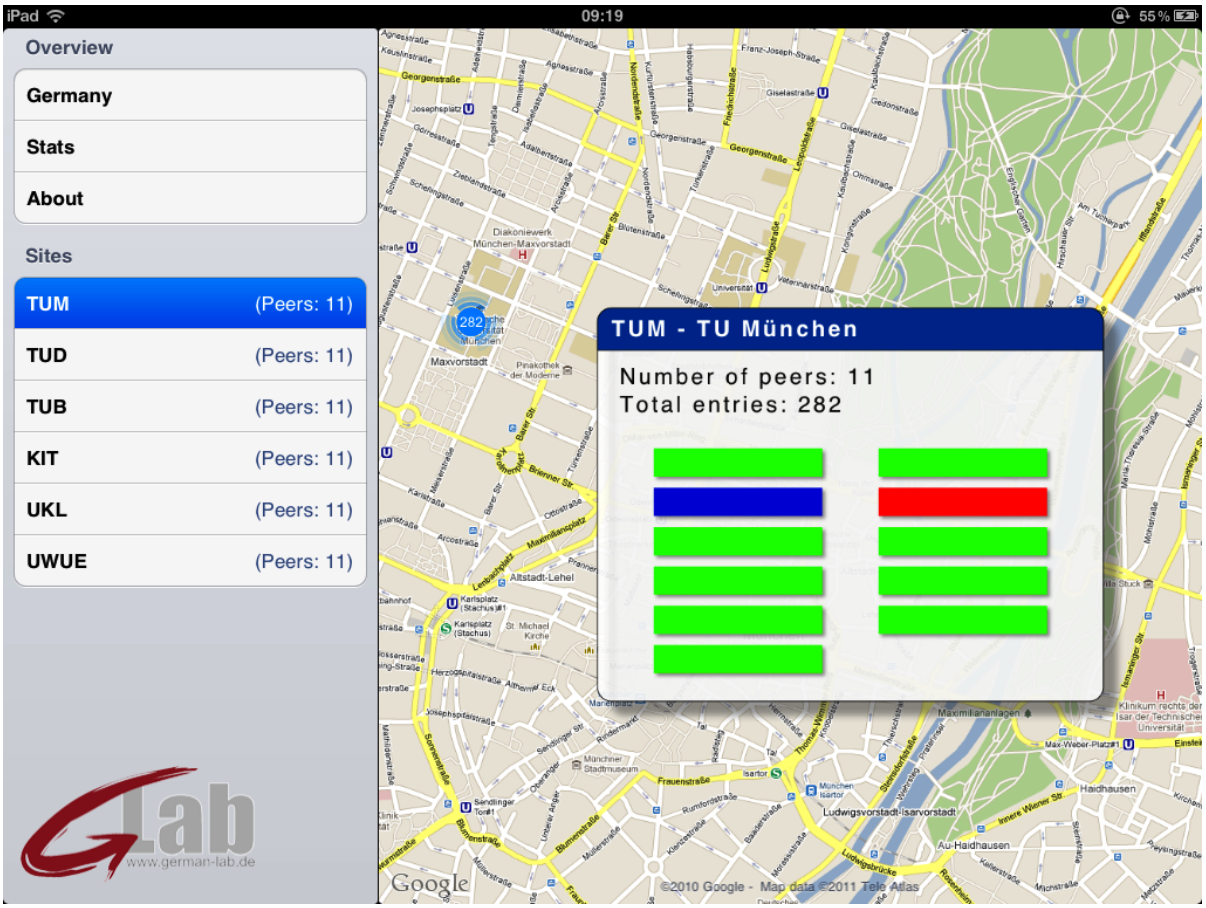


Figure 6.6.: Screenshot of the HiiMap topology monitor in close–up mode.

Each bar in the close–up view represents a peer in the DHT of the region. A green bar signals an online—but currently inactive—peer. Blue indicates that a mapping entry is currently queried, and red shows a peer which received an mapping entry update. It could be shown during the demo that upon deploying or retrieving protocols the mapping system was updated or queried and information was not statically entered at the nodes.
Furthermore, the topology monitor helped to illustrate the architecture and operation of the otherwise non–visible mapping system.

6.3. Public Key Infrastructure

After the initial version of the HiiMap prototype proved the overall feasibility of the architecture, it was extended to incorporate the security framework (cf. Chap. 4).

The complete functionality of the framework, however, is not implemented by the prototype. The client key management, for example, is not based on smart cards, and messages are not cryptographically signed. The objective for the prototype was to prove the practicability of the distributed public key infrastructure. A prototypical client implementation based on smart cards, therefore, would not have added much value in regard to the costs of such a development.

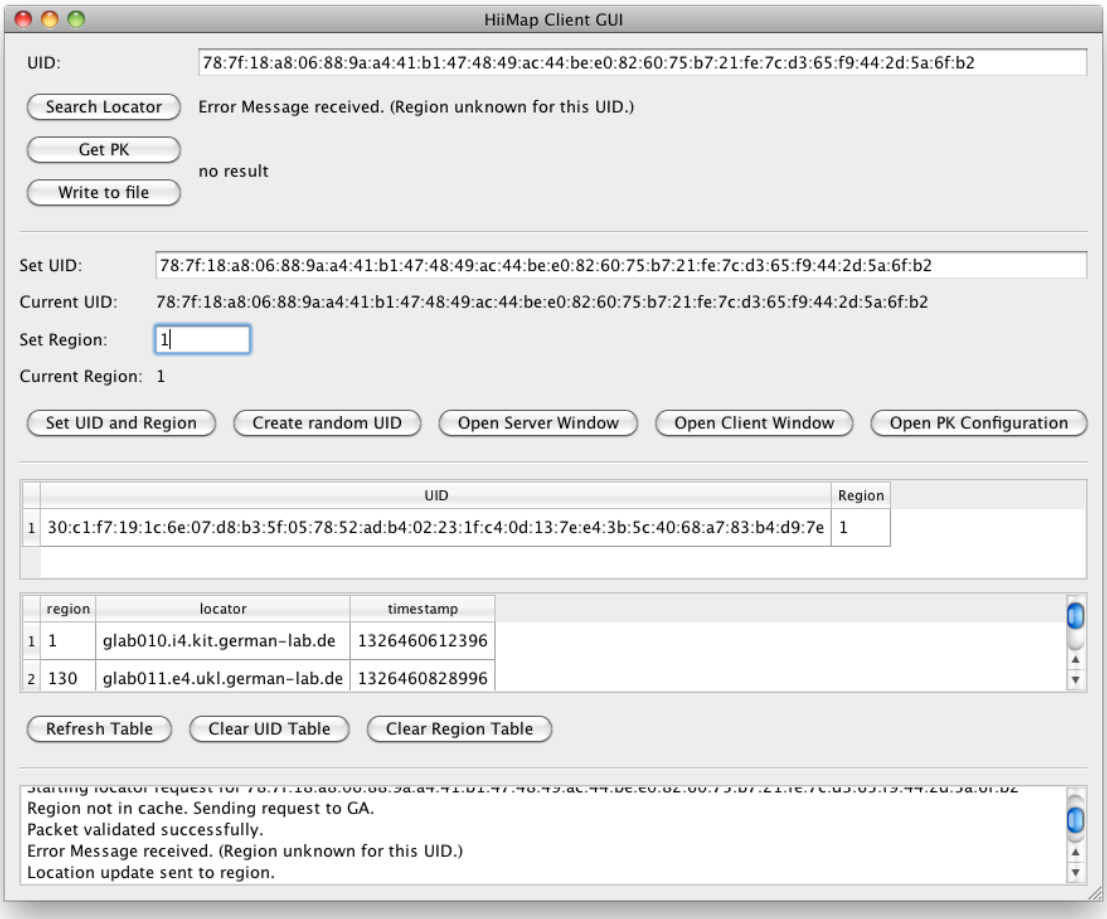


Figure 6.7.: HiiMap client demo software with PKI extension.

