Department of Computer Science
Series of Publications A
Report A-2012-8

Secure Connectivity With Persistent Identities

Samu Varjonen

To be presented with the permission of the Faculty of the Science of the University of Helsinki, for public criticism in Hall 13, University of Helsinki Main Building, on 14 of November 2012 at noon.

University of Helsinki
Finland

Contact information

Postal address:
Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: postmaster@cs.helsinki.fi (Internet)

URL: http://www.cs.Helsinki.FI/

Telephone: +358 9 1911

Telefax: +358 9 191 51120

Supervisor(s): Jussi Kangasharju, Sasu Tarkoma, Andrei Gurtov

Pre-examiners: Jarmo Harju, Tampere University of Technology, Finland and Mika Ylianttila, University of Oulu, Finland

Opponent: Hannu H. Kari, National Defence University, Finland
Custos: Jussi Kangasharju, University of Helsinki, Finland

ISSN 1238-8645
ISBN 978-952-10-8340-2 (paperback)
ISBN 978-952-10-8341-9 (PDF)
Computing Reviews (1998) Classification: C.2, C.2.0, C.2.1, C.2.2, C.2.4, C.2.5, C.2.6
Helsinki 2012
Helsinki University Print

Secure Connectivity With Persistent Identities

Samu Varjonen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
samu.varjonen@helsinki.fi
http://www.cs.helsinki.fi/u/sklvarjo

PhD Thesis, Series of Publications A, Report A-2012-8
Helsinki, November 2012, 139 pages
ISSN 1238-8645
ISBN 978-952-10-8340-2 (paperback)
ISBN 978-952-10-8341-9 (PDF)

Abstract

In the current Internet the Internet Protocol address is burdened with two roles. It serves as the identifier and the locator for the host. As the host moves its identity changes with its locator. The research community thinks that the Future Internet will include identifier-locator split in some form. Identifier-locator split is seen as the solution to multiple problems. However, identifier-locator split introduces multiple new problems to the Internet. In this dissertation we concentrate on: the feasibility of using identifier-locator split with legacy applications, securing the resolution steps, using the persistent identity for access control, improving mobility in environments using multiple address families and so improving the disruption tolerance for connectivity.

In order to quantify the effect of the introduction of the identifier-locator split to the networking applications, we gathered extensive set of data from the Ubuntu Linux Long Term Support versions. From the gathered statistics we characterized the usage of Sockets API. From the statistics we reported ten interesting fingínds about security, IPv6 and configuration issues. Based on our findings we concluded that the Sockets API is heterogeneous and that it is difficult to introduce modifications to the way applications utilize network. We suggested fixes for the security and UDP multihoming support.

Identifier-locator split introduces a new identifier that has to be resolved to an locator, i.e., the IP-address. Solutions for the resolution exist, but the previous studies did not handle security or left security as further study. Based on these observations on resolution systems, we described an architecture for secure identifier-locator mappings based on a Distributed Hash Table. The architecture we describe solves three core problems: a) support for flat namespace, b) frequent user-based updates, c) the security of the architecture, by using the cryptographic properties of the identifiers in the Host Identity Protocol (HIP). Our performance analysis of the HIP-enabled DHT demonstrate the feasibility of our architecture.

For corporations and communities the access control of hosts is rather easy, for example when a host is taken into use its identity is included into the firewall configuration. For home user it is not so easy to manage the identifiers that may needed to establish an inbound connection. We designed a DHT-based system to distribute the information of friendships that is based on the self-certifying cryptographic identities of HIP. The system relies on one-hop trust paths to be used to access control the incoming connections. The solution we described can be generalized for situations where the before-hand inspection of the content is impossible or otherwise hard to implement, for example middleboxes may not have the storage to delay large transfers.

We argued that identifier-locator split protocols can solve three major challenges that the Internet architecture is facing: a) IPv4 address space is too small, b) end-to-end connectivity is broken due to Network Address Translations (NATs), c) the Internet architecture lacks a mechanism that supports end-host mobility in the transition state Internet. We demonstrated that cross-family handovers can be used to alleviate the transfer from IPv4 to IPv6. We described a shortcoming in current HIP mobility specifications preventing cross-family handovers and suggested a simple solution to it. Our performance evaluation with our implementation indicates that HIP-based cross-family handovers perform as well as intra-family handovers.

Computing Reviews (1998) Categories and Subject Descriptors:

C.2	Computer-Communication Networks
C.2.0	General
C.2.1	Network Architecture and Design
C.2.2	Network Protocols

C.2.4 Distributed Systems
C.2.5 Local and Wide-Area Networks
C.2.6 Internetworking

General Terms:
Identifier-Locator split, Security, Host Identity Protocol, Mobility, Distributed Hash Table, Domain Name System, Cryptographic Identifiers

Additional Key Words and Phrases:
Improving secure connectivity with persistent cryptographic identifiers

Acknowledgements

I would like to thank multiple people who have on their behalf influenced my research that resulted in this dissertation.

I thank Jukka Manner and late Kimmo Raatikainen especially for getting me started on this journey.

I extend my appreciation and thanks to my supervisor Jussi Kangasharju and my instructors Andrei Gurtov and Sasu Tarkoma for all the advice and help I got from them during the journey. I would also wish to thank Kristiina Karvonen for all the helpful discussions.

I thank my pre-examiners Jarmo Harju and Mika Ylianttila for their work and feedback for improving the dissertation. I would also wish to thank my assigned opponent Hannu H. Kari.

I am grateful for all the help and advice I received from my colleagues at Helsinki Institute for Information Technology: Miika Komu, Joakim Koskela, Boris Nechaev, Dmitry Korzun, Dmitriy Kuptsov, Andrey Khurri, Ken Rimey, and Oleg Ponomarev.

I was lucky enough to work with very inspiring researchers outside the University of Helsinki. I would like to thank especially Tobias Heer and René Hummen from the RWTH Aachen.

Last but not least, I extend special thanks to my family. Without the encouragement from my wife Noora and my children Vilho and Sanni, I would not have been able to complete this.

Helsinki, October 2012
Samu Varjonen

Contents

List of Figures

List of Tables

Abbreviation list

3DES	Triple-DES.
3G	Third Generation.
A	Address.
ACE	Adaptive Communication.
AES-CCM	Advanced Encryption Standard - Counter with CBC-MAC mode.
AH	Authentication Header.
API	Application Programming Interface.
ARP	Address Resolution Protocol.
AS	Autonomous System.
ASN.1	Abstract Syntax Notation One.
BBM	Break-Before-Make.
BER	Basic Encoding Rules.
BEX	Base EXchange.
BSD	Berkeley Software Distribution.
C	Country.
CA	Certification Authority.
CBA	Credit Based Authorization.
CN	Correspondent Node.
CN	Common Name.
CNAME	Canonical Name.
CoA	Care-of-Address.
CoDoNS	Co-operative Domain Name System.
CPU	Central Processing Unit.
CRC	Certificate Result Certificate.
CRL	Certificate Revocation List.

D-H	Diffie-Helman.
DCCP	Datagram Congestion Control Protocol.
DES	Data Encryption Standard.
DES-CBC	Data Encryption Standard - Cipher Block Chaining.
DHCP	Dynamic Host Configuration Protocol.
DHT	Distributed Hash Table.
DN	Distinguished Name.
DNS	Domain Name System.
DNSsec	Domain Name Security Extensions.
DOI	Domain Of Interpretation.
DoS	Denial-of-Service.
DSA	Digital Signature Algorithm.
DSMIPv6	Dual-Stack Mobile IPv6.
EID	End-Host-IDentifier.
ESP	Encapsulated Secure Payload.
FA	Foreign Agent.
FQDN	Fully Qualified Domain Name.
GNU	GNU's Not Unix!.
GUI	Graphical User Interface.
HA	Home Agent.
HDRR	HIP DHT Resource Record.
HI	Host Identifier.
HIP	Host Identity Protocols.
HIPL	HIP for Linux.
HIT	Host Identity Tag.
HPC	High-Performance Computing.
HTTP	Hyper Text Transfer Protocol.
I1	First Initiator packet.
I2	Second Initiator packet.
IAN	Issuer Alternative Name.
ICMP	Internet Control Message Protocol.
ICMPv6	Internet Control Message Protocol version 6.
IDEA	International Data Encryption Algorithm.
IETF	Internet Engineering Task Force.

IKE	Internet Key Exchange.
IKEv2	Internet Key Exchange version 2.
ILNP	Identifier-Locator Network Protocol.
IP	Internet Protocol.
IPsec	IP security architecture.
IPv4	Internet Protocol version 4.
IPv6	Internet Protocol version 6.
IRC	Internet Relay Chat.
ISAKMP	Internet Security Association and Key Management Protocol.
ISP	Internet Service Provider.
LDAP	Lightweight Directory Access Protocol.
LISP	Locator/Identifier Separation Protocol.
LSI	Local Scope Identifier.
LTS	Long Term Support.
MBB	Make-Before-Break.
MD5	Message Digest series 5.
MIP	Mobile IP.
MIPv4	Mobile IP version 4.
MIPv6	Mobile IP version 6.
MN	Mobile Node.
MOBIKE	Mobility and multihoming extensions for IKEv2.
MTU	Maximum Transmission Unit.
MX	Mail Exchange.
NAT	Network Address Translation.
NS	Name Server.
NXDOMAIN	Non-Existent Domain.
O	Organization.
OS	Operating System.
OU	Organization Unit.
P2P	Peer-to-Peer.
P2PSIP	Peer-to-Peer SIP.
PFS	Perfect Forward Security.
PGP	Pretty Good Privacy.

PIN	Personal Identification Number.
PKI	Public Key Infrastructure.
POSIX	Portable Operating System Interface for uniX.
PRNG	Pseudo Random Number Generator.
R1	First Responder packet.
R2	Second Responder packet.
RC4	Rivest Cipher 4.
RC5	Rivest Cipher 5.
RLOC	Routable LOCator.
RR	Resource Record.
RRSet	Resource Record Set.
RRSIG	Resource Record Signature.
RSA	Rivest Shamir Adleman.
RTT	Round-Trip Time.
RVS	RendezVous Server.
SA	Security Associations.
SAD	Security Association Database.
SAN	Subject Alternative Name.
SCTP	Stream Control Transmission Protocol.
SD	Service Description.
SHA-1	Secure Hash Algorithm variant 1.
SHIM6	Site Multihoming by IPv6 Intermediation.
SIP	Session Initiation Protocol.
SN	Surname.
SPD	Security Policy Database.
SPI	Security Parameter Index.
SPIT	Spam over Internet Telephony.
SPKI	Simple Public Key Infrastructure.
SSH	Secure SHell.
SSL	Secure Socket Layer.
SSLv2	Secure Socket Layer version 2.
SSLv3	Secure Socket Layer version 3.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
TLSv1	Transport Layer Security version 1.
TTL	Time-To-Live.

UA	User Agent.
UDP	User Datagram Protocol.
UMTS	Universal Mobile Telecommunications System.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
VoIP	Voice over IP.
VPN	Virtual Private Network.
WLAN	Wireless Local Area Network.
WOT	Web Of Trust.
XML-RPC	eXtensible Markup Language - Remote Procedure Call.

Chapter 1

Introduction

In the current Internet we face at least two major problem categories: mobility and security. First, we have Internet Protocol (IP)-addresses that are used as the locators for the hosts. However, IP-addresses are currently used also as the identifiers for the hosts. In practice, this means that the identity of the host changes when the mobile host changes its attachment point. Second, for transport protocols this is fatal, for example, Transmission Control Protocol (TCP) sessions will break upon such an event. Multihoming has traditionally been a concern only for servers and their fault-tolerant networking. The introduction of such technologies as Wireless Local Area Network (WLAN) and Third Generation (3G) have introduced the need for multihoming to common users and their equipment. For applications this introduces problems as they have no practical way to handle multihoming. Identifier-locator split has been identified as a promising solution to combat these problems.

Identifier-locator split protocols achieve these goals. Currently discussed identifier-locator split protocols follow one of two principles: address rewriting or mapping and encapsulating. In the address rewriting method, an Internet Protocol version 6 (IPv6) address is divided into front and back half. The front half of the IPv6 address represents the locator of the host and the back half represents its identity. The Identifier-Locator Network Protocol (ILNP) [14] is a protocol that implements the address rewriting method. The deployment of address rewriting schemes requires major renumbering in the network and compulsory support for IPv6 (because of the longer address format). The current Internet is in a transition phase towards IPv6. However, IPv6 connectivity cannot be guaranteed everywhere yet, hampering the immediate deployment of protocols that solely rely on IPv6.