Except for the GA, the components of the mapping system were not required to be modified. The GA was extended to generate and provide the transformation directive to determine the storage locations of public key shares (cf. Chap. 4.3.1). In a real world example, this directive can be used to implement load balancing between the regions. For the prototype, however, the possible storage location numbers were evenly distributed over all existing regions. The load balancer and DHT peer software was not required to be extended as key shares are stored in the option list of each mapping entries. The protocol messages (`USR_LOCATION_UPDATE`, `USR_LOCATION_RESPONSE`) already included the option list as well.

The GUI–based client was extended to provide functionality to create, upload and download shares of a public key as well as reconstructing one. Figure 6.7 shows the modified GUI. In the top section two buttons were added. These allow the user to retrieve the public key for the UID listed in the top text box and to store the key in the ssh–rsa key file format. Furthermore, the "Open PK Configuration" button was added. This button opens the public key configuration dialog shown in Figure 6.8.

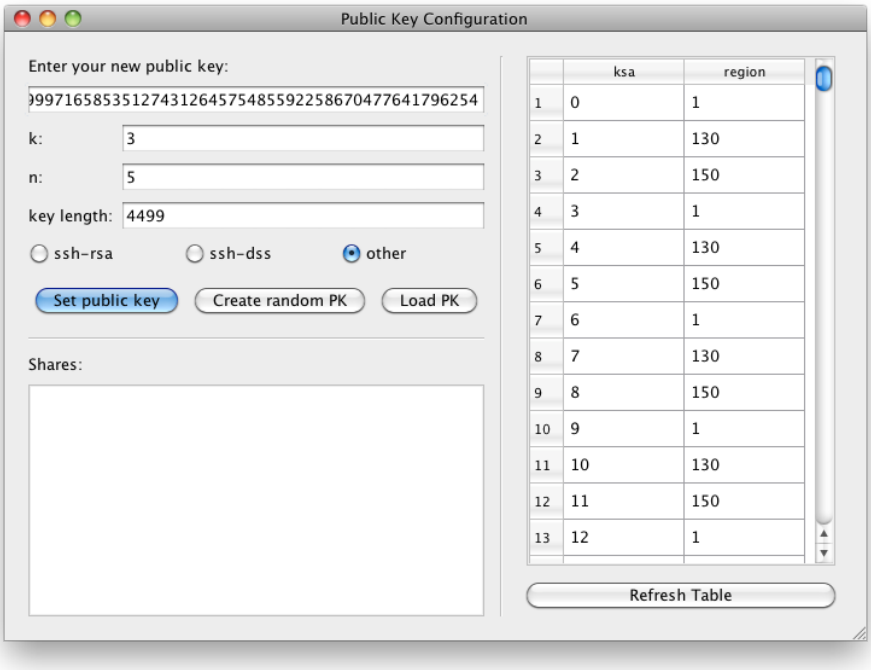


Figure 6.8.: Shares generation sub–window of the HiiMap client demo software.

This dialog allows the user to either randomly generate a public key or to select one from the file system. Furthermore, the user is able to select the number of shares to generate (n) and the threshold of shares required to reconstruct the key (k). On the right side of the window, the downloaded transformation directive is shown.

For the prototype, share generation and key reconstruction followed Shamirs secret sharing scheme [Sha79] with a variable key length. To create the shares, a polynomial with random supporting points is defined. Shares are then taken from this polynomial. To reconstruct the public key, the polynomial must be calculated from the shares. This is done using Neville interpolation in modular arithmetic. This means that the set of BigInteger values ($\mathbb{I}$)—which are required to handle numbers bigger than 2^{32}—modular a prime number (p) form a field. This has two advantages. The modulo operation limits the range of the field in x and y direction. The limitation in y direction, thereby, means that the size of the BigInteger values is limited and, therefore, the computational effort is restrained ($\mathbb{I} = \{x \in \mathbb{N}_0 \,|\, x < p\}$). By limiting the range of the field in x direction, the likelihood of each value is more uniformly distributed.

6.4. Privacy Service

The prototype of the privacy service required new components as well as the modification of the DHT peer and client software. The mapping system was extended to support proxy selection. To do so, a new module was added to the DHT peer software. This module is registered for any location request message which passes the main message queues (cf. Sec. 6.2.3). To follow the observer pattern in the initial software designe proved to be very valuable as the new module could be integrated without the need to modify much of the existing code.
The privacy service's central instance was implemented using a modified version of the DHT peer's source code. It was stripped off the EDRA algorithm and resilience module. It is able to accept `USR_LOCATION_UPDATE` messages from mobile nodes, maintains an own mapping database and can send `USR_LOCATION_UPDATE`s on behalf of a mobile node, containing all of the service's care–of locators. It further responds to location requests from its own proxies.

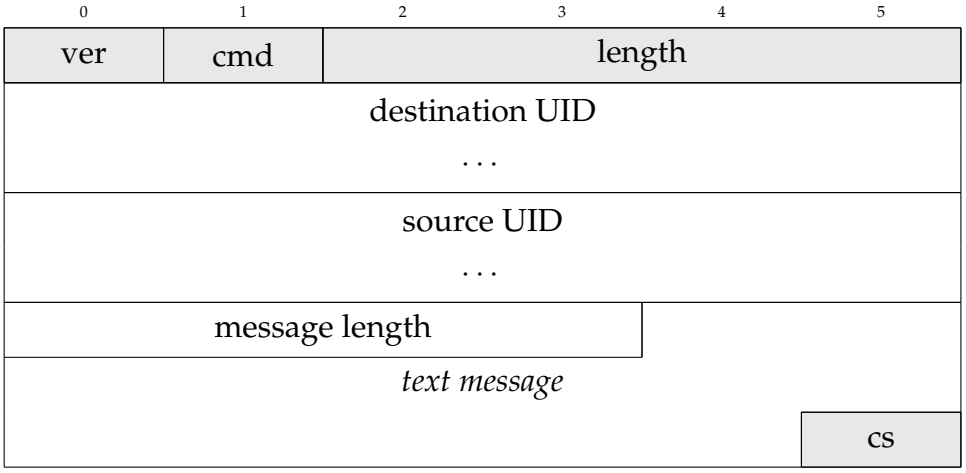


Figure 6.9.: Structure of the HiiMap privacy service test message.

The proxy software is a completely new component. It parses inbound packets and tries to resolve the enlisted destination UID in its internal cache. In case no entry is found, a location request is sent to the privacy service's central instance. After retrieving the destination locator (either from the cache or privacy service), the original packet is encapsulated with a new header and sent to the mobile node. The test message to be forwarded between a mobile and a correspondent node, thereby, consists in addition to the HiiMap header of the destination and source UID and a text message. Figure 6.9 shows the format of the test message (not encapsulated).
The GUI–based HiiMap client was extended to select whether to use a privacy service and to specify the UID of the service's central instance. Figure 6.10 shows a screenshot of the modified GUI. At the bottom of the middle section the UID of the privacy service can be entered. By pressing the button "Update to CS" (CS = central service/instance), a location update is sent to the privacy service.

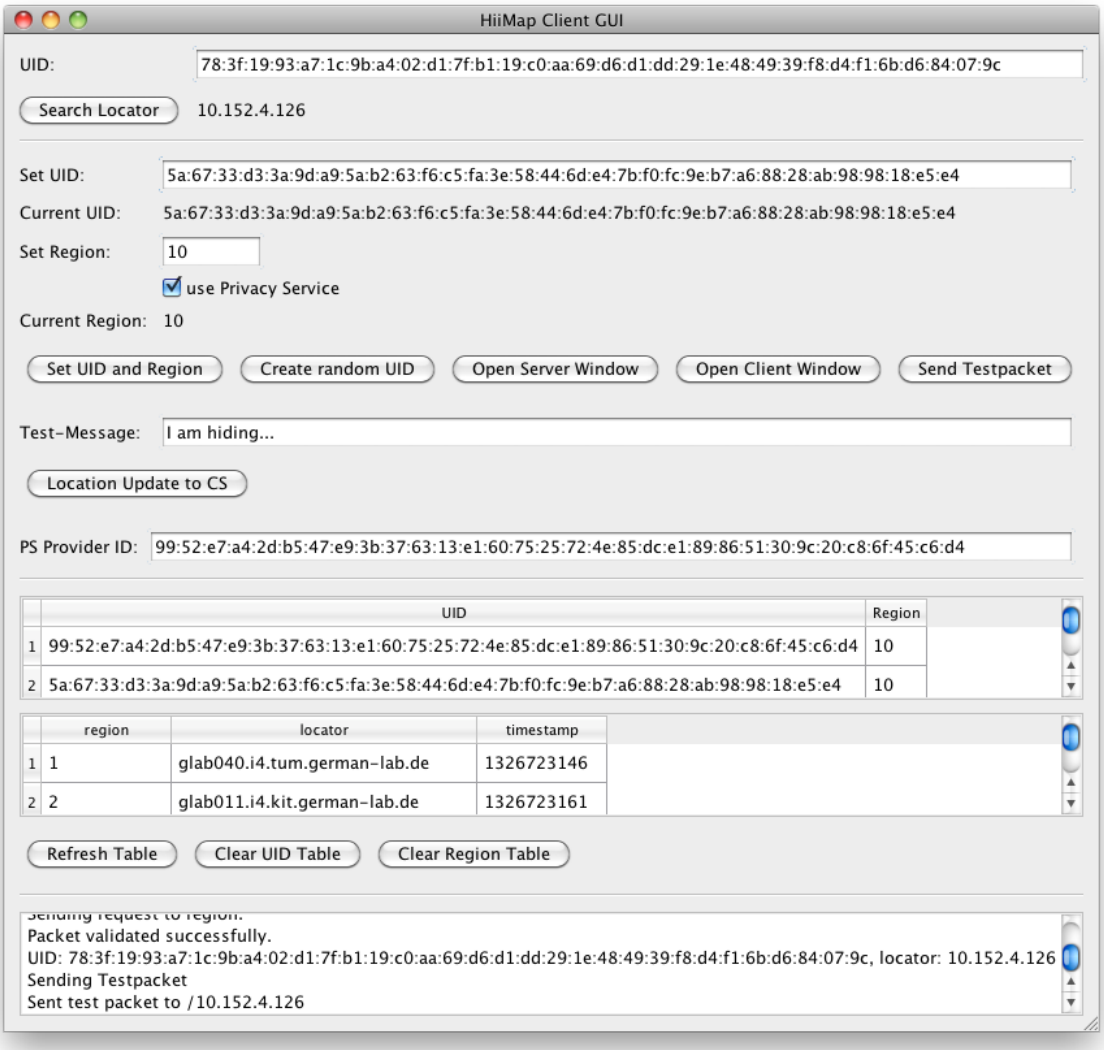


Figure 6.10.: HiiMap client demo software with location privacy extension.

By pressing the "Send Testpacket" button, the test message from the text box below is sent to the UID entered in the top most text box. The locator shown below this UID (beginning with 78:3f) is a care–of locator from a proxy. The test message, therefore, is sent to the proxy instead of directly to the mobile node. After successfully receiving the test message, the client software at the mobile node shows an alert dialog, containing the text message (not shown in the screenshot).

7. Conclusion & Outlook

As the Internet revolutionized the way we communicate, our expectations towards it changed accordingly. In the thirty years of its existence it evolved from a network to interconnect a few nodes to a global infrastructure transferring information for billions of users. The result of this unprecedented growth, however, is that the initial design principles do not match today's usage patterns anymore. When the core architecture of the Internet was designed in the late 1970s and early 1980s, Internet–capable mobile phones and almost real–time financial transactions, for example, were pure science fiction. This is why the design goals did not include mobility, security or the ability to scale to an enormous amount of subscribers.

As a result, the current Internet architecture faces various challenges which need to be solved within the next years in order to keep up the growth rate. Novel architecture proposals, therefore, are discussed in the research community. National and international research projects (e.g., G-Lab, 4WARD, Geni) are funded to research novel concepts. The spectrum, thereby, ranges from novel addressing schemes up to ecological considerations.

In this thesis, the aspects of scalability, mobility, security, trust and privacy were further examined. Based on these properties a novel Internet architecture was introduced and it was discussed how different mechanisms help to overcome the challenging aspects of today's architecture. The architecture is based on the locator/identifier separation paradigm, and its core is formed by a distributed mapping system. In addition to the mapping system, a security framework as well as a privacy service is integrated.

Mapping System The challenge of a locator/identifier split based architecture is how to map identifiers to locators. In this thesis, current mapping mechanisms as well as Next Generation Internet (NGI) ones were discussed and their shortcomings in regard to a NGI scenario outlined. As a consequence, a novel mapping system was introduced in this thesis which eliminates these shortcomings. The proposed system is called Hierarchical Internet Mapping Architecture (HiiMap) and designed to meet the expectations towards a NGI architecture. The mapping system is split into regions in order to provide the basis of a trustful mapping service. The reason behind this approach is to limit the influence of a single authority over the mapping system. Each region is controlled by an own authority to distribute the government–ship and to establish a common law basis among the participants of a region. In order to cope with the enormous load of the mapping system, an architecture based on Distributed Hash Tables (DHTs) was introduced. This architecture is able to scale with the query rate and amount of entries to store. To minimize lookup delays, caching hierarchies and a one hop DHT protocol are used.

Security Framework The presented security framework has two objectives. It secures the operation of the HiiMap mapping system and, further, provides a trust anchor for higher layer security protocols and applications. It, thereby, is divided into a Public Key Infrastructure (PKI) and client key management part. The public key distribution is integrated into the mapping system and based on threshold cryptography. This has the benefit that users are not required to trust a single instance provided by a third party. Public keys can even be retrieved in case parts of the mapping system collaborate with an attacker. The client key management is based on cryptographic smart cards in oder to provide a secure and user–friendly solution. The concept, thereby, is flexible enough to allow different types of smart cards purpose–build for various Internet capable devices. To this end, the objective of the framework is to overcome the limitations of today's key management solutions as well as other proposed NGI security concepts. In order to verify the integrity of the framework it was assessed against the AAA protocol evaluation guidelines.

Privacy Service The locator/identifier separation paradigm allows nodes to be tracked by providing real–time location information to attackers. The HiiMap architecture, therefore, integrates a privacy service which was presented in this thesis. The service is based on proxies which conceal the topological location of an end system. The advantage of the introduced privacy service over other proposals is its integration with the mapping system. This enables the service to limit the side effects of triangular routing caused by the necessity to reroute packets to the proxies. This is done by individually selecting a proxy for each communication flow. A proxy, thereby, is selected which is as close to the correspondent node and the direct communication path. This ensures that no location information is leaked, and triangular routing is kept to a minimum. The subsequent economical evaluation of the concept showed the importance of this selection mechanisms from a financial point of view as well.

The elements of the HiiMap architecture (i.e., mapping system, security framework and privacy service) were evaluated in various ways. All of the components, however, have been implemented in order to prove the feasibility of the proposed mechanisms. To this end, the prototypes and evaluations of the components showed that the HiiMap NGI architecture is a practical solution to the challenging aspects of today's Internet architecture.
Nevertheless, it is not realistic to assume that the current Internet architecture is replaced by a clean–slate one in a flag–day like manner. Future research, therefore, should concentrate on how elements from the introduced architecture can be back ported to today's Internet. The security framework (cf. Chap. 4), for example, is not dependent on a locator/identifier split based addressing scheme and, therefore, could easily be deployed in the current Internet architecture. Another approach besides back–porting elements could be to operate both architectures—the current one and HiiMap—in parallel and slowly migrate from one to the other. A sample migration strategy is roughly sketched out in Appendix B. Further research topics are to integrate other NGI aspects into the architecture, like application–tailored protocols, virtualization or cost–effective network management.

A. HiiMap Mapping System Protocol

This appendix lists the HiiMap protocol specification. The protocol messages are used by the prototypes presented in Chapter 6. The HiiMap protocol is a classic binary protocol. The current protocol version is `0x01` (hex). All bytes are sent in network byte order.

A.1. Data Types

These are the data types used for this protocol.

Type	Description	Num. of bytes
`byte`	signed byte	1
`ubyte`	unsigned byte	1
`int32`	signed integer	4
`uint32`	unsigned integer	4
`int64`	signed long integer	8
`uid`	128 bit unique identifier	16
`string`	int32 containing its length plus 0 or more characters as bytes	4+
`string list`	int32 containing its number of strings plus 0 or more strings	4+
`IDSpace`	represents a certain part of the ID space. Consists of two UIDs (start and end) and one additional byte	33
`data`	unspecified data field	-

A.2. Assigned Numbers

Error codes used in the error message.