In mapping and encapsulating schemes, additional identifiers are mapped to locators and packets are encapsulated. Locators are only used in the packet headers at the network layer. Mapping and encapsulating approaches can be divided into two categories: network based and host-based. The Locator/Identifier Separation Protocol (LISP) [34] is an example of a network-based approach. Host Identity Protocols (HIP) [91] is an example of a host-based protocol. Mapping and encapsulating-based approaches have the benefit that they work on top of Internet Protocol version 4 (IPv4) as well as on top of IPv6. In addition, mapping and encapsulating schemes do not require changes to the core routing of the Internet. In this thesis we concentrate on identifier-locator split protocols that implement the mapping and encapsulating scheme and in more detail to host-based approaches, which use additional identifiers that are used in and above the transport layer. Additionally we concentrate on the problems caused by the mobility and the related solutions.

We start with a study on the usage of the Portable Operating System Interface for uniX (POSIX) socket Application Programming Interface (API) (aka. Berkeley sockets). We gathered statistics on the usage of functions and definitions in application source code related to network communications. We selected Long Term Support (LTS) Ubuntu distribution versions and the bleeding edge version Maverick Meerkat (10.10) as our source for applications and their source code. Based on these statistics we draw out a image of how networking is handled in current applications. The main goal of this study is to find out whether we can use current applications with end-host based identifier-locator split protocols.

While the introduction of identifier-locator split brings us many benefits it complicates resolution systems. Currently we need to resolve Fully Qualified Domain Names (FQDNs) to locator (e.g., IP-address). With identifier-locator split we add an additional End-Host-IDentifier (EID) in between FQDN and EID. Moreover, the usage of identifier-locator split requires all hosts to have resolution records in the system, in comparison to current DNS that mainly contains records for stationary servers. In practice this means an extra resolution step with additional complications due to security and efficiency.

When the connection is made to a host we face a question, whether to accept the connection or not. The problem there lies in the question: do we know the host's EID or not? Furthermore, do we know what is the purpose of this connection. In our research we use Voice over IP (VoIP) as an example. VoIP suffers from *spit* that is similar to email spam. We show how persistent cryptographic identities with introducers can improve the situation. The approach is based on distribution of certificates and it requires the initiator of the connection to find the trust-path between itself and the responder of the connection.

Now that we have the connection between hosts we face mobility full on. The current Internet uses both address families (i.e., IPv4 and IPv6 and some parts use just either. In this transition phase Internet we cannot guarantee that the connection can survive after handovers. Identifier-locator split helps in this by masking the mobility behind the EID so that from the perspective of the application nothing happened. Our study improves the situation by describing a shortcoming in current HIP mobility specifications preventing cross-family handovers and suggests a simple solution to it that is compatible with Network Address Translation (NAT)ted networks.

1.1 Problem Statement

The IP was designed in an era when the user equipment was stationary and scarce. The situation has changed. In the current day Internet there are ever more devices. Moreover, Internet has a dynamic nature, due to larger population of mobile users.

Due to the growing number of nomadic users we need a way to contact users in various addresses in a secure manner. The research community has proposed Identifier-locator split protocols as a solution for this problem.

Identifier-locator split solves the majority of the problem. However, the current research leaves open questions: deployment with legacy applications, secure resolution of identifiers, access control, mobility in multi-family environments.

1.2 Contributions

The main contributions of this dissertation are:

- *Statistics and empirical experiences on the sockets API.* We gathered extensive set of data that we analyzed. The analysis tells us what are the current practices to use the sockets API in networking applications. We reported ten interesting findings that include security, IPv6, and configuration related issues. Based on the findings we conclude that the Sockets API usage is heterogeneous and that it is difficult to introduce general modifications to the way applications utilize networking features. We partly addressed the extent of this challenge by suggesting fixes on security and UDP multihoming support.

- *Secure resolution system for mobile users.* We showed that while there is much work done on resolution systems for mobile users [86, 33, 4, 81, 15], they lack discussion about how to maintain the information securely and efficiently. We propose a secure architecture for mobile clients using identifier-locator split protocols.

- *A simple way to identify connections from acceptable parties.* We identified two distinct problems in the proposed solutions using trust paths that rendered the solutions inefficient. We addressed these problems by using the self-certifying cryptographic identities of HIP. Our solution is in a sense a distributed white list based on host identifiers that identify the incoming connections from friends.

- *Unified way to transport certificates with HIP.* We provided the specification for the certificate usage in HIP control packets [42] as an additional contribution for the paper [137]. This work continues as an internet draft describing how the hosts can advertise services and requirements for the services, as part of the services the hosts may require the usage of certificates [43].

- *Statistics and empirical experiences on cross-family handovers with HIP.* It was stated in the specification [93] that the handovers across IP families is left for further study. Jokela et al. [54] discussed about the cross-family handovers but presented no performance of implementation work. Furthermore their primary environment was FreeBSD. We presented the cross-family handovers with Linux and specifically concentrated on the fault tolerance aspects of the handovers rather than load balancing. In our work we described the shortcomings of the specifications and suggested simple solutions for them.

Statistics and empirical experiences on the sockets API discussed in Section 3 are based on the published research report [73]. The original idea for gathering statistics came from Komu, who acted as the editor for the report. The author handled the gathering and the calibration of the data. The analysis of the data was carried out as a joint effort.

The secure resolution system for mobile clients discussed in Chapter 4 is based on the published paper [138] [1]. The original idea is from the author but it was further refined jointly with Heer. Implementing the prototype and measuring the prototypes performance were entirely made

[1] This paper received the best paper award in the Globecom 2011 - Next Generation Networking Symposium (GC'11 - NGN)

by the author. Writing of the paper was a joint effort of the author of this dissertation and Heer, with the help of rest of the coauthors.

The separation of friends from spitters discussed in Chapter 5 is based on the research in the publication [137]. The original idea came from discussions with Gurtov. Author made the prototype implementation and performed the measurements presented in the publication. Writing was entirely done by the author of this dissertation. The specification of certificates in HIP is a joint effort with Heer.

Experiences and experiments with cross-family handovers discussed in Chapter 6 are based on two publications [139, 140]. The original idea for the cross-family handovers came from the HIP community and the fact that the specifications did not discuss about the possibility. The original prototype implementation for HIP for Linux (HIPL) [2] and the performance measurements were done by the author of this dissertation. Writing was done jointly by the author of this dissertation and Komu with the supervision of Gurtov.

1.3 Research History

The dissertation research was carried out in four projects, Trustworthy Internet (TrustInet), Infrastructure for HIP (InfraHIP), Secure Peer-to-Peer Services Overlay Architecture (SPEAR) and GoodNet at Helsinki Institute for Information Technology (HIIT).

Security is seen by many as the primary problem of the Internet today. While novel solutions are constantly proposed by Internet researchers, the current rigid Internet architecture makes it difficult to deploy something new, especially if router modifications are necessary. While the scale of security problems in the Internet is immense for any single organization to tackle, the TrustInet project contributed solutions towards a better Internet.

TrustInet project collaborated closely with the Infrastructure for HIP (InfraHIP) project. ”Infra” in the project name stands for Infrastructure. The project focused on developing the missing infrastructure pieces such as Domain Name System (DNS), NAT, and firewall support to enable a widespread deployment of HIP. The InfraHIP project studied application related aspects of HIP, including APIs, rendezvous service, operating system security, multiple end-points within a single host, process migration, and issues related to enterprise-level solutions.

[2] `http://hipl.hiit.fi/`, 22.9.2012

The SPEAR project attempts to design and develop generic mechanisms to support P2P services. To achieve this the main focus of the project is in the integration of support for the HIP based overlay networking environment (HIP-BONE) in the HIP architecture. This way services, such as Peer-to-Peer SIP (P2PSIP) and Peer-to-Peer (P2P) Hyper Text Transfer Protocol (HTTP), can be supported within HIP architecture.

1.4 Structure of Dissertation

In this chapter, we briefly outlined the main research problems and stated the contributions of this dissertation. The rest of the dissertation is organized as follows.

Chapter 2 gives an overview of the used technologies. Section 2.1 presents a brief overview of the resolution mechanims currently in use. In Section 2.2 we give an overview on IP mobility and dual stack mobility. Section 2.3 gives the preliminary information on security mechanism used in this dissertation and an overview on other security solutions supporting mobility. In Section 2.4 we describe the basics of Host Identity Protocol (HIP), including the HIP base exchange and the mobility management

In Chapter 3 we present statistical and empirical observations on the sockets API. In order to understand how network applications behave today, we analyzed 2187 software packages from four Ubuntu Linux distributions. We quantified the use of networking-related system calls, structures and constants. We focused on Berkeley socket API as it is the de facto standard for most networking applications. With this simple methodology, we characterized the capabilities of network applications.

In Chapter 4 we discuss how the resolution with the additional identifiers can be handled in an efficient and secure manner. In Section 4.2 we discuss the problems that a resolution system has to face. We also offer a general solution to these problems and highlight its properties. In Section 4.3 we introduce the details of the resolution system. In Section 4.4 we present a qualitative analysis of the feasibility of our proposal using the Host Identity Protocol as an example. We also provide a high level analysis of the processing times in comparison with the observed processing times of live systems. Section 4.5 gives an overview of the related work.

In Chapter 5 we use spit protection as the means to illustrate how the persistent identifiers can be used for access control. In Section 5.3 we discuss about the usage of trust paths to identify friends amongst spitters. In Section 5.4 we describe the overall system including what is distributed and by what means. Also the flow of control is described in the system when a connection is made between participants unknown to each other. In Section 5.5 we evaluate the behaviour of the system, i.e., latencies of how long it takes to gather the one-hop trust path information, and how much space on the wire does the trust information take.

In Chapter 6 we discuss how to survive mobility in the transitional phase Internet. In Section 6.3, we outline the shortcomings in current HIP mobility specifications, propose a simple solution and share our experience in implementing cross-family handovers with Linux networking stack. We evaluate performance of intra-family and cross-family handovers for TCP flows in Section 6.4.

Chapter 7 concludes the dissertation by summarizing the main contributions made in this dissertation and some future work is outlined.

Chapter 2

Background

In this chapter we give brief introduction to the techniques and concepts used in the later chapters. The related work is described in this chapter.

2.1 Resolution architectures

In order to communicate, the connecting peer must know the IP-address of the responding peer. In the early days of the Internet it was enough for the users to have a file that contained all the needed names and addresses. The Internet has grown significantly from those days and a single file is not sufficient anymore. DNS was designed to store and resolve theses names, i.e., FQDNs to the corresponding addresses.

In the following sections we discuss about the current day DNS and about the suggested systems that may replace or supplement the DNS.

2.1.1 Domain Name System

The DNS forms a hierarchical tree of domain name servers, as it would be inconvenient to administer the Internet as one. In the hierarchy the root name servers maintain the top-level domains and they know which branch, i.e., intermediate name server to contact when information of a sub-domain is needed and similarly until the leaf, i.e., the authoritative name server with the needed information matching the FQDN is found. The data in the DNS is stored into Resource Records (RRs) that contain the information related to the associated name. Common types of the RRs are Address (A) for address to a name, Name Server (NS) for the administrative name server's address, Canonical Name (CNAME) for alias of the name and Mail Exchange (MX) for mail server's name of the domain.

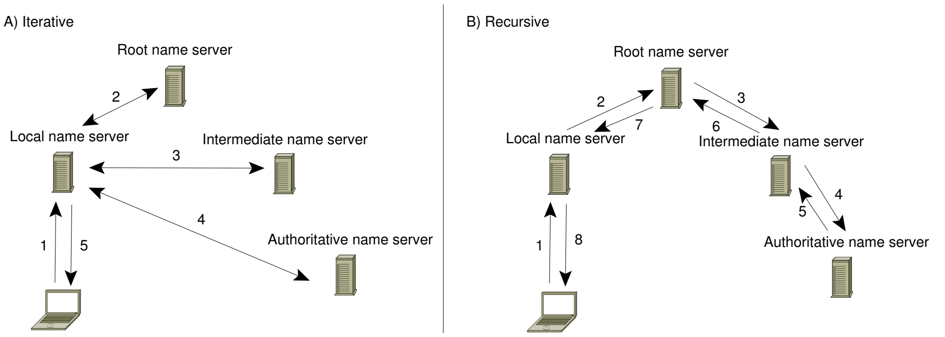


Figure 2.1: Name resolution iteratively (a) and recursively (b) in DNS.