Symbolic name	Error code (hex)	Description
`OK`	`0x00`	ok
`NX_IDENT`	`0x01`	UID not known to the mapping system
`NX_OPT`	`0x02`	Requested options not available for the UID
`NX_REG`	`0x03`	Contacting the wrong region

Command codes used in the protocol header.

Symbolic name	Command code (hex)
ERROR_MSG	0x00
REG_EDRA_PING	0x01
REG_EDRA_UPDATE	0x02
REG_EDRA_JOIN	0x03
REG_EDRA_LEAVE	0x04
GA_INSERT_REQUEST	0x21
GA_INSERT_RESPONSE	0x22
GA_DELETE_REQUEST	0x23
GA_DELETE_RESPONSE	0x24
USR_LOCATION_REQUEST	0x31
USR_LOCATION_RESPONSE	0x32
USR_LOCATION_UPDATE	0x33
USR_LOCATION_DELETE	0x34
USR_REGION_REQUEST	0x35
USR_REGION_RESPONSE	0x36
USR_REGTABLE_REQUEST	0x37
USR_REGTABLE_RESPONSE	0x38
USR_KSATABLE_REQUEST	0x39
USR_KSATABLE_RESPONSE	0x3a
LB_SYNCHRONIZE	0x51
LB_ALIVE	0x52
PRIVACY_LOC_UPDATE	0x61
PRIVACY_LOC_RESQUEST	0x62
PRIVACY_LOC_RESPONSE	0x63
PRIVACY_TUNNEL	0x64
RESILIANCE_MESSAGE	0x45
DATA_PACKET	0x11

A.3. Protocol Specification

The protocol description is split into five parts:

- Protocol header
- User protocol

- Mapping system internal protocol
- Backup system protocol
- Privacy service protocol

A.3.1. Protocol Header

The HiiMap header is used by any packet (except the Topology Monitor Protocol). It consists of the following fields:

```
ubyte  protocol version
ubyte  command code
int32  payload length
data   payload
ubyte  checksum
```

The `command code` must be one of the numbers from Section A.2. To calculate the checksum, all bytes of the packet excluding the checksum byte itself are added. The checksum is the value of the sum discarding any overflow information (remainder operation).

A.3.2. User Protocol Messages

ERROR_MSG

This message is used to communicate any error or status values to the client.

```
uid     unique identifier
ubyte   error code
string  detailed error description (optional)
string  locator used for forwarding (optional)
```

The `uid` indicates the request this message is a response for. In case the optional error description is not set, the string length field (4 bytes) will be zero, and no character bytes are appended.

USR_LOCATION_REQUEST

Mapping request sent by a client.

```
uid     unique identifier
string  option field
string  locator used for forwarding (optional)
```

USR_LOCATION_RESPONSE

Response from the mapping system to a mapping request.

```
uid          unique identifier
string       locator
string       locator used for forwarding (optional)
string list  options
```

USR_LOCATION_UPDATE

Message used to update an entry in the mapping system. *Only possible to overwrite a complete entry so far!*

```
uid          unique identifier
string       locator
string list  option list
```

USR_LOCATION_DELETE

Delete an entry from the mapping system.

```
uid  unique identifier
```

USR_REGION_REQUEST

This message is used to request the region of an identifier.

```
uid  unique identifier
```

USR_REGION_RESPONSE

The response to the request for an unknown region. It includes the identifier and the corresponding region.

```
uid    unique identifier
ubyte  region
```

USR_REGTABLE_REQUEST

This request is used to update the table storing the region to locator mappings.

```
in64  timestamp of the latest entry
```

USR_REGTABLE_RESPONSE

The message contains the whole or one part of the region to locator table. The table will be split into several packets if it does not fit into one message.

```
int32  number of packets
int32  number of entries in this packet
list   region to locator entries
```

One region to locator entry is defined as follows:

```
int32   region
string  locator
```

USR_KSATABLE_REQUEST

This message is used to request the transformation directive from the GA.

```
int64  timestamp of the current table
```

USR_KSATABLE_RESPONSE

This message contains the whole transformation directive. The regions are sorted in ascending order starting with zero.

```
int64       timestamp of the table
list int32  regions (optional)
```

OPTION STRINGS

OPTION FIELD The option field contains a comma–separated list of keywords for all the requested options. For example: `KEY,DATE,VALID`

OPTION LIST The OPTION LIST contains all requested options (each in a single string). The string starts with the option keyword followed by an equal sign and the option value itself. The list order may not match the request order. In case a requested option is not stored in the mapping system, a plain value entry is returned.

```
KEY=7843258784303389502
DATE=
VALID=true
```

NOTE: In the example above, the value `DATE` was not found in the mapping system.

A.3.3. Mapping System Internal Protocol Messages

The following messages are for region internal usage. They are used for the EDRA algorithm to balance the DHT and for status messages between peers, load balancers and the Global Authority (GA).

REG_EDRA_PING

Keep alive message of the EDRA algorithm.

```
int32  ttl
ubyte  type
```

REG_EDRA_UPDATE

This message is sent between DHT peers to signal an event.

```
int32   ttl
ubyte   type
uid     responsibility UID
uid     peer UID
string  locator list
int32   port
```

REG_EDRA_JOIN

This message is sent by a peer joining the DHT.

```
uid     responsibility UID
uid     peer UID
string  locator list
int32   port
```

REG_EDRA_LEAVE

A peer must send this message before gracefully leaving the DHT.

```
int32  ttl
ubyte  type
uid    responsibility UID
uid    peer UID
```

LB_SYNCHRONIZE

Message sent by a peer to register with the load balancer.

```
byte    region
int32   port
uid     unique identifier
string  locator
```

LB_ALIVE

Initializes the heartbeat connection and keeps it alive.

```
byte   message code
int32  number of entries
```

Message codes:

Symbolic name	Code (hex)	Description
INIT	0x00	initialize connection
INITACK	0x01	acknowledge connection
REQUEST	0x02	heartbeat request
RESPONSE	0x03	heartbeat response

GA_INSERT_REQUEST

Request to enlist the responsible region for the UID.

```
uid   unique identifier
byte  region
```

GA_INSERT_RESPONSE

Response to an insert request.

```
uid  unique identifier
```

GA_DELETE_REQUEST

Request to delete the responsible region entry for this UID.

```
uid  unique identifier
```

GA_DELETE_RESPONSE

Response to a delete request.

```
uid  unique identifier
```

A.3.4. Backup System Protocol Messages

These messages are used by the DHT peer's resilience module. All messages of the resilience module are encapsulated in message of type `RESILIENCE_MESSAGE`. The command code in the HiiMap header, therefore, is set to `0x45`. The Backup system protocol messages further use their own command codes.
Backup system protocol command codes for messages between the main and backup node.

Symbolic name	Command code (hex)
SETUP	0x01
ACK	0x02
RESIZE	0x03
FULL_DATA_UPDATE_T	0x04
DELETE_ENTRY	0x05
TERMINATE_T	0x06
HELLO_T	0x07
HASH_REQUEST	0x08
DATA_REQUEST_T	0x0A

Backup system protocol command codes for messages between the backup and main node.

Symbolic name	Command code (hex)
FULL_DATA_UPDATE_O	0x0C
OK	0x0D
TERMINATE_O	0x10
HELLO_O	0x11
INITIAL_HASH	0x12
DATA_REQUEST_O	0x14

SETUP

This message is used to initiate a backup connection and is usually answered by a OK message. It is sent from the main node to the backup node to–be.

```
uid       message origin
int32     handshake session number
IDSpace   affected UID subspace
byte      max.  possible sessions
byte      proposed num.  of sessions
byteArray proposed session numbers
```

OK

This message is a positive answer to a SETUP message (sent from the backup to the main node).

```
uid       message origin
int32     handshake session number
```

ACK

This is the final message of a connection establishment process.

```
uid       message origin
int32     handshake session number
byte      session number
```

INITIAL_HASH

This message is used to determine whether the data on both sides of the backup connection is synchronized. It contains all information about the data and parameters on the backup node.

```
IDSpace   affected UID subspace
int32     length of hash value
byteArray hash value of all UID entries in UID subspace
int32     num.  of entries stored within the affected UID
          subspace
```

HASH_RESPONSE

This message equals the `INITIAL_HASH` message but usually only covers a part of the whole ID space handled by a backup connection.