The DNS may handle the queries in two ways: iterative or recursive. First, in the iterative manner (see *A* in Figure 2.1) the query is sent to the local name server that queries the root name server and gets the address of the intermediate name server as the response. Then the local name server contacts the intermediate server from the response and similarly until the authoritative name server with the correct Resource Record Set (RRSet) is found. Second, in the recursive manner (see *B* in Figure 2.1) the local name server queries the root name server and the root name server in turn queries the intermediate name server and the response returns recursively the same route, hence the name. If the given FQDN cannot be resolved into an address, a *name error* is returned [1].

DNS caches can improve the latencies of popular name queries. If the queried name is not found from the local or the nearest cache the query will be resolved as explained above. In the case the name is not found from the cache the latency of the resolution is the same or little longer than without the cache. Upon receiving of the response the successful response is cached and the subsequent queries about that name are served directly from the cache resulting in shorter latencies than without the cache. Caching RRSets locally can so improve the performance of the DNS.

Although, DNS caches can improve the performance, they can be seen as a source of problems [56]. Load balancing and mobility are two issues most affected by the use of caches. In the case of load balancing, it may be beneficial to distribute the load of one server to multiple servers. This is easily achieved by letting the DNS return an address from a pool of addresses in a seemingly random manner. If end-hosts use local caches the the DNS cannot return a new address as the end-host does not make actual query to the DNS but uses the previously cached address. Similarly in the mobility, the end-host uses the address given to it earlier and does not get

[1] NXDOMAIN response containing RCODE = 3, [8]

the current address of the Mobile Node (MN). The issues with the caches can be solved by using lower Time-To-Lives (TTLs) on RRs that belong to load balancing servers or MNs, but this effectively strips the performance benefits of the DNS caches.

Caching can also be implemented for the error cases. In negative caching the name errors [8] are cached. Negative cache's records should have low TTLs as the name could be taken in to use and because storing negative results for a long time could render the system unusable [110]. This is caused by the fact that many of the negative answers are caused by typos and users can make unlimited number of typos or they try to reverse map IP-addresses that do not have reverse mapping [56].

2.1.2 Domain Name System Security extension

Recently there has been more and more malicious behavior against the DNS system and also the client population who has to use DNS has increased. Studies have shown that the legacy DNS is not suitable anymore. This is because of the administrative needs of the DNS and its lack of fast reconfiguration [110, 24]. Attack resilience is considered as the biggest problem of the legacy DNS.

A malicious user can masquerade as a name server by spoofing the IP-address of a legit name server. It is possible that a name server does not check the originating addresses of the RRs when inserting a RR to its database [2]. This attack is known as *cache poisoning* and it allows malicious users to forged RRs. Domain Name Security Extensions (DNSsec) [29] was originally designed to protect against attacks such as the ones described above [3].

DNSsec makes use of public key cryptography and digital signatures in order to provide authentication of the origin and integrity protection for the RRs. DNSsec stores the signatures to a new type of RRs called the Resource Record Signature (RRSIG). DNSsec also adds new header bits to the legacy DNS messages. With the additional bits the client can tell the DNS that the response should include the signatures. Upon receiving the response the client makes additional query to get the used public key and verifies the signature with the received key.

Although, DNSsec improves the security of the legacy DNS it also introduces a new kinds of attacks [9, 10, 11]. For example, a malicious user can force the DNSsec enabled name server to waste its resources on cryptographic tasks. Another security problem of DNSsec is called zone enu-

[2] This is common configuration problem

[3] For more details about threats against the DNS that the DNSsec solves see [13].

meration or zone walking. In zone enumeration the malicious user issues false queries toward the system and finds out the topology of the attacked network. This works as the DNSsec responds to a non-existent name with the signed name of the next existing name [4]. Only slave servers of the primary master name server should be able to do zone enumeration and although the namespace information is public it is considered to be bad to let outsiders have access to the whole namespace contents [9] [6, p. 356 – 358]. [79] was developed to protect against the zone enumeration attack by giving a possibility to return a signed 3NSEC record instead of the NSEC record. 3NSEC record contains the hashed value of the NSEC record.

2.1.3 Performance of Domain Name System

Ever growing client population results in growing namespaces. Growing client population affects performance and results in the need to have bigger namespaces. While namespace grows, it has been shown that it does not grow evenly. Ramasubramanian and Sirer noticed that most of the new names go into the popular domains such as *.com* [110]. This on its behalf skews the load distribution in legacy DNS. This uneven distribution causes the load of the servers to distribute unevenly, so that the servers administering popular namespaces have higher load. Pang et al. noticed that DNS servers with high load usually are more available [102]. This can be caused by constant maintenance, better hardware and redundant network connections. They also noticed that most of the users use only a small fraction of the available servers [102].

Failures in the DNS system can be caused by anything from incorrect configuration of the servers to simple typos made by users. Albitz and Liu state that most of the failed queries are caused by improperly configured servers. Their book also lists the most common configuration mistakes [6]. In the studies made at the MIT they showed that 23 % of the queries get no answer. This result is affected by the retransmissions in the network. In their paper Jung, Sit and Balakrishnan found that only 13 % of the queries actually result in an error [56]. In that study it was noticed that those errors were caused by missing reverse mappings and incorrect NS records. In their paper they suggest a modification to the DNS that could improve the performance of the legacy DNS system's root servers. The improvement was suggested because they noticed that from 15 % to 27 % of all the root server traffic results in negative answer and that most of these errors were caused by typos pointing to non-existent names. These errors could be

[4] the next NSEC record

from users but also from incorrectly implemented resolvers. They suggest that intermediate servers should refuse to forward mall-formed queries.

Currently users have more and more mobile equipment and they have the need to be contacted on the move. Mobility can cause user's IP-address to change from network to network depending on users movement. The legacy DNS can be considered too static for this kind of usage. Because DNS was designed when the equipment was more or less stationary, it was not taken in to account that someday fast user-based updates of the RRs could be needed. For example a user could use a VoIP software on a mobile equipment. If a call is made trying to contact the user the system has to have a way to resolve user's current address. This can be implemented in many ways, but the idea in most of them is to use one static server called RendezVous Server (RVS). In this approach the user's RRSet points to this RVS and the user updates its current address directly to the RVS.

2.1.4 DNS over Distributed Hash Tables

For the reasons, described in the previous section, the research community has proposed supplementary systems and even systems that could replace the DNS. The main concern is the performance, which can be divided into categories: availability, attack resilience, lookup latency, failure rate, and load distribution.

Availability: In the legacy DNS a malfunctioning or down server can effectively separate the client from the network. The separation might not be complete depending on the server that is down. The client might be able to resolve names of the local network or names in some partition of Internet. Although, the client cannot use DNS to resolve the names to addresses, it may be possible that the connection to the peer could be made if the client has some other way to resolve names. DNS caches can be used to provide this. While caches are effective in providing resolutions, they can be problematic as discussed in Section 2.1.1. DNS over Distributed Hash Table (DHT) systems can be considered to have benefits in this category as DHTs are self-healing and they support replication of data they store. This means that the DHT is not affected by the churn. In overall the distributed systems can perform better in this category.

Attack resilience: This category is partly a sub-category of the availability, because of the Denial-of-Service (DoS) attacks. The main idea in DoS is to flood the target with so much bogus work that the target cannot reply to any valid queries. In the legacy DNS it is fairly easy for an attacker to find the attack points that have significant effect on the system performance, when the attack succeeds. In a DHT based system it is harder for the attacker to find a *weak* point. Rest of this category consist of attacks targeting stored data. In the legacy DNS it is hard for an attacker to inject false data to the servers, but the attacker could spoof answers from DNS.

Lookup latency: Latency is considered to be the time between sending of an query and receiving the answer from the resolving system. There are several things that affect the latency, e. g. the usage of caches and DNSsec. The caches tend to lower the latency but may result in stale records as discussed in 2.1.1. If the client wants to be sure that the answer is completely valid, it needs to query all the keys and the signatures up to the root level and this can result in significantly longer latencies.

Failure rate: This category considers the availability of the data. Failure rate is an estimate of the amount of negative answers from the system, like the name error message discussed earlier in 2.1.1.

Load Distribution: This category considers the distribution of RRSets in the system. In the legacy DNS the hierarchy was designed so that the data would distribute in an even manner to the name servers. It has been shown that this does not work in the current DNS. In the DNS over DHT systems the DHT provides the better balance for the load by hashing the key under which the value is stored.

DHTs are distributed systems that provide decentralized storage services where data is saved in key-value pairs that are distributed into the system based on their key. In other words, DHTs provide scalable and failure tolerant storage systems. DNS over DHT systems can handle DoS attacks better than hierarchical legacy DNS because there is no hierarchy in the system and because the data is replicated to a set of servers, usually from 6 to 8 servers depending on the underlying DHT. Replicating servers can also be chosen in pseudo random manner. This makes it harder for a malicious user to bring down the system as there is no single attack point in the system. A malicious user would have to bring down a large set of servers before effect of the attack is noticeable [110].

Load balancing in the DNS over DHT is better than in the legacy DNS. This is caused by the consistent hashing that the DHT systems use. In DHT systems the keys are hashed before the systems know which node should save the key-value pair. This results in a fairly even load balance throughout the system. In the legacy systems the storing server is decided over the hierarchical parameters. In other words the server that controls that part of the namespace stores the data. In the DNS over DHT systems the zone data is basically removed and there is no one node handling certain part of the namespace.

One of the biggest problems of the DNS over DHT systems is the usage of the DNSsec. The systems are still required to have access to a secret key belonging to a well known Internet Service Provider (ISP) or get a well known ISP to sign the DNS over DHT system's RRSets to be trusted. Ramasubramanian and Sirer propose that the signature could be bought from a respectable namespace owner [110]. Nothing prevents the user from signing the RRSet with their own key but then the problem becomes similar than in Pretty Good Privacy (PGP), i.e., whose signature is trustworthy. Above all is the fact that the DNSsec is not that widely deployed.

In the following sections we give a short overview on DHTs and on selected DNS over DHT systems. The selected systems approach the subject from different angles. DDNS [24] can be described as a distributed cache for the legacy DNS. Co-operative Domain Name System (CoDoNS) [110] on the other hand can be described as a peer-to-peer replacement for the legacy DNS. Although, CoDoNS can also cooperate and act as a distributed cache for the legacy DNS.

Distributed Hash Table

Peer-to-Peer (P2P) systems are overlay networks for storing and relaying data. P2Ps systems can be divided into two sub-categories: unstructured and structured. Unstructured systems do not impose any structure to the overlay networks, hence the name. Unstructured overlays are resilient to churn but are not efficient when searching unpopular data. Popular unstructured networks include: Napster [28], Gnutella [58], Freenet [19], eMule/eDonkey2000 [88], BitTorrent [20]. Structured systems impose structure for the overlay network, e.g., DHTs such as Chord [131], Tapestry [147], and Pastry [119]. In this thesis we concentrate on the structured overlay networks.

In structured systems the topology is tightly controlled and any file can be located in a small number of overlay hops (commonly in $\mathcal{O}(log\ n)$ hops). Structured systems usually form a ring geometry, but can form trees, hypercubes, etc... Nodes in the structure overlay get an identifier, that is usually formed by applying a hash function to the nodes IP-address. This identifier determines the nodes place in the topology of the overlay. When data is stored into the overlay or when data is searched from the overlay, the data is hashed into an identifier, similarly as when creating a node identifier for a participating node. This data identifier is then routed to the closest higher node identifier and the data should reside on that node. This works as all the nodes maintain a routing table, in Chord this is called the finger table, which points to nodes further in the node identifier space. Usually the finger table's first entries point closer in the identifier space and farther we go in the table farther the nodes are in the identifier space [5].

When a node joins the system it uses a consistent hash to generate its node identifier. It then contacts the overlay in order to find its successor's identifier. This node is then marked as the new nodes successor. The joining node then takes over part of the successors load. When a node detects a failure on a finger that is in its finger table, the node uses the next best finger to forward traffic. If the finger has been silent for a while, it is removed in the routing maintenance loop.

Bamboo-DHT

In this section we give a brief explanation of DHTs using the Bamboo-DHT implementation. Bamboo-DHT is chosen as later chapters use it, or the OpenDHT instantiation of Bamboo-DHT, or systems that use the similar interface. Little differences exist in the exact syntax and the handling of the TTL depending on the underlying DHT infrastructure, but the basics are the same.