RESIZE

This message is used to tell the backup node that the UID subspace handled by the connection has changed.

```
IDSpace   affected UID subspace
```

FULL_DATA_UPDATE_O / _T

This messages are used to send a full data entry to the other side of the connection.

```
uid       unique identifier
string    locator
string    optionlist
list
```

DELETE_ENTRY

This message tells the backup node to delete a certain entry. NOTE: This message is not used in the context of modified UID subspaces or the establishment or termination of connections. It is only used in case the original entry is completely deleted from the mapping data base.

```
uid       unique identifier
```

DATA_REQUEST_O / _T

This request is sent to the other side of the backup connection to transmit all data falling into the specified UID subspace.

```
IDSpace   affected UID subspace
```

HELLO_O / _T

This message is used as a keep alive signal for the backup connection (between backup and main node).

```
uid       message origin
byte      session number
```

TERMINATE_O / _T

This message is used to terminate a backup connection.

```
uid       message origin
byte      session number
```

A.3.5. Privacy Service Protocol Messages

PRIVACY_LOC_UPDATE

This message is used by a client to update its locator with the privacy service.

```
uid     unique identifier
string  locator
```

PRIVACY_LOC_REQUEST

This command is used by a proxy to request the real locator of a Unique Identifier (UID) from the privacy service.

```
uid  unique identifier
```

PRIVACY_LOC_RESPONSE

Response from the privacy service to a location request.

```
uid     unique identifier
string  locator
```

A.4. Topo Monitor Protocol (TMP)

These protocol messages are used between a region's load balancer and the topology monitor iOS application.

A.4.1. TMP Header

The Topo Monitor Protocol (TMP) does not use the HiiMap header as specified in Section A.3.1! An own header as follows is used:

```
ubyte  TMP protocol version
ubyte  TMP command code
int32  payload length
data   payload
```

These are the command codes for the TMP protocol.

Symbolic name	Command code (hex)
ERROR_MSG	0x00
TMP_PEER_LIST	0x70
TMP_PEER_STAT	0x71

A.4.2. TMP Messages

ERROR_MSG

This message is used to communicate any error or status values to the client.

```
ubyte   error code
string  detailed error description
```

TMP_PEER_LIST

This message contains a list with all peers in the region and their current number of entries. This list may be sent at any time. It is assumed that all nodes in this list are online. In case one peer is in the list but currently offline, a TMP_PEER_STAT message with status offline must be sent immediately for the node.

```
int32    number of peers
{ int32  number of entries at the peer }
```

NOTE: The last entry is repeated as often as specified by 'number of peers'.

TMP_PEER_STAT

The status for a specific peer. The peer number in this message must correspond to the list number of the TMP_PEER_LIST message sent last.

```
int32  peer number
ubyte  peer status
```

Peer status	Code (hex)
TMP_STAT_UPD	0x00
TMP_STAT_REQ	0x01
TMP_STAT_ONL	0x10
TMP_STAT_OFF	0x11

A.5. Protocol Implementation Guide

The following demo code shows a sample Java implementation of a client. It is shown how to programmatically request and receive a mapping entry. Listing A.1 shows the part of the sample client on how to request a mapping for a specific identifier (UID). First, the request (*USR_LOCATION_REQUEST*) is constructed and then encapsulated into a HiiMap header. Afterwards, the client waits for a reply from the mapping system (with timeout). In a third step, the received response packet is parsed by the HiiMap packet header class. Depending on the command code, the payload is either interpreted (and parsed) as a *ERROR_MSG* or a *USR_LOCATION_RESPONSE*.

```
1  // The identifier we want to request the mapping for
2  Identifier id1 = new Identifier("80:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00");
3
4  // Constructing the request (no options)
5  USR_Loc_Req request = new USR_Loc_Req(id1,"");
6  // Use the request as the payload for the HiiMap packet
7  HiiMapHeader header = new HiiMapHeader(Config.getProtocolVersion(),HiiMapPacketType.USR_LOCATION_REQUEST,
       request);
8  // Send the packet
9  DatagramPacket pak = new DatagramPacket(header.getRawPacket(),header.getSize(),InetAddress.getByName(glab010.i4.tum.
       german-lab.de),Config.getMappingPort());
10 socket.send(pak);
11
12 System.out.println("---REQUEST SENT---");
13
14 // Wait for the response
15 DatagramPacket resp = new DatagramPacket(new byte[1024],1024);
16 while(true) {
17         socket.setSoTimeout(5 * 1000);
18         socket.receive(resp);
19         if (resp.getData()[0] != Config.getProtocolVersion()) {
20                 System.out.println("[ConectionHeandler] " + resp.getAddress().getHostAddress() + " is using the
                       wrong protocol version: " + resp.getData()[0]);
21         } else {
22                 break;
23         }
24 }
```

```
25
26  // Parse the packet
27  HiiMapHeader respHeader = new HiiMapHeader(resp.getData(), resp.getLength());
28
29  // Interpret the packet
30  switch (respHeader.getCommand()) {
31  case HiiMapPacketType.ERROR_MSG: {
32          // Seems like the requested identifier isn't stored in the mapping system
33          ERROR_msg error = (ERROR_msg) respHeader.getPayLoad();
34          System.out.println("Error code received: " + error.getCode());
35          break;
36  }
37  case HiiMapPacketType.USR_LOCATION_RESPONSE: {
38          // Read the returned values
39          USR_Loc_Resp data = (USR_Loc_Resp) respHeader.getPayLoad();
40
41          OptionList optList = data.getOptList();
42          System.out.println("ID: \t " + data.getUid().getStringID());
43          System.out.println("Locator: " + data.getLocator());
44          for (String opt : optList.getOptList()) {
45                  System.out.println("OPT: \t " + opt);
46          }
47          break;
48  }
49  }
```

Listing A.1: "Requesting a mapping entry"

Below is the *USR_LOCATION_REQUEST* packet as seen by wireshark:

```
0000  01 31 00 00 00 14 80 00 00 00 00 00 00 00 00 00  .1...$..........
0010  00 00 00 00 00 00 00 00 00 00 c6                 ...........
```

(red = protocol version, green = command code, blue = payload length, gray = uid, orange = number of options, purple = checksum)

Below is the *USR_LOCATION_RESPONSE* packet as seen by wireshark:

```
0000  01 32 00 00 00 62 80 00 00 00 00 00 00 00 00 00  .2...r..........
0010  00 00 00 00 00 00 00 00 00 10 36 39 2e 31 37 2e  ..........69.17.
0020  31 31 36 2e 31 32 34 26 38 30 00 00 00 02 00 00  116.124&80......
0030  00 16 6e 6f 74 65 3d 49 5f 61 6d 5f 61 5f 74 65  ..note=I_am_a_te
0040  73 74 5f 65 6e 74 72 79 00 00 00 1c 72 77 3d 66  st_entry....rw=f
0050  65 65 6c 5f 66 72 65 65 5f 74 6f 5f 6f 76 65 72  eel_free_to_over
0060  77 72 69 74 65 5f 6d 65 cc                       write_me.
```

(red = protocol version, green = command code, blue = payload length, purple = checksum, orange/darkblue = payload)

B. IPv6 to HiiMap Migration

In this appendix, a possible migration strategy from a plain IPv6 to the Hierarchical Internet Mapping Architecture (HiiMap) architecture is sketched out. The address space for the locator in the HiiMap architecture remains to be IPv6. This description does neither claim to be complete nor to include all eventualities. For this migration strategy it is assumed that all Internet subscribers as well as intermediate network nodes (e.g., routers) support the IPv6 protocol and do not rely on IPv4 anymore.
The strategy is designed around a slow migration from one to the other architecture, wherein both architectures coexist until the migration is completed. In the beginning all end systems are only equipped with an IPv6 protocol stack and are not capable of the HiiMap protocol. The first step, which starts the migration process, is to set up the HiiMap mapping system (cf. Chap. 3) and its security framework (cf. Chap. 4). This provides the infrastructure for end systems ready to switch to the new architecture.
New end systems (or older ones which are upgraded) are required to handle both protocol stacks—the IPv6 and HiiMap one. Instead of operating two independent protocol stacks, however, a shim layer is introduced which is transparent for IPv6 communication flows and adds additional functionality for HiiMap ones. Applications of dual–stack end systems solely use Unique Identifiers (UIDs) to address peer applications—this is even in case the peer is only equipped with an IPv6 protocol stack. Figure B.1 illustrates how the shim layer behaves in case both end systems communicate via the HiiMap protocol.