Bamboo-DHT was originally based on Tapestry [146], but is currently considered as a Pastry [120] like system or as a completely new system. In Bamboo-DHT the nodes of the overlay network are organized as a ring as in [131]. In the ring nodes are assigned identifiers based on their IP.

Bamboo-DHT provides a simple put, removable put, get and remove API. Put messages are defined as *put(key, value, TTL)* and get messages are defined as *value = get(key)*. Removable put is defined as *put(key, value, H(secret), TTL)*. If the secret parameter is left NULL the removable put message is treated in a similar manner as the regular put. The remove message is defined as *rm(key, H(value), secret, TTL)*. When a put message is delivered to a node in the overlay, the message is routed to the responsible node that stores the value. The network finds the responsible node by comparing the key and the node IDs. The node with closest identifier to the used key stores the value.

Bamboo-DHT supports replication where the data is saved to the responsible node and to its replicas. In the case where the subsequent put message has same key and value as the original put, the TTLs of the messages will be compared and larger of the TTLs is updated to the DHT. In case where only the key is same but the values are different, the value will be saved under the same key. Every key-value pair has a TTL value that is a value between 1 and 604,800 seconds. TTL tells the system how

[5] common way to create this is to have the first finger point to identifier $2 + 2^0$ or the closest node with a higher identifier. The second to $2 + 2^1$ and the third to $2 + 2^3$ and so on

long it should be stored in the system. Remove message should have longer TTL than the original put message. This is because the replication can cause the original put to reappear. This happens when the put is made and replicated and one of the replicating nodes goes down. If the TTL is shorter than in original put, the remove message is removed from the system before the replicating node comes back. If the remove is not stored over the TTL from the system the replicating node starts the replication process and the key-value pair will reappear.

Removable put includes the hash of the secret. The hash is used to identify the original issuer of the put. In the remove message the originator of the put message reveals the secret to the system. Hash of the value is used to identify the correct value under one key in cases where there is several values.

Co-operative Domain Name System

CoDoNS servers work on top of Pastry [120] and Beehive [111]. CoDoNS uses the same message format and the same protocol as the legacy DNS system, making it easy for them to inter-operate. CoDoNS servers are organized as a ring and the nodes are assigned an identifier, which is used to tell which node is the home node for a given RRSet's keys hash. CoDoNS system offers fault tolerance by replicating the RRSets to n number of adjacent nodes from the key's home node. In the case where the home node of the RRSet goes down, the next node in the ring will take its place in the system.

Users can directly insert records to CoDoNS. When a client issues a query to the CoDoNS system, the contacted node will reply immediately, if the node is the key-value pair's home node, otherwise the message will be routed to the responsible home node. If the home node does not have the RRSet queried, the home node forwards the query to the legacy DNS and from that answer the RRSet will be cached to the home node and pro-actively checked from legacy DNS (see fig. 2.2).

CoDoNS servers support direct caching where the users can insert RRSets only to their own CoDoNS server and instruct their server to serve RRSets rather from their own cache than from the whole system. CoDoNS system also supports reverse queries where an IP-address is resolved to a name rather than a name to IP-address. Most of the DHT based systems have a TTL to limit the lifetime of the key-value pairs. CoDoNS system does not have its own TTL but instead uses the RRSet's TTL to identify when it needs to update RRSet from legacy DNS. Because of the load balancing done in legacy DNS the CoDoNS system will not store RRSets with low

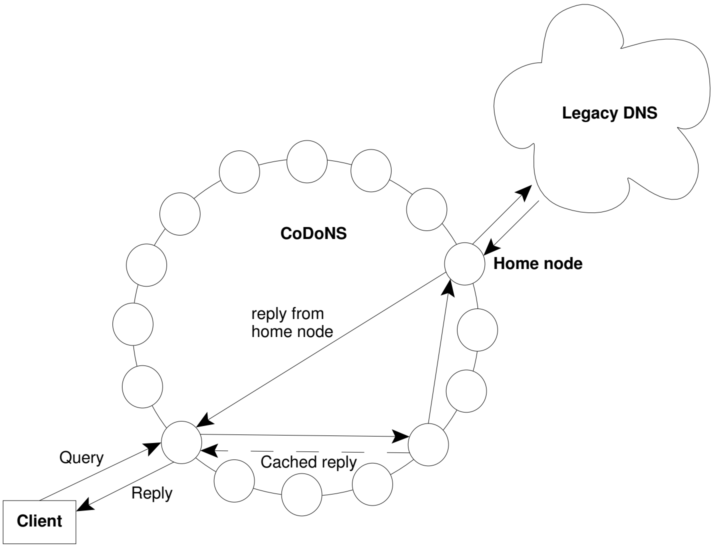


Figure 2.2: CoDoNS architecture. Client sends a query to CoDoNS and the query is forwarded to the home node. If the home node does not know the name queried, it asks it from the legacy DNS. If a node on the way finds the answer from its cache it may answer directly (dashed line).

TTL. Usually RRSets that have low TTLs are used with load balancing or indirection systems. CoDoNS system supports negative caching, where the Non-Existent Domain (NXDOMAIN) messages are cached to the system for short periods of time.

Ramasubramanian and Sirer tell in their studies that their system can outperform legacy DNS system, at least when considering the bandwith and the storage requirements. In their paper [110] they say that CoDoNS system uses less storage base and less bandwith. Exact reason for this result was not clear, because the protocol is similar and uses the same message format as the legacy DNS. Similarly as in DDNS, CoDoNS is also more attack resilient than legacy DNS. In CoDoNS system the latency can be shorter than in legacy DNS, but it can also be much higher. Higher latencies occur when the value is not found from the system and the query is forwarded to the legacy DNS. Then the latency includes all the routing inside the CoDoNS system and the forwarding and routing in legacy DNS and the way back.

DDNS

DDNS[6] is a Domain name system that uses DHash and DNSsec. DDNS

[6] Not to be mixed up with Dynamic DNS defined in [141]

uses DHash's ability to balance load and the fact that DHash is self-healing, meaning that it does not suffer from the *churn*, i.e., leaving and joining of nodes. This is provided by the underlying DHash, which is an overlay on top of Chord [131]. In Chord the nodes are organized in a ring like fashion. Every time a get message is issued to a node it will be routed through the ring to the node responsible for storing the key-value pair. When the response is routed back, the nodes along the way save the key-value pair to their caches and so shortening the answer time of subsequent queries. To provide the robustness DHash replicates the key-value pairs to a pseudo randomly chosen set of six servers. DDNS uses Secure Hash Algorithm variant 1 (SHA-1) hash of FQDN concatenated with resource type as the key to store the value, e.g., SHA-1(www.testcompany.com—A).

For every put and update message in the DDNS, DHash verifies the signature before accepting the message. The same applies on the client side, the client should verify the signature included in the result of the query before using it. Cox et al. propose to publish keys in KEY record type for popular names, to minimize the need to do verifying along the way for every node the query traverses [24].

While DDNS seems to be efficient replacement for legacy DNS, it lacks support for load balancing features implemented in legacy DNS [7]. Legacy DNS is capable to balance load to a set of servers by returning the addresses in random order. Some hosts such as www.google.com use round robin style load balancing where addresses revolve. Moreover, DDNS is distributed but it still needs the authoritative hierarchy to supply it signatures for the DNSsec. When a user wants to add a domain name to the system, the user needs to have access to some well-known ISP's secret key. In reality the client can ask the well-known party to sign its RR or buy the signature from a well-known and trusted company like Verisign. If there is no Certification Authority (CA) hierarchy, Web Of Trust (WOT) [2] approaches could be adopted.

Cox et al. [24] show in their results that the distributed systems are more resistant to DoS attacks. This is based on the fact that it is harder for the malicious user to find a single weak point in the system where all the machines are as worthy as the other. So no one machine is more important than the other. Cox et al. also showed in their paper [24] that the underlying DHash implementation improves the latency for the popular name queries. This is a result of the Dhash's way of caching answers on the way. Every node that routes the query to the node that stores the queried

[7] Here the load balancing is considered as a service for the owner of the RR and does not mean how the load is distributed over the DNS

value, stores the answer for short times on their local caches.

2.1.5 Comparative performance

It is difficult to compare the legacy DNS and the DNS over DHT systems. Performance information has been gathered from the legacy DNS system by multiple research groups and the performance data from the DNS over DHT systems were gathered from small systems compared to the legacy DNS using different test methods and the size of the systems. Performance tests done with DDNS were results from a rather small setup and of limited time interval. In CoDoNS system the performance results were based to a circa 70 server setup on planetlab [8] with a load from varying group of volunteer test users.

When discussing about the latency, the legacy DNS outperforms the DNS over DHT systems. For example, studies made by Cox et al. show that DDNS system has median latency of circa 350 ms and the legacy DNS has median latency of circa 43 ms [24]. Pappas et al. show similar results in their studies [103]. In their paper the results were similar in normal operation conditions. When they introduced high node failure to the systems, the DNS over DHT systems outperformed the legacy DNS. This is a result from the replication features in the DHT systems. Pang et al. studied the DHT based systems in a more general manner and compared the results against the similar setups with the legacy DNS. In their studies they showed that the DHT systems suffer from their maintenance loops. Over certain intervals the DHT systems fix their routing tables and check if their neighbor lists are up to date. In their studies they showed that the performance of the DHT based systems can be improved by extending the maintenance interval. The extension also had its downside, by extending the maintenance interval the self-healing features of the DHT suffered, making the system less available.

Gupta et al. showed in their study [36] another way to improve the latencies in DHT based systems. They suggested a one-hop routing scheme. In that every node in the network would have complete information of the network. Their studies showed that one-hop routing scheme can handle systems consisting of even 2 million nodes. For systems larger than that they introduced two-hop routing scheme. These schemes introduced a pseudo hierarchy to the normal the DHT ring. The ring was divided into slices and they were divided further into units. Both slices and units had leaders, chosen from the middle of the slice or unit. The leaders of the slices multi-

[8] `https://www.planet-lab.org/`, 22.9.2012

cast the routing table changes to their subordinate unit leaders and other slice leaders. The unit leaders multicast routing table changes to their subordinates. This is done to reduce the traffic caused by the routing table changes in the system. Gupta et al. also discussed about the supernode and inner ring of supernodes concepts. They suggested that there could be a formally maintained inner ring of supernodes. These supernodes would be maintained by ISPs and governments. The intention was to provide a stable base system where every one could join in.

2.2 Mobility

The Internet was designed in an era when the hosts did not move. The closest thing to mobility was the unlikely relocation of hardware to a new physical location. Moreover, the IP-address of the host changed in the relocation. In the current Internet, due to the increasing popularity of wireless networking, mobility gains popularity in an ever increasing speed introducing a need to keep the host reachable no matter where they are, and to keep the ongoing connections intact during the mobility events.

In the following sections we discuss about the mobility as it is handled in the current day specifications.

2.2.1 Mobile IPv4

The Mobile IP version 4 (MIPv4) offers transparent mobility for hosts, meaning that MIPv4 does not require any changes above the network layer. MIPv4 defines three entities. First, the MN that is a host that changes the point of attachment in the network. Second, the Home Agent (HA) that is a router on MNs home network that maintains the current location information for the MN. Third, the Foreign Agent (FA) that is a router in a visited network which provides routing and tunneling services for the visiting MNs.

In the MIPv4 protocol the host has two addresses, a permanent one (home address), and an address (Care-of-Address (CoA)) that changes according to the point of attachment. The host can be contacted via the host's HA with the hosts home address. The host's HA then forwards the traffic to the hosts current CoA obtained from the FA.

The mobility agents (HA, FA) advertise their presence in the network by periodically sending advertisements in Internet Control Message Protocol (ICMP) Router Advertisements. From these messages the MN discovers the network it is in. If the network is the MN's home network the MN operates without mobility services. Upon returning to home network MN deregisters its care-of address from the HA. If the network is foreign the MN obtains a new care-of address from the FA of the visited network or by address assignment mechanisms, such as Dynamic Host Configuration Protocol (DHCP) (see Figure 2.3, step 1). The new CoA is then registered with the MN's HA (see Figure 2.3, step 2). When Correspondent Node (CN) sends datagrams to the MN's home address the HA of the MN intercepts the datagrams and tunnels the datagrams to the MN's CoA (see Figure 2.3. steps 3 and 4). The FA in the foreign network then de-tunnels the datagrams and sends them to the MN. When the MN replies to the CN, it can use its current CoA and tunnel all its packets through the HA. Perkins et. al. describe an extension ("route optimization") to the Mobile IP (MIP) which allows the MN to communicate with the CN directly [108]. If the MN in a foreign network initiates a connection it is generally done using standard IP routing mechanisms.

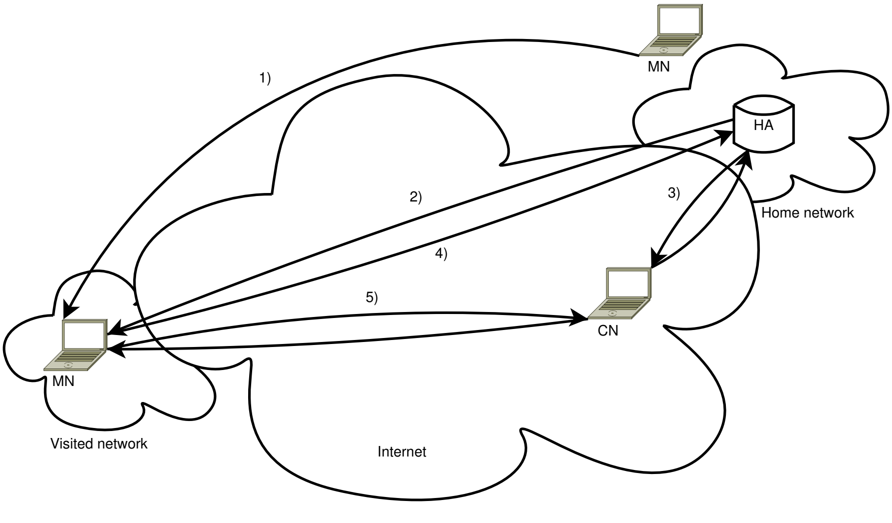


Figure 2.3: Message flow of Mobile IP.

2.2.2 Mobile IPv6

The Internet is depleting its addresses in alarming speed. IPv6 has been defined to offer a larger address base. The Mobile IP version 6 (MIPv6) [52] builds upon the lessons learned from the MIPv4. Although, MIPv6 is very similar to MIPv4 and shares many features with MIPv4, MIPv6 is fully integrated to IPv6 and offers many improved features over MIPv4.

The main difference in the MIPv6 is the support for the routing optimizations by default. By using the routing optimizations MIPv6 avoids the overhead from the triangular routing from which the MIPv4 suffered. Another difference is the dynamic HA discovery.

2.2.3 Dual-stacked hosts

From the existence of the two IP families rises dual stacking and cross-family mobility. In the current Internet we have hosts that implement both IPv4 and IPv6 protocol stacks. The protocol families can run on their own stacks or they can be implemented as a hybrid stacks which accepts both families. Usually hybrid stacks are IPv6 internally and internally encode the received IPv4 addresses to formats such as IPv4-mapped address[45].

This ability to work with both of the families at the same time presents problems for mobility. The current Internet has areas that support only IPv4, IPv6, and some areas support both. When MN moves into this kind of network we cannot guarantee that both of the hosts communicating are in areas supporting compatible addressing.

Dual-stacked MNs could use MIP but the MNs would have to send signaling messages for both protocol versions upon every handover. Moreover, the network administrators would have to run HAs and FAs for both protocol families. Dual-Stack Mobile IPv6 (DSMIPv6) [126] proposes a way to use MIPv6 to support both families.

2.3 Security

Thus far we have been discussing about the connectivity and mobility of the hosts. In this Section we discuss about the relevant security protocols for securing the connectivity and the mobility of the hosts.

2.3.1 Certificates

A certificate is a signed data record containing a name and a public key. A certificate can be interpreted to mean “the key speaks on behalf of the issuer”. The certificate can be used to verify that a public key belongs to an individual.

This section gives a brief introduction to digital certificates, such as X.509.v3 [49] and Simple Public Key Infrastructure (SPKI) certificates [31].

X.509.v3

X.509 is used to bind the public key to the identity of the entity, to whom the certificate was given. X.509 certificates comprise of many fields, common to all certificates. In addition to the common fields X.509 has many extensions, especially version 3, which can be seen as the de facto standard for certificates.

The issuer and the subject are defined with Distinguished Name (DN). DN contains many possible fields, including fields such as Country (C), Organization (O), Organization Unit (OU), Common Name (CN), Surname (SN). DN was originally designed for the X.500 directory service, which was meant as a global directory of entities. This was never realized and now some fields may have contracting meanings in different systems. For the certificates the combination of these fields has to identify the subject or the issuer inside the CA.

X.509.v3 has many extensions. Some of the extensions are marked as critical and they have to be supported on every entity supporting X.509.v3. Usually the critical extensions extend some of the basic fields, such as the issuer, or the subject. The most common extensions are CA, Alternative Name, Key Usage, Extended Key Usage, and Certificate Revocation List. The CA-extension informs that the certificate in question contains the public key of the CA. Alternative name or general name contains additional identifying fields that do not fit to the CN, such as FQDN, IP, or Uniform Resource Locator (URL). Alternative name is a critical extension and can be used in Internet Key Exchange (IKE) in-place of DN. Key usage extension specifies for what purpose the key, contained in the certificate, is for. Extended key usage specifies more usages for the keys. Certificate Revocation List (CRL) distribution points extension is a field that points to the verifier from where to check if the certificate has been revoked.

Simple Public Key Infrastructure Certificates

Simple Public Key Infrastructure (SPKI) certificate is a signed message. The message contains five elements (5-tuple), issuer, subject, delegation, authorization, and validity dates. From these elements only issuer and subject are mandatory while the rest are optional. Issuer is defined as the public key or the hash of the key of the entity who gives out the certificate. The private key of the issuer is used to sign the message and so to create the certificate. Subject is the entity that receives the rights given with the certificate. Subject is defined as the public key or the hash of the key of the entity. The subject, whose key is presented in the certificate, is the only entity that can use the certificate. The delegation field informs about the right to delegate the rights given in this certificate (also known as propagation). If the issuer gives the delegation rights to the subject, the subject can then issue certificates to its subjects. The authorization contains the access/use rights given to the subject. These rights can be freely defined by the issuers. Validity dates define the validity period of the certificate.

If the issuer's private key is compromised the certificate needs to be revoked. SPKI does not have revocation lists which are maintained centrally, such as X.509.v3 has. SPKI allows every issuer to maintain and inform about the location of its revocation list but this is not mandatory.

SPKI certificates are expressed as S-expressions, which is Lisp-like notation for message formats. The S-expressions can be converted to binary format, for transport, and back. The S-expressions and the binary formats serve the same purpose as the Abstract Syntax Notation One (ASN.1)'s Basic Encoding Rules (BER) [59] for X.509.v3.

The access rights often create arbitrarily long certificate chains. The verification and handling of the chains may become a tedious task for the entities. For this reason the SPKI defines a scheme called the 5-tuple reduction. In the reduction the service producer's server, receiving the chain, uses the reduction rules to calculate a single certificate. SPKI also allows the issuer to create a Certificate Result Certificate (CRC). This simplified certificate has the same rigths as the original certificate chain.

2.3.2 IP security

IP security architecture (IPsec) in general is used to create virtual private networks between different networks or between network and a host. For example, a nomadic host may need access to the a network from the outside, with IPsec it is possible to create an encrypted tunnel between the network and the host and allow the host to access the network. It may also be too expensive to build a dedicated network between two networks. IPsec can be used to build an encrypted tunnel between the networks so that it looks like the networks are the one and the same.

IPsec-protocol works on the network layer and extends the already existing IP headers offering a possibility to protect the upper layer protocols. IPsec negotiates a Security Associations (SA) between the communicating parties. The SA defines the receiver's IP-address, protocol identifier, and the Security Parameter Index (SPI). The SPI differentiates multiple simultaneous IPsec connections between hosts. The SAs can be either in transport mode or in tunnel mode. In transport mode IPsec adds headers of its own between the original network layer and upper layer headers. In the tunnel mode IPsec encapsulates the upper layer packets in whole. SAs can be added in layers, for example one SA can protect traffic between the gateways and another SA inside the first one protects the connection between the end-points.

In the next sections we briefly describe IPsec architecture and its encryption, authentication algorithms and key management. In the last subsection we discuss about the differences of IKE and Internet Key Exchange version 2 (IKEv2) [60].

Architecture

Data transfer in TCP networks can be protected in many ways. The used ways depend a lot on the layer needing the protection. IPsec is used to protect the packets on the network layer. IPsec comprises of three major parts, the Encapsulated Secure Payload (ESP) [66], the Authentication Header (AH) [65], and the IKE [39] (see Figure 2.4). AH is used for authentication and integrity protection, ESP is used to encrypt and tunnel the data, and IKE is used to manage the keys needed for the authentication and encryption. IKE itself comprises of two protocols, Oakley [99] and SKEME [75] which work according to the Internet Security Association and Key Management Protocol (ISAKMP) [87] framework.

AH and ESP need to negotiate multiple parameters between the communicating parties. These parameters include, the used encryption algorithm, the used keys and the used options. In IPsec SAs are used to manage these parameters. The SA is defined to be a one-way connection that offers security for the transported data. For this reason, one is needed for the both directions. IPsec keeps track of the ongoing connections in the Security Association Database (SAD). All incoming and outgoing packets are checked against the SAD to see if the messages need security services.

The protection that IPsec offers is based on the security policies, which are defined in the Security Policy Database (SPD). SPD is maintained by the network administrator who can tell which types of packets need security services. The SPD contains lists of selectors that define which packets belong to certain rule. The rules can accurately define which packets get security services [64], i.e., the SPD selectors are used to define which SAD entries should be used to protect the traffic.

Protocols

AH offers authentication and integrity protection services for IP packets. ESP is used for the confidentiality for the IP packets and authentication when needed. These protocols can be used separately or they can be combined so that the desired level of security is achieved. Neither of the pro-

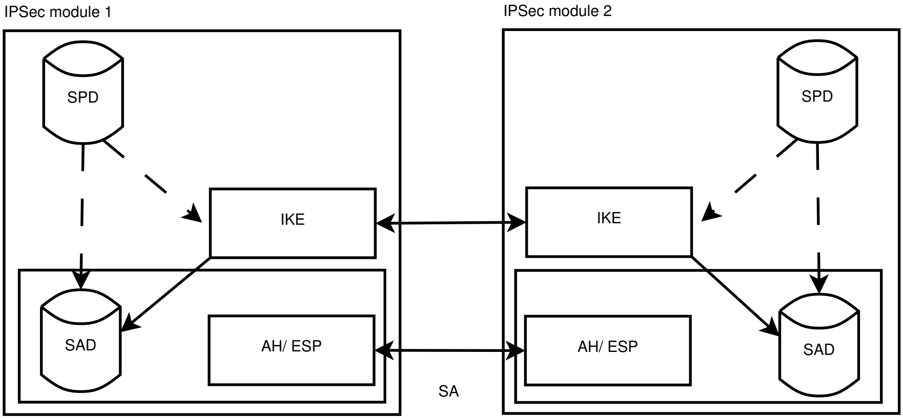


Figure 2.4: IPSec architecture.

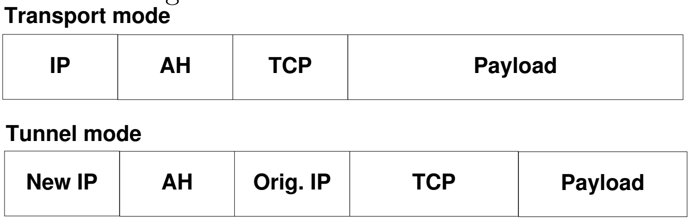


Figure 2.5: Authentication Header.

tocols are fixed to certain cryptographic algorithms, instead the used algorithms can be chosen based on the needs. For compatibility reasons a certain set of algorithms is defined that all the implementation must support. For AH the defined algorithms are Message Digest series 5 (MD5), SHA-1 and others [83, 84, 109]. For ESP, Data Encryption Standard - Cipher Block Chaining (DES-CBC) and others are defined [82, 107]. AH also offers replay protection in addition to authentication and integrity protection.

Integrity and authentication in AH are implemented as additional header to the IP header (see Figure 2.5). AH uses one-way algorithms and secret keys to guarantee integrity and authentication. The specifications also allow public keys to be used with AH. Against replay attacks AH includes a sequence number in to its header that is increased monotonically on every new packet.