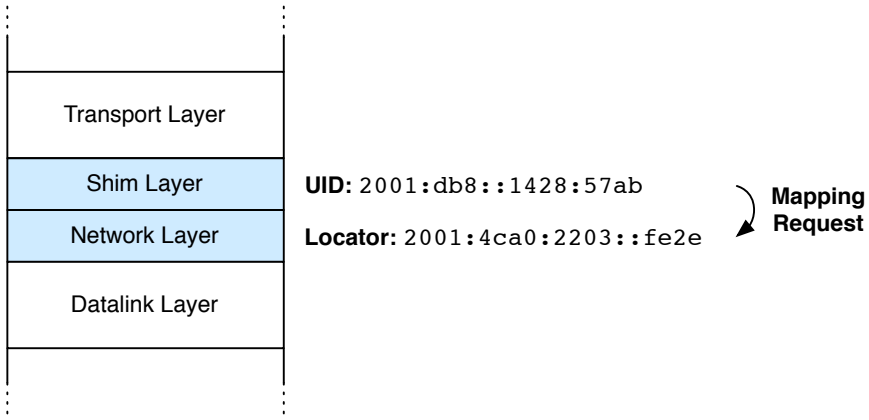


Figure B.1.: Migration strategy: Shim layer in HiiMap operation.

The shim layer in this case takes a UID as the destination address and translates it into a locator by issuing a request to the mapping system. In case an entry for the UID is

found in the mapping system, the peer's current IPv6 locator is returned and used by the network layer.
In case, however, the peer is not equipped with the HiiMap protocol stack, no entry will be found in the mapping system. The shim layer then passes the UID to the network layer which will interpret it as a regular IPv6 address as both addresses are 128 bit long (cf. Fig. B.2). The shim layer further does not encapsulate the application data into the HiiMap header (which includes source and destination UIDs) before passing it on to the network layer. Response packets are routed based on the end system's IPv6 locator.

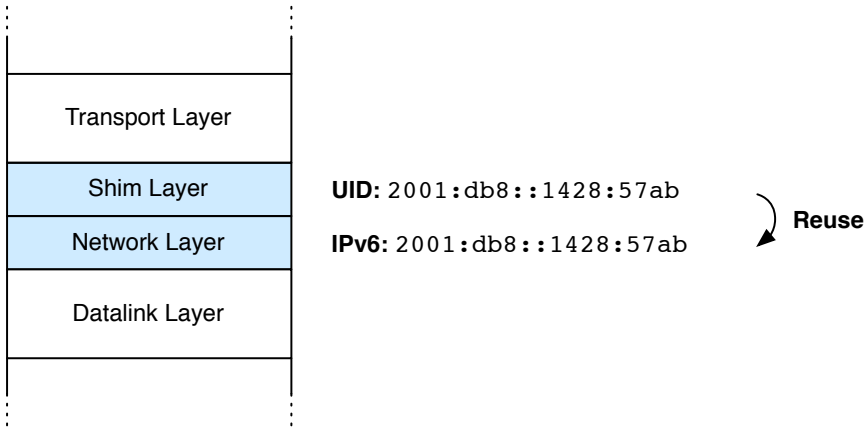


Figure B.2.: Migration strategy: Shim layer in IPv6 operation.

In that way, end systems operating a dual stack are able to initiate communication flows with peers of both architectures. These end systems, however, are not reachable by IPv6–only peers in case they roam. IPv6–only end systems are not aware of UIDs and are solely able to address packets to IPv6 addresses. During the migration phase, therefore, end systems which operate a dual protocol stack and want to offer services to other end systems (e.g., servers) need to use their IPv6 locator as their UID. In that way, a mapping request for the UID resolves to the end system's IPv6 locator while IPv6–only peers use the end system's UID as a regular IPv6 destination address. In fact, IPv6–only peers are not aware that the address they are using is an UID as it does not distinguish from a regular IPv6 address. Hence, mechanisms like Domain Name System (DNS) do not have to be altered to indicate whether the returned address is a UID or IPv6 address.
The migration is completed after all nodes are upgraded to use the HiiMap architecture. End systems offering a service are then able to change their locator as every peer is able to issue mapping requests in order to learn the end system's current locator.

Abbreviations

AS	Autonomous System
CAPEX	Capital Expenses
DFZ	Default Free Zone
DHT	Distributed Hash Table
DNS	Domain Name System
FIB	Forwarding Information Base
GA	Global Authority
HiiMap	Hierarchical Internet Mapping Architecture
IP	Internet Protocol
ISP	Internet Service Provider
LRZ	Leibniz Supercomputing Center
MAC	Media Access Control
NGI	Next Generation Internet
PKI	Public Key Infrastructure
RIB	Routing Information Base
RP	Regional Prefix
RTT	Round Trip Time
OPEX	Operational Expenses
UID	Unique Identifier

Bibliography

Publications by the author

[FH10] Wolfgang Fritz and Oliver Hanka, *Smart card based security in locator/identifier–split architectures*, Networks (ICN), 2010 Ninth International Conference on (Les Menuires, France), April 2010, pp. 194 –200.

[Han10a] Oliver Hanka, *Location privacy in Next Generation Internet architectures*, Internet Architecture Board Workshop on Internet Privacy (Boston, USA), December 2010.

[Han10b] Oliver Hanka, *A privacy service for locator/identifier–split architectures based on mobile IP mechanisms*, Advances in Future Internet (AFIN), 2010 Second International Conference on (Venice/Mestre, Italy), July 2010, pp. 6 –10.

[Han11] Oliver Hanka, *The cost of location privacy in locator/identifier–split architectures*, 30th IEEE International Performance Computing and Communications Conference - HotWiSec (Orlando, FL, USA), November 2011.

[Han12] Oliver Hanka, *How to prevent identity fraud in locator/identifier–split architectures*, International Conference on Computing, Networking and Communications (ICNC 2012) (Maui, HI, USA), January 2012.

[HEP+11] Oliver Hanka, Michael Eichhorn, Martin Pfannenstein, Jörg Eberspächer, and Eckehard Steinbach, *A distributed public key infrastructure based on threshold cryptography for the HiiMap Next Generation Internet architecture*, Future Internet **3** (2011), no. 1, 14–30.

[HF11] Oliver Hanka and Wolfgang Fritz, *An holistic approach to public/private–key based security in locator/identifier–split architectures*, International Journal On Advances in Security **3** (2011), no. 3&4, 135–145.

[HKS+09] Oliver Hanka, Gerald Kunzmann, Christoph Spleiss, Jörg Eberspächer, and Armin Bauer, *HiiMap: Hierarchical Internet mapping architecture*, Future Information Networks, 2009. ICFIN 2009. First International Conference on (Beijing, China, P.R. China), October 2009, pp. 17–24.

[HL11] Oliver Hanka and Julian Lamberty, *Security analysis of the cryptographic namespace design in Next Generation Internet architectures*, IEEE ICC 2011 - 4th International Workshop on the Network of the Future (Kyoto, Japan), June 2011.

[HSKE09] Oliver Hanka, Christoph Spleiss, Gerald Kunzmann, and Jörg Eberspächer, *A novel DHT–based network architecture for the next generation internet*, Networks, 2009. ICN '09. Eighth International Conference on (Cancun, Mexico), March 2009, pp. 332–341.

[HW11] Oliver Hanka and Hans Wippel, *Secure deployment of application–tailored protocols in future networks*, 2nd IFIP International Conference Network of the Future (Paris, France), November 2011.