Sender computes the needed values for the authentication from the IP packet's fixed headers, also the AH header is included in the computation. Changing fields, such as the IP hop counter are treated as zeros. After the computation the header is added to the IP packet and the packet is sent to the receiver. Upon receiving the packet the receiver then verifies the values. The verification is done similarly as in sending.

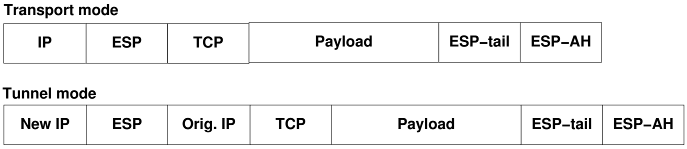


Figure 2.6: Encapsulated Payload.

ESP offers confidentiality services for IP packets, also a possibility for authentication and integrity protection is offered. Authentication and integrity protection are two services that work together and from here on in this section they will be referred only as authentication. ESP has a replay protection mechanism similar to AH. Confidentiality can be used without authentication but this is not encouraged, as the confidentiality only protection is vulnerable to active attacks [16]. Authentication is offered by using the headers defined for AH. ESP encapsulates the original packets between two headers (see 2.6). The ESP tail has three fields, padding, length of padding and next header. Padding field is used when a block cipher is used and the encrypted data has to be of certain length.

When sending an ESP packet the original packet is encapsulated including the original IP header. Possible padding is included and the packet is encrypted. As a final item the used initialization vector is added to the end of the encrypted packet. If authentication is needed it is computed similarly as in AH with the distinction that only the ESP header, the encrypted payload and the ESP tail are included in the computations.

ESP definitions define the following encryption algorithms to be implemented: Triple-DES (3DES) [22], DES-CBC [21], Rivest Cipher 5 (RC5) [116], CAST [3], International Data Encryption Algorithm (IDEA) [78], Blowfish [123], and Rivest Cipher 4 (RC4) [115, 109]. For the authentication the same algorithms are defined as for AH (see above). It is also possible to use NULL in authentication and confidentiality. However, it is recommended that they should not be NULL at the same time. NULL encryption, for example, is useful for testing.

Key Management

IPsec uses cryptographic functions that need keys to work. Key management is one of the main problems in cryptography and it includes key creation, distribution, storage, and revocation, In this section we describe the key management from the IPsec perspective.

Public keys are used in many protocols and applications. These applications are called Public Key Infrastructures (PKIs) when managing large numbers of keys. PKIs are based on CAs who verify and authenticate public keys. To start confidential communications the parties need to be authenticated in order to get the proper rights to get the needed keys to encrypt the traffic. These kinds of protocols are usually called as authenticated key exchange protocols that are further separated to connection and connectionless models.

IKE was designed to offer general authentication and key exchange for IPsec. IKE comprises of two different parts: Oakley, and SKEME.

SKEME is designed specifically for IPsec. SKEME describes a way to transport the keys to the communicating parties. SKEME can use PKIs and it also supports pre-shared keys. The pre-sharing can be done by manually entering the keys or by distributing the keys with Kerberos [68]. With public keys, SKEME uses Diffie-Hellman to draw the keys [112].

SKEME has four different modes, basic with public keys and Diffie-Hellman, key exchange for public keys without Diffie-Hellman, key exchange for pre-shared keys with Diffie-Hellman and fast re-keying for symmetric keys.

The SKEME protocol itself has three phases: SHARE, EXCH, and AUTH. In SHARE phase "half-keys" are exchanged. These half-keys are protected with public keys. Secret key K_0 is hashed from the half-keys. The shared secret key exists when moving to EXCH phase. Depending on the mode either Diffie-Hellman public values, or random values are distributed in the EXCH phase. EXCH phase completes with the computation of the Diffie-Hellman values (i.e., depending on the mode). The exchanged values are verified in the AUTH phase. The verification is done with the shared secret key created in the SHARE phase.

Diffie-Hellman has two parameters p and g that are created from the chosen cyclic group G. Both parameters p and g are public. Parameter p is prime number and parameter g is an integer that is smaller than parameter p. Parameter g can create every element from *1* to *p-1* by $g^n \bmod p$ where n is between *1* and *p-1*. This way all the numbers $g \bmod p, g^2 \bmod p, \cdot\cdot\cdot, g^{p-1} \bmod p$ are different and created from primes between *1* and *p - 1*

When Alice and Bob want to negotiate a shared secret key with Diffie-Hellman they start by creating secret random numbers (Alice a and Bob b). Alice and Bob create the public values using the parameters p, g, and the random value (phase 1 in fig. 2.7). Alice's public value is $g^a \bmod p$ and Bob's $g^b \bmod p$, these values are then exchanged (phase 2 in fig. 2.7). Alice computes a value $(k^a)^b$ by calculating $(g^b)^a \bmod p$ and Bob computes value $(k^b)^a$ by calculating $(g^a)^b \bmod p$ (phase 3 in fig. 2.7). Now Alice and Bob share a secret key K because $(k^a)^b = (k^b)^a = K$ (phase 4 in fig. 2.7).

Oakley is a key exchange protocol [99] that has a lot in common with

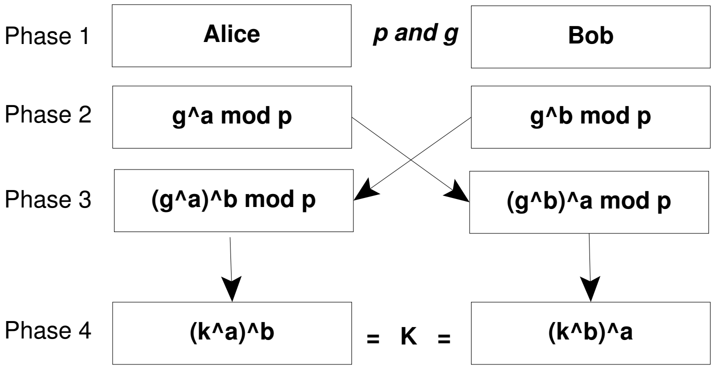


Figure 2.7: Diffie-Hellman.

SKEME. The main difference to SKEME is the possibility to negotiate about the key exchange, encryption and authentication and the ability to create new keys from old keys. The main idea in Oakley is that the sender starts the negotiation by telling the receiver which algorithms the sender wants to use and other needed authentication materials. Receiver replies with his own set of algorithms for use. This negotiation is continued until there is a common understanding of the used algorithms. Oakley comprises of three main parts: cookie exchange, Diffie-Hellman public value exchange, and authentication. Both of the communication parties give one cookie per key, the cookies are used to form the key ids. This key id is used in the creation of new keys. Oakley gives keys for IPsec and describes their usage.

The usage of security services needs the SA management and ISAKMP is designed to negotiate, create, edit, and delete SAs and their parameters. ISAKMP can be seen as generic framework that is not dependent on the underlying mechanisms. ISAKMP has two phases, SA negotiation, and protecting the ISAKMP signaling. In the first phase the parameters for the SA are negotiated, the communicating parties are authenticated, and part of the keys are created. These elements create an SA called the ISAKMP SA. ISAKMP SAs are two-way in comparison to IPsec SAs and they are used to protect the following ISAKMP signaling. In the second phase security assocations for other security protocols are negotiated. Multiple IKE SAs can be negotiated with ISAKMP SA because the lifetime of ISAKMP SAs are longer than IPsec SAs. ISAKMP does not define any specific authentication of encryption algorithm, This enables clear distinction between SA negotiation and key management. ISAKMP offers only basic model for messages which can be changed depending on the payload, for example, SA, key management, identity, authentication, and random value.

The Domain Of Interpretation (DOI) [109] defines the contents for the

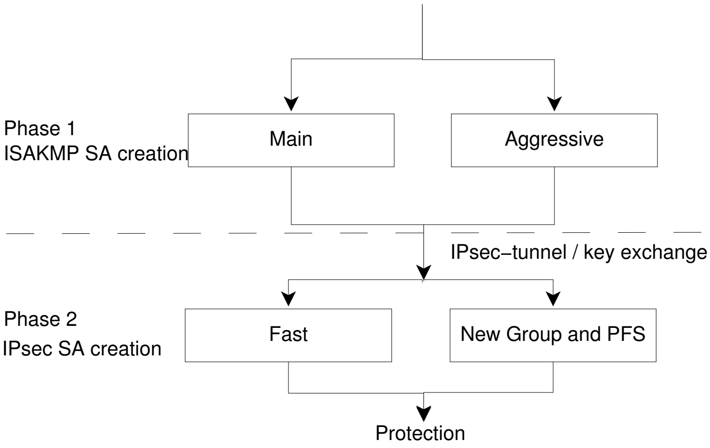


Figure 2.8: IKE phases in SA negotiations.

ISAKMP messages when used with IPsec. IKE uses ISAKMP to form a concrete protocol. The key management protocol attached to ISAKMP takes influence from Oakley as well as from SKEME, in detail IKE uses the models defined by Oakley and borrows the public key encryption and authentication from SKEME. In addition the fast re-keying and random value exchange are taken from SKEME.

IKE has four main modes, main, aggressive, fast, new group (see fig. 2.8). The main and aggressive modes are used in the first phase of IKE and the fast mode is used in the second phase of IKE. The new group mode is not attached to any phase, because it does not belong to the first or to the second phase. However, the new group mode cannot be used before the ISAKMP SA is created. The new group mode is used to negotiate the new groups for Diffie-Hellman exchange. The used encryption, and hash algorithms, authentication methods, and Diffie-Hellman groups are negotiated. Three keys are created in the first phase, one key for encryption, authentication, and for creation of new keys. These keys are not dependent on the exchanged random values. The keys are computed using the hash functions defined in the ISAKMP SA and so they are dependent on the chosen authentication mode.

The main mode is an instantiation of the identity exchange in ISAKMP. The purpose of the identity exchange is to separate the data for the identity and from the authentication data. The main mode has six messages. First two messages create the SA, the next two exchange the Diffie-Hellman public values, and the last two messages verify the received values. The first two messages give the protocol the possibility to negotiate the used encryption algorithm, hash function, and Diffie-Hellman groups. Four authentication methods have been defined: digital signature, two different public key authentication methods, and one pre-shared secret based authentication method. The two next messages give the opportunity to create a shared secret using Diffie-Hellman. The shared secret is used in the creation of the session keys. Two keys are used for the encryption and hashing. The last two messages mainly verify the received values. The aggressive mode is very similar to the main mode but comprises only of three messages. In these two modes the chosen algorithms have an effect on the message contents and on the way the session keys are created.

The Diffie-Hellman groups define the public values to be used with Diffie-Hellman. The groups are used to define primes that the protocol uses. For this reason the chosen group has an effect on the creation of the session keys and to their cryptographic strength. First of the groups are modular exponentiation groups, The second and third groups are elliptic curve groups over the field GF[2^N] (EC2N), Elliptic curve groups over the field GF[P] (ECP). The groups change depending on the parameters used to in the creation of the groups [99]. IKE offers four different groups of whose definition follows the before mentioned groups for Oakley [39].

In the second phase the exchanged messages are encrypted and authenticated. The authentication is based on the chosen hash function. The fast mode is used when creating a SA for IPsec. Every negotiation ends to the creation of two SAs, one for each direction. The purpose of the negotiation is to agree on number of IPsec parameters, exchange random values, and to identification of the traffic belonging to this SA. Random values are used in the creation of the session keys that are inherited from the shared secret that was created in the first phase using Diffie-Hellman.

IKEv2

IKEv2 offers the same basic services as version one: authentication and key generation, cryptographic algorithm negotiation, and re-keying services. IKEv2 comprises of two phases, out of which the first is further divided into two steps 1.1 and 1.2.

In the first phase 1.1 the participants create the IKE SA and compute a master secret (see Figure 2.9, steps 1 and 2). In the phase 1.1 the initiator sends the chosen SPI value (in HDR), the list of supported crytographic algorithms (SAi1), Diffie-Hellman values (KEi) and the nonce (Ni). Responder replies with the chosen cryptographic algorithm (SAr1)) based on the initiators list (SAi1), Diffie-Hellman values (KEr) and the nonce (Nr). In the phase 1.1 the responder can request a certificate for authentication from the initiator. After these messages are exchanged, each peer generates a SKEYSEED based on the exchanged Diffie-Hellman values and the

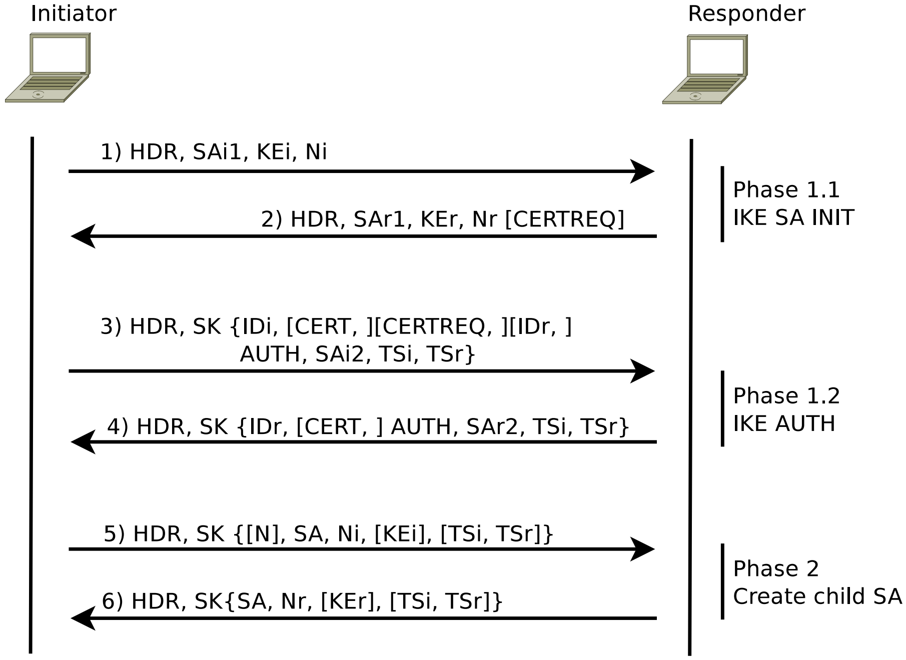

Initiator
Responder
1) HDR, SAi1, KEi, Ni
2) HDR, SAr1, KEr, Nr [CERTREQ]
Phase 1.1
IKE SA INIT
3) HDR, SK {IDi, [CERT,][CERTREQ,][IDr,]
AUTH, SAi2, TSi, TSr}
4) HDR, SK {IDr, [CERT,] AUTH, SAr2, TSi, TSr}
Phase 1.2
IKE AUTH
5) HDR, SK {[N], SA, Ni, [KEi], [TSi, TSr]}
6) HDR, SK{SA, Nr, [KEr], [TSi, TSr]}
Phase 2
Create child SA

Figure 2.9: IKE version 2 message flow.

exchanged nonces. From SKEYSEED the master secret is created, from which the IPsec SA keys are created in phase 2. The phase 1.2 is protected by the keys created in Phase 1.1 and in Phase 2 (in Figure 2.9 marked with SK...).

In the phase 1.2 the participants authenticate each other and negotiate the used IPsec algorithms (see Figure 2.9, steps 3 and 4). In the messages peers exchange values for pre-shared secrets for authentication (AUTH), the cryptographic algorithms list and response (as in phase 1.1, SAi2 and SAr2) for child SA creation, and Traffic selector information is exchanged (TSi, TSr). In phase 1.2 the initiator sends the possible certificate requested by the responder in the phase 1.1. In the phase 1.2 the initiator may request a certificate for authentication from the responder.

In the phase 2 the participants setup the IPsec AH or ESP SAs (see Figure 2.9, steps 5 and 6). In the phase 2 the initiator indicates to the responder if there is a need to negotiate a new IPsec SA (N). If there is no need, it means that this is the first IPsec SA for this IKE SA. The Diffie-Hellman values (KEi and KEr) in the phase 2 are sent only if Perfect Forward Security (PFS) is required. If PFS is required a new set of keys is computed. The nonces sent in phase 2 differ from the nonces sent in phase 1. If a new IPsec SA is created also new cryptographic algorithms are negotiated and new traffic selectors are exchanged (TSi and TSr).

Differences between the IKE versions

In the previous sections we have been discussing about IKE and about IKEv2. The new version has many differences compared to the older version, simplified behavior, support for mobility (see Section 2.3.2), NAT traversal, etc. For example IKEv2 has no aggressive mode and has only one authentication method that is based on the usage of digital signatures using public keys. The new main phase comprises of only four messages compared to the six messages needed in the previous version. In the first two messages the certificates are exchanged and in the next messages the requested certificates are sent. In the first phase the communicating parties identify each other by signing the concatenation of first messages. The used algorithms are communicated to the communicating parties by using certificates that were exchanged earlier.

IKEv2 changes also IPsec [61]. The operation principles were changed, AH [62] and ESP[63] got new versions. For example, in ESP it is not allowed to use DES-CBC algorithm [30]. Data Encryption Standard (DES) is replaced with 3DES as DES is cryptographically too weak [109]. 3DES is similar to DES but 3DES uses three keys consecutively (hence the name). The operation of IPsec changed a lot, for example, efficiency and implementation simplicity are taken in to consideration. The new version makes the form of the SA multiples more flexible and defines that all the IPsec traffic including the signaling must use the SAD.

One major change was due to the IPv6. For example, AH has now an extended sequence number and improved SPI. The extended sequence number is now twice as long as before but is backward compatible with IKE. The improvements to the SPI allow better support for multicast and unicast by improving the queries from the SAD.

ESP also has the support for the extended sequence number and for the improved SPI. ESP got also the possibility to add the padding to the end of the IP packet instead only to the ESP encapsulation. This way some of the attacks based on traffic shaping can be avoided. ESP got also support for the combined encryption and integrity algorithms. These combined algorithms are still undefined but Advanced Encryption Standard - Counter with CBC-MAC mode (AES-CCM) [47] is seen as one possibility [30]. Combined algorithms are seen as a improvement for the security, because they avoid many of the attacks described in [104].

MOBIKE

Mobility for IKEv2 is defined as an extension to the main protocol. The mobility and multihoming extension Mobility and multihoming extensions for IKEv2 (MOBIKE) allows the peers to change the addresses associated and to associate multiple addresses with the IPsec SA.

The MOBIKE message flow is depicted in Figure 2.10, the figure also illustrates the NAT traversal parameters MOBIKE borrowed from IKEv2. The steps 1 and 2 are the same as for the default IKEv2 phase 1 with the addition of the indicator for the support for MOBIKE in phase 1.2. Phase 2 is not depicted in Figure 2.10. In the step 2 the change in the attachment point is signaled from the lower layers to the MOBIKE. In the example the MN (initiator in Figure 2.10) sends a message containing the new address with a request to update the associated address of the current SA.

Upon receiving the new address the CN records the address and sends acknowledges it to the MN. Optionally depending on the policy the CN can then start the return routability test to see that the correct entity is in the new address by mirroring a protected cookie.

2.4 Host Identity Protocol

Currently discussed identifier-locator split protocols follow one of two principles: address rewriting, or mapping and encapsulating. In the address rewriting method, an IPv6 address is divided into a front and a back half. The front half of the IPv6 address represents the locator of the host, and the back half represents its identity. The ILNP [14] is a protocol that implements the address rewriting method. The deployment of address rewriting schemes requires major renumbering in the network and compulsory support for IPv6 because of the insufficient address length of IPv4. The current Internet is in a transition phase in which IPv6 connectivity cannot be guaranteed everywhere yet. This hampers the immediate deployment of protocols that rely on IPv6.

In mapping and encapsulating schemes, additional identifiers are mapped to locators, and packets are encapsulated so that locators are only used in the packet headers at the network layer, while higher layers see the identifiers in the encapsulated packets. Mapping and encapsulating approaches can be grouped into two categories: network-based and host-based encapsulation. LISP [34] is an example of a network-based approach. HIP [91] is an example of a host-based protocol. Mapping and encapsulating-based approaches have the benefit that they work on top of IPv4 as well as on top of IPv6. In addition, mapping and encapsulating schemes do not require

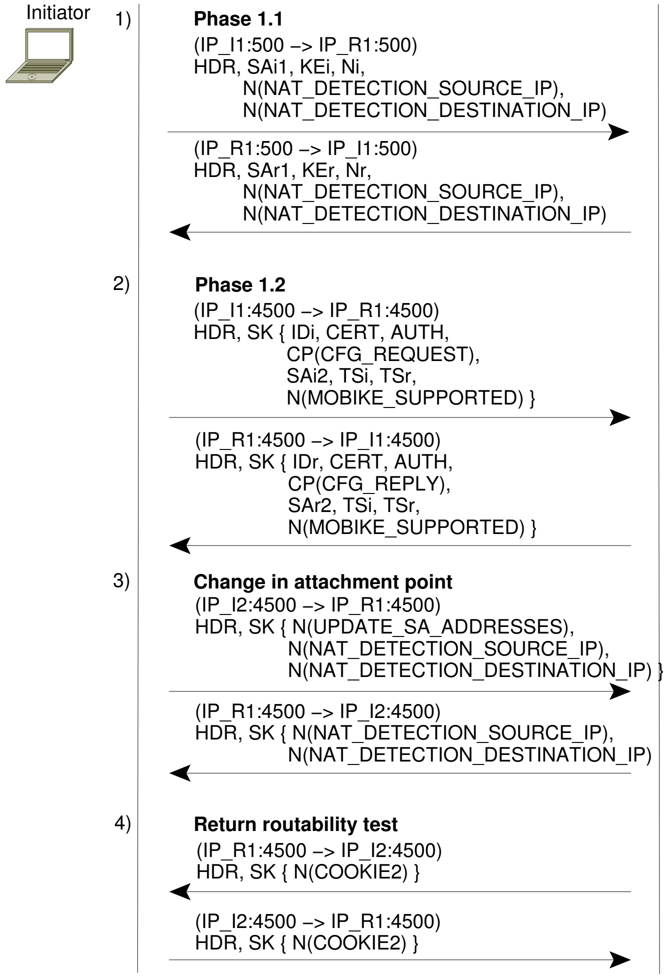

Initiator
Responder
1)
Phase 1.1
(IP_I1:500 –> IP_R1:500)
HDR, SAi1, KEi, Ni,
N(NAT_DETECTION_SOURCE_IP),
N(NAT_DETECTION_DESTINATION_IP)
(IP_R1:500 –> IP_I1:500)
HDR, SAr1, KEr, Nr,
N(NAT_DETECTION_SOURCE_IP),
N(NAT_DETECTION_DESTINATION_IP)
2)
Phase 1.2
(IP_I1:4500 –> IP_R1:4500)
HDR, SK { IDi, CERT, AUTH,
CP(CFG_REQUEST),
SAi2, TSi, TSr,
N(MOBIKE_SUPPORTED) }
(IP_R1:4500 –> IP_I1:4500)
HDR, SK { IDr, CERT, AUTH,
CP(CFG_REPLY),
SAr2, TSi, TSr,
N(MOBIKE_SUPPORTED) }
3)
Change in attachment point
(IP_I2:4500 –> IP_R1:4500)
HDR, SK { N(UPDATE_SA_ADDRESSES),
N(NAT_DETECTION_SOURCE_IP),
N(NAT_DETECTION_DESTINATION_IP) }
(IP_R1:4500 –> IP_I2:4500)
HDR, SK { N(NAT_DETECTION_SOURCE_IP),
N(NAT_DETECTION_DESTINATION_IP)
4)
Return routability test
(IP_R1:4500 –> IP_I2:4500)
HDR, SK { N(COOKIE2) }
(IP_I2:4500 –> IP_R1:4500)
HDR, SK { N(COOKIE2) }

Figure 2.10: MOBIKE message flow in the case where communicating parties both have one address and the Initiator changes its attachment point.

changes to the core routing of the Internet. In this dissertation we focus on identifier-locator split protocols that implement the mapping and encapsulating scheme, and more specifically, on host-based approaches, which use additional identifiers in and above the transport layer. These approaches show two distinct requirements that set them apart from other identifier-locator split protocols: host-based locator updates and support for flat namespaces.

HIP [91, 92, 37] introduces a flat cryptographic namespace based on public-private key pairs. An identifier in the namespace is the public key of a public-private key that the end-host creates for itself. This identifier is called Host Identifier (HI).