[LH10] Bernhard Lichtinger and Oliver Hanka, *Secure setup of inter–provider ethernet services based on a novel naming schema*, Network Operations and Management Symposium (NOMS), 2010 IEEE (Osaka, Japan), April 2010, pp. 886 –889.

[WHSM10] Hans Wippel, Oliver Hanka, Christoph Spleiss, and Denis Martin, *Evaluation of future network architectures and services in the G-Lab testbed*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 46, Springer, 2010, pp. 587–589.

General publications

[AAER06] B. Ahlgren, J. Arkko, L. Eggert, and J. Rajahalme, *A node identity internetworking architecture*, INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings (Washington, DC, USA), IEEE Computer Society, April 2006, pp. 1–6.

[ABF+08] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, *Accountable Internet protocol (AIP)*, Proceedings of the ACM SIGCOMM 2008 conference on Data communication (New York, NY, USA), SIGCOMM '08, ACM, August 2008, pp. 339–350.

[ABH08] R. Atkinson, S. Bhatti, and S. Hailes, *Mobility through naming: impact on DNS*, Proceedings of the 3rd international workshop on Mobility in the evolving Internet architecture (New York, NY, USA), MobiArch '08, ACM, August 2008, pp. 7–12.

[Aur03] T. Aura, *Cryptographically generated addresses (CGA)*, Information Security, Lecture Notes in Computer Science, vol. 2851, Springer Berlin / Heidelberg, 2003, pp. 29–43.

[BBJ+09] E. Barker, W. Burr, A. Jones, T. Polk, S. Rose, M. Smid, and Q. Dang, *Recommendation for key management - part 3*, Special publication 800-57, NIST, December 2009.

[BGT04] Tian Bu, Lixin Gao, and Don Towsley, *On characterizing bgp routing table growth*, Computer Networks **45** (2004), no. 1, 45 – 54.

[BS03] A.R. Beresford and F. Stajano, *Location privacy in pervasive computing*, Pervasive Computing, IEEE **2** (2003), no. 1, 46 – 55.

[BSD10] Walter Bamberger, Josef Schlittenlacher, and Klaus Diepold, *A trust model for intervehicular communication based on belief theory*, Social Computing (SocialCom), 2010 IEEE Second International Conference on, IEEE Computer Society, August 2010, pp. 73–80.

[Cha81] David L. Chaum, *Untraceable electronic mail, return addresses, and digital pseudonyms*, Commun. ACM **24** (1981), 84–90.

[Cla88] D. Clark, *The design philosophy of the DARPA Internet protocols*, SIGCOMM Comput. Commun. Rev. **18** (1988), 106–114.

[Coo11] A. Cooper, *Report from the Internet privacy workshop*, Work in progress (draft-iab-privacy-workshop-01), October 2011.

[Eck06] C. Eckert, *IT–Sicherheit: Konzepte - Verfahren - Protokolle*, Oldenbourg, 2006.

[ES00] Carl Ellsion and Bruce Schneider, *Ten risks of PKI: What you're not being told about public key infrastructure*, Computer Security Journal **16** (2000), no. 1, 1–7.

[FCM+09] A. Feldmann, L. Cittadini, W. Mühlbauer, R. Bush, and O. Maennel, *HAIR: Hierarchical architecture for Internet routing*, ReArch'09 (New York, NY, USA), ACM, December 2009.

[Fel07] A. Feldmann, *Internet clean-slate design: what and why?*, SIGCOMM Comput. Commun. Rev. **37** (2007), 59–64.

[FFML10] Dino Farinacci, Vince Fuller, Dave Meyer, and Darrel Lewis, *Locator/ID separation protocol (LISP)*, Work in progress (draft-ietf-lisp-09), October 2010.

[GAB09] Alex Galis, Henrik Abramowicz, and Marcus Brunner, *Management and service-aware networking architectures (MANA) for future Internet*, Tech. report, Future Internet Assembly, May 2009.

[GPW+04] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz, *Comparing elliptic curve cryptography and RSA on 8-bit CPUs*, Lecture Notes in Computer Science, vol. 3156/2004, Springer, Berlin/Heidelberg, Germany, 2004, pp. 925–943.

[Han06] M. Handley, *Why the Internet only just works*, BT Technology Journal **24** (2006), 119–129.

[ISO94] ISO, *Information technology - open systems interconnection - basic reference model: The basic model*, ISO/IEC standard 7498-1:1994n, International Organization for Standardization, November 1994.

[ITU05] ITU, *The international public telecommunication numbering plan*, ITU-T Recommendation E.164, Telecomunication Standarization Sector of ITU, February 2005.

[JCAC+10] L. Jakab, A. Cabellos-Aparicio, F. Coras, D. Saucez, and O. Bonaventure, *LISP–TREE: A DNS hierarchy to support the LISP mapping system*, Selected Areas in Communications, IEEE Journal on **28** (2010), no. 8, 1332 –1343.

[Jia06] Raj Jian, *Internet 3.0: Ten problems with current Internet architecture and solutions for the next generation*, Military Communications Conference, 2006. MILCOM 2006. IEEE, October 2006, pp. 1 –9.

[JSBM02] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris, *DNS performance and the effectiveness of caching*, IEEE/ACM Trans. Netw. **10** (2002), no. 5, 589–603.

[KBR06] Joseph S. Kong, Jesse S. A. Bridgewater, and Vwani P. Roychowdhury, *A general framework for scalability and performance analysis of DHT routing systems*, Dependable Systems and Networks, International Conference on **0** (2006), 343–354.

[KCC+07] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, *A data-oriented (and beyond) network architecture*, SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications (New York, NY, USA), ACM, August 2007, pp. 181–192.

[KKFT08] Jun Kurihara, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka, *A new (k,n)-threshold secret sharing scheme and its extension*, Proceedings of the 11th international conference on Information Security (Berlin, Heidelberg), ISC '08, Springer-Verlag, 2008, pp. 455–470.

[Kob87] Neal Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation **48** (1987), no. 177, 203–209.

[Kon11] Elisavet Konstantinou, *Efficient cluster-based group key agreement protocols for wireless ad hoc networks*, Journal of Network and Computer Applications **34** (2011), no. 1, 384 – 393.

[KZL+01] Jiejun Kong, Petros Zerfos, Haiyun Luo, Songwu Lu, and Lixia Zhang, *Providing robust and ubiquitous security support for mobile ad-hoc networks*, Network Protocols, 2001. Ninth International Conference on, November 2001, pp. 251 –260.

[LB04] J. Linn and M. Branchaud, *An examination of asserted PKI issues and proposed alternatives*, 3rd Annual PKI R&D Workshop, March 2004, pp. 34–47.

[LOP05] Javier Lopez, Rolf Oppliger, and Ganther Pernul, *Why have public key infrastructures failed so far?*, Internet Research: Electronic Networking Applications and Policy **15** (2005), no. 5, 544–556.

[MA06] L.R. Monnerat and C.L. Amorim, *D1HT: a distributed one hop hash table*, Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, April 2006, p. 10 pp.

[MM04] C. Siva Ram Murthy and B.S. Manoj, *Ad hoc wireless networks: Architectures and protocols*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[MO09] K. Maekawa and Y. Okabe, *An enhanced location privacy framework with mobility using host identity protocol*, Applications and the Internet, 2009. SAINT '09. Ninth Annual International Symposium on, July 2009, pp. 23 –29.

[MSS+06] Alfredo Matos, Justino Santos, Susana Sargento, Rui Aguiar, João Girão, and Marco Liebsch, *HIP location privacy framework*, Proceedings of first ACM/IEEE international workshop on Mobility in the evolving Internet architecture (New York, NY, USA), MobiArch '06, ACM, December 2006, pp. 57–62.

[PJPSi10] J. Pan, R. Jain, S. Paul, and Chakchai So-in, *MILSA: A new evolutionary architecture for scalability, mobility, and multihoming in the future Internet*, Selected Areas in Communications, IEEE Journal on **28** (2010), no. 8, 1344 –1362.

[QIdLB07] B. Quoitin, L. Iannone, C. de Launois, and O. Bonaventure, *Evaluating the benefits of the locator/identifier separation*, Proceedings of 2nd ACM/IEEE international workshop on Mobility in the evolving Internet architecture (New York, NY, USA), MobiArch '07, ACM, 2007, pp. 5:1–5:6.

[RD10] J. Rexford and C. Dovrolis, *Future Internet architecture: clean–slate versus evolutionary research*, Commun. ACM **53** (2010), 36–40.

[RE03] Wolfgang Rankl and Wolfgang Effing, *Smart Card Handbook*, 3rd edition ed., John Wiley & Sons, 2003.

[RFC 2459] R. Housley, W. Ford, W. Polk, and D. Solo, *RFC 2459: Internet x.509 public key infrastructure certificate and CRL profile*, January 1999.

[RFC 3280] R. Housley, W. Polk, W. Ford, and D. Solo, *RFC 3280: Internet x.509 public key infrastructure certificate and certificate revocation list (CRL) profile*, April 2002.

[RFC0675] V. G. Cerf, Y. K. Dalal, and C. A. Sunshine, *RFC 675: Specification of Internet Transmission Control Program*, December 1974.

[RFC0791] Information Sciences Institute, USC, *RFC 791: Internet protocol*, September 1981.

[RFC0801] J. Postel, *RFC 801: Ncp/tcp transition plan*, November 1981.

[RFC0826] David C. Plummer, *RFC 826: An ethernet address resolution protocol*, November 1982.

[RFC1034] P. Mockapetris, *RFC 1034: Domain names - concepts and facilities*, November 1987.

[RFC1035] P. Mockapetris, *RFC 1035: Domain names - implementation and specification*, November 1987.

[RFC1122] R. Braden, *RFC 1122: Requirements for Internet hosts – communication layers*, October 1989.

[RFC1883] S. Deering and R. Hinden, *RFC 1883: Internet Protocol, version 6 (IPv6) specification*, December 1995.

[RFC2460] S. Deering and R. Hinden, *RFC 2460: Internet Protocol, version 6 (IPv6) specification*, December 1998.

[RFC2989] B. Aboba, P. Calhoun, S. Glass, T. Hiller, P. McCann, H. Shiino, G. Zorn, G. Dommety, C. Perkins, B. Patil, D. Mitton, S. Manning, M. Beadles, P. Walsh, X. Chen, S. Sivalingham, A. Hameed, M. Munson, S. Jacobs, B. Lim, B. Hirschman, R. Hsu, Y. Xu, E. Campbell, S. Baba, and E. Jaques, *RFC 2989: Criteria for evaluating AAA protocols for network access*, November 2000.

[RFC3715] B. Aboda and W. Dixon, *RFC 3715: Ipsec–network address translation (NAT) compatibility requirements*, December 2005.

[RFC3775] D. Johnson, C. Perkins, and J. Arkko, *RFC 3775: Mobility support in IPv6*, June 2004.

[RFC4033] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *RFC 4033: DNS security introduction and requirements*, March 2005.

[RFC4301] S. Kent and K. Seo, *RFC 4301: Security architecture for the Internet protocol*, December 2005.

[RFC4423] R. Moskowitz and P. Nikander, *RFC 4423: Host identity protocol*, May 2006.

[RFC4984] D. Meyer, L. Zhang, and K. Fall, *RFC 4984: Report from the IAB workshop on routing and addressing*, September 2007.

[RFC5204] J. Laganier and L. Eggert, *RFC 5204: Host identity protocol (HIP) rendezvous extension*, April 2008.

[RFC5944] C. Perkins, *RFC 5944: IP mobility support in IPv4, revised*, November 2010.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Commun. ACM **21** (1978), 120–126.

[RSA83] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public–key cryptosystems*, Communications of the ACM **26** (1983), no. 1, 96–99.

[Sch03] J. Schiller, *Mobile communications*, 2. ed., Addison Wesley, London, Great Britain, 2003.

[SGP+07] M. Siekkinen, V. Goebel, T. Plagemann, K.-A. Skevik, M. Banfield, and I. Brusic, *Beyond the future Internet–requirements of autonomic networking architectures to address long term future networking challenges*, Future Trends of Distributed Computing Systems, IEEE International Workshop **0** (2007), 89–98.

[Sha79] A. Shamir, *How to share a secret*, Communications of the ACM **22** (1979), no. 11, 612–613.

[Sha85] A. Shamir, *Identity–based cryptosystems and signature schemes*, Advances in Cryptology (George Blakley and David Chaum, eds.), Lecture Notes in Computer Science, vol. 196, Springer Berlin / Heidelberg, 1985, pp. 47–53.

[Sho99] P.W. Shor, *Polynomial–time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM review **41** (1999), no. 2, 303–332.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, *Chord: A scalable peer-to-peer lookup service for Internet applications*, SIGCOMM Comput. Commun. Rev. **31** (2001), 149–160.

[SQBB10] C. Song, Z. Qu, N. Blumm, and A. Barabasi, *Limits of predictability in human mobility*, Science **327** (2010), no. 5968, 1018–1021.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark, *End–to–end arguments in system design*, ACM Trans. Comput. Syst. **2** (1984), 277–288.

[Woh11] Petra Wohlmacher, *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung*, Bundesanzeiger 85, Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen, May 2011.

[YN06] Jukka Ylitalo and Pekka Nikander, *BLIND: A complete identity protection framework for end–points*, Security Protocols (Bruce Christianson, Bruno Crispo, James Malcolm, and Michael Roe, eds.), Lecture Notes in Computer Science, vol. 3957, Springer Berlin Heidelberg, 2006, pp. 163–176.

Cited websites

[Ass11] Root Server Technical Operations Association, *DNS root server*, http://www.root-servers.org/, July 2011.

[BDR+96] M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Weiner, *Minimal key lengths for symmetric ciphers to provide adequate commercial security*, http://www.schneier.com/paper-keylength.pdf, January 1996.

[Cap12] Justin Cappos, *Seattle - open peer–to–peer computing*, https://seattle.cs.washington.edu/, January 2012.

[Dev11] Gieseke & Devrient, *Giesecke & Devrient secure flash solutions*, http://www.gd-sfs.com/, December 2011.

[EGI10] EGIHosting, *Colocation pricing*, http://www.egihosting.com/colocation.aspx, November 2010.

[Hus10] G. Huston, *IPv4 address report*, http://www.potaroo.net/tools/ipv4/, August 2010.

[Hus11] G. Huston, *The growth of the BGP table - 1994 to present*, http://bgp.potaroo.net/, March 2011.

[Lei11] Leibnitz Rechenzentrum, *DNS statistics*, http://dnsstat.lrz-muenchen.de/cgi-bin/dns.cgi, October 2011.

[Lei12] Leibnitz Rechenzentrum, *Leibniz supercomputing centre*, http://www.lrz.de, January 2012.

[NCC10] Ripe NCC, *RIS statistics report (2010-05-07 - 2010-05-14)*, http://www.ris.ripe.net/weekly-report/reports/risreport-20100507-20100514.txt, May 2010.

[Nor10] William Norton, *Internet transit prices - historical and projected*, http://drpeering.net/white-papers/Internet-Transit-Pricing-Historical-And-Projected.php, November 2010.

[Pay10] PayScale, Inc., *Salary survey for industry: Network and communications services*, http://www.payscale.com/research/US/Industry=Network_and_Communications_Services/Salary, May 2010.

[TER12] TERENA, *Eduroam*, http://www.eduroam.org/, January 2012.

[TG09] P. Tran-Gia, *G-Lab*, http://www.german-lab.de, October 2009.

[The10] The CA / Browser Forum, *Guidelines for the issuance and management of extended validation certificates*, http://www.cabforum.org/Guidelines_v1_3.pdf, November 2010.

[The11] The Cooperative Association for Internet Data Analysis, *AS relationships*, http://www.caida.org/data/active/as-relationships/index.xml, August 2011.

[TPU03] The Trustees of Princeton University, *PlanetLab*, http://www.planet-lab.org/, January 2003.

[Tra11] Transaction Processing Performance Council, *TPC–C - top ten performance results - non-clustered - version 5 results*, http://www.tpc.org/tpcc/results/tpcc_perf_results.asp?resulttype=noncluster, November 2011.

[Uni06] United Nations, *Press release org/1469*, http://www.un.org/News/Press/docs/2006/org1469.doc.htm, July 2006.