The protocol employs two fixed-length representations of HIs because varying length identifiers are inconvenient in networking APIs for existing legacy stacks and protocol header encodings [91]. Furthermore, using full-length public keys in packet headers would result in too much overhead and would be incompatible with unmodified (*legacy*) applications. The first representation type is Host Identity Tag (HIT). HITs can be used directly with IPv6-enabled applications because of their size and format. The HIT is generated by hashing the HI and concatenating it with an ORCHID prefix (2001:10::/28) [95]. The second representation type is Local Scope Identifier (LSI) that is the size of an IPv4 address to support legacy applications. LSIs are valid only within the local host due to high collision probability of two hosts choosing the same LSI.

Since HIP uses cryptographic keys as identifiers, host authentication and the establishment of a secure channel between HIP hosts is very simple. Moreover, HIP is designed to be extensible. A modular packet and parameter concept allows adding new functionality to HIP easily. HIP parameters are carried in HIP control packets.

2.4.1 Resolution

HIP can be deployed in two ways, non-opportunistic and opportunistic. In non-opportunistic way HIP needs infrastructure to map names to the identifiers and identifiers to routable IP-addresses (*locators*). HIP specifies DNS extensions [94] for this purpose and the eXtensible Markup Language - Remote Procedure Call (XML-RPC) interface [5] for home users who cannot publish their identifiers on DNS servers. In opportunistic mode, the Initiator of the communications makes a leap of faith and tries to initiate communication solely using a locator without knowing Responders identifier beforehand [72].

HIP introduces a new HIP Resource Record (HIP RR) for the DNS. HIP RR allows the hosts to store the HIT, the Public key HI, and the RendezVous Server (RVS) address of the host. By default the hosts query the FQDN to IP from the DNS. With HIP the host queries the HIP RR first and as a secondary query the host queries the FQDN to IP of the RVS. The RVS server acts as a single-stable-point in the Internet via which the host can be contacted (see [76]), similarly as in MIP (see Section 2.2).

HIP has support for the XML-RPC interface [5]. The XML-RPC interface provides basic features, such as *put*, *put-rm*, *rm* and *get* operations. Put operation inserts a key and a value to the storage and get retrieves the value, matching the key, from the storage. Remove operation is protected by a hash of a secret that is inserted with the value and revealed in plain text when removed. The XML-RPC uses also a time-to-live value in order to remove stale information from the storage.

The HIP DHT Resource Record (HDRR) is a HIP control-packet-like structure used to store HIP mappings with the XML-RPC interface. It can contain multiple IPv4 and/or IPv6 addresses, and it also contains the HI (the public key from which the EID was created). The HDRR is protected by a signature calculated over the HIP packet header and the included parameters. For our purposes, this format lacks only a sequence number to prevent replay attacks. However, due to the modular parameter concept, the sequence number is easy to add to the HDRR. We can even reuse the sequence number used in the basic HIP control packets, because the parameter format for the HDRR is the same as in HIP control messages.

When the application uses a HIT or a LSI to establish new outgoing communications, networking stack intercepts the packet and triggers a Base EXchange (BEX) in the networking stack to set up symmetric keys for the IPsec tunnel.

2.4.2 Base Exchange

BEX [92] is a secure Diffie-Hellman exchange that authenticates the end-hosts to each other using their public keys, and negotiates algorithms and symmetric keys for IPsec ESP [53]. The BEX is protected against replay attacks and authenticated with public-key signatures.

In HIP terminology, the client-side is referred as *Initiator* and the server-side as *Responder*. The BEX consists of four messages (fig 2.11 illustrates a base exchange). First, the Initiator starts the base exchange with an First Initiator packet (I1) packet (in Figure 2.11 step 1). Upon receiving of the I1 one the responder selects a precomputed First Responder packet (R1) containing a computational puzzle [9] (in Figure 2.11 step 2). Second, the Responder replies with its public key and Diffie-Hellman key material in an R1 (in Figure 2.11 step 3). Upon receiving of the R1, Initiator of the connection checks the validity of the packet and solves the computation puzzle in the received R1 (in Figure 2.11 steps 4 and 5). Third, the Initiator responds with an Second Initiator packet (I2) packet that contains its public key and Diffie-Hellman key material (in Figure 2.11 step 6 and 7). Fourth, the Responder concludes the BEX with an Second Responder packet (R2) packet if the solution in the received I2 control packet was

[9] The computational puzzle is based on consistent hashing of a random number I. The hardness of the solution can be varied by changing the K, i.e., the number bits that need to be verified. This is a good way to provide random delay for the BEX as the solution may be found on the first, the last, or on any try in between.

correct (in Figure 2.11 steps 8, 9, and 10). After this, the HIP state (HIP association) can transition to ESTABLISHED state on both sides. The end-hosts have agreed on SPI numbers and symmetric keys for IPsec ESP. Based on the exchanged keying material, the end-hosts create IPsec security associations (in Figure 2.11 steps 11 and 12). Finally, the applications can commence communication over the created IPsec tunnel.

After the BEX is completed successfully and both end-hosts have reached ESTABLISHED state, the two end-hosts can commence to send upper-layer traffic to each other over the encrypted ESP tunnel. From here on, state created during the BEX is called a HIP Association (HA).

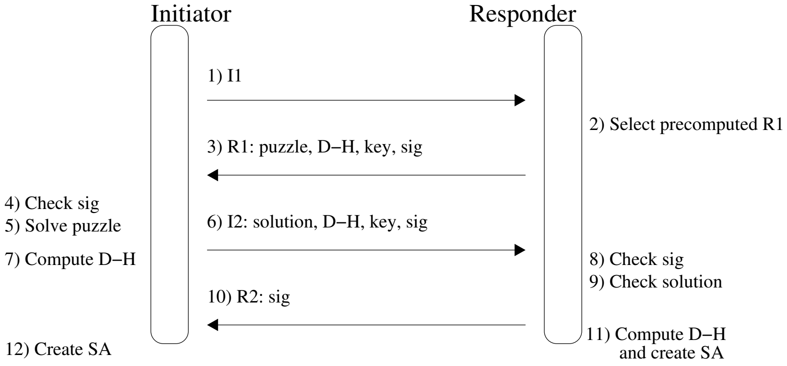


Figure 2.11: HIP Base Exchange.

After the BEX, the end-hosts lose their roles as Initiator and Responder because there is no need for such separation. Now, the end-hosts can process mobility related packets which requires different kind of state handling as discussed in the next section.

2.4.3 Mobility Management

This section summarizes HIP-based mobility as described in RFC5206 [93]. We use MIP terminology [108, 85] for denoting two communicating end-hosts, i.e., MN is a moving node and CN is an immobile node. It should be noticed that the terminology can a bit misleading because HIP architecture allows both hosts to move simultaneously [46]. We use the HIP state machine terminology [92, 93] extensively here. We also refer to routable IP-addresses as *locators*.

The core idea in HIP-based mobility is that when a mobile node detects a change in its locators, it sends its complete new set of locators to all of its correspondent nodes. A correspondent node receives the new locator set and verifies each address in the set for reachability by sending an UPDATE packet with random nonce (echo request) to the mobile node. The mobile node responds with a packet containing the same nonce (echo response). This procedure allows the correspondent node to securely verify that the mobile node is in the location it claims to be. This procedure is also referred as the *return routability test.* It should be noticed that there are no separate return routability tests for addresses used in the BEX because the BEX itself acts as an implicit return routability test.

In HIP-based mobility, a locator pair has ACTIVE, DEPRECATED and UNVERIFIED states. Figure 2.12 illustrates HIP-based mobility from the view point of locator pair state. For simplicity, retransmissions and optional negotiation of new Diffie-Helman (D-H) key material are excluded from the figure.

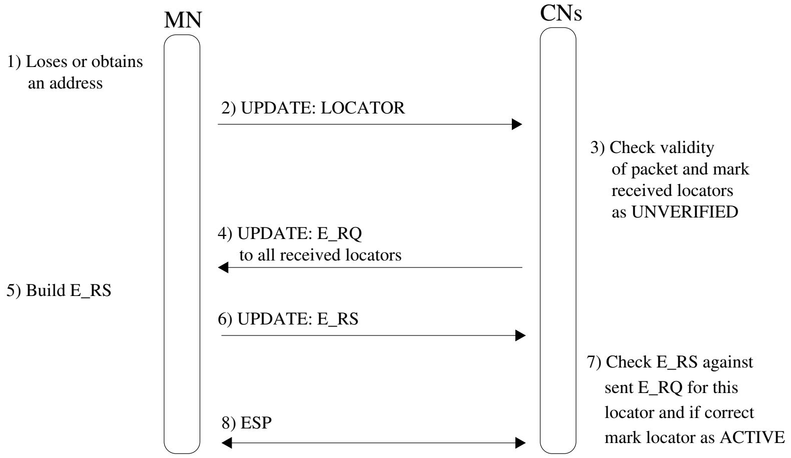

MN
CNs
1) Loses or obtains an address
2) UPDATE: LOCATOR
3) Check validity of packet and mark received locators as UNVERIFIED
4) UPDATE: E_RQ to all received locators
5) Build E_RS
6) UPDATE: E_RS
7) Check E_RS against sent E_RQ for this locator and if correct mark locator as ACTIVE
8) ESP

Figure 2.12: Return routability tests and locator state.

When the mobile node moves (in step 1, fig 2.12), its set of locators changes and it builds a LOCATOR parameter that contains the new locator set. The mobile node can exclude some locators from the LOCATOR parameter according to its local policies. For example, a mobile node might not advertise expensive links for all correspondent nodes, or it might exclude some locators for privacy reasons. The corresponding node transitions the state of locator to DEPRECATED, when the mobile node excludes the particular locator from its locator set (not shown in fig 2.12). In step 2, the mobile node sends an initial UPDATE, containing the LOCATOR parameter listing the locator set which the mobile node publishes to all of its correspondent nodes.

Now, the correspondent node receives the UPDATE packet and validates the packet by verifying packet checksum, correctness of the signature, sequence number and comparison of SPI number with existing SAs (step 3 in fig 2.12). Then, the correspondent node processes the LOCATOR parameter from the UPDATE packet.

The correspondent node marks all received locators as UNVERIFIED and deprecates existing locators excluded from the new locator set (not shown in fig 2.12). Next, the correspondent node builds an UPDATE packet containing an ECHO_REQUEST parameter (E_RQ in fig 2.12) containing a random nonce value and sends it to mobile node's locator to be tested for reachability (step 4). The correspondent node repeats this for all of the locators contained in the locator set of the mobile node.

In step 5, the mobile node receives the packet and echoes the same nonce in an ECHO_RESPONSE parameter to the correspondent node in step 6. The correspondent node receives the UPDATE packet and validates its integrity and nonce in step 7. The correspondent node transitions now the state of the peer locator to ACTIVE. If the mobile node failed to respond within a certain time, the correspondent node would deprecate the locator and remove the locator from its peer locator list.

It should be noticed that locators can be present already in the base exchange. When a locator has a so called *preferred bit* set, the sender of the locator enforces the recipient to use a specific locator for HIP-related communications for, e.g., load-balancing purposes.

2.4.4 Service Identifiers

Several HIP-related documents are concerned with the provision and discovery of services, e.g., the HIP registration extension [77] and the HIP middlebox authentication extension [41, 40]. [43] describes a HIP parameter that lets service providers communicate properties and requirements of a service to the HIP end-hosts and to on-path HIP- aware network entities. Service providers can either be other HIP end-hosts (Initiator or Responder), on-path network entities (HIP- aware middleboxes and other HIP-aware network infrastructure elements), or entities using the HIP registration extension.

The service announcement and service acknowledgement procedure is a two-way communication process that integrates into the regular HIP control channel packet exchanges.

During a base exchange or HIP update mechanism, a service provider can add a signed or an unsigned service offer to an R1, I2, R2, or UPDATE packet. In addition, the unsigned service offer can also be added to I1 packets. The service offer parameter provides general information about the service and service-specific information for the client. This information is addressed to the receiver of the HIP control packet. Each HIP packet can contain multiple service offer parameters from one or more service providers.

The client reads the service offer parameters from the incoming HIP control packet and based on local policies decides to accept or deny the service offer from the service provider. If it decides to accept the service offer and if the service requires an acknowledgement, it responds by creating a service acknowledgement parameter which it sends in the signed part of the next regular HIP control packet. If the HIP control packet containing the service offer does not require an immediate response in the next control packet, the receiver of the service offer generates an additional HIP UPDATE packet that contains the service acknowledgement parameter. If a client declines the service offer or no acknowledgement is required, it does not respond with a service acknowledgement parameter.

The service offer parameter comes in two flavors: signed and unsigned. The service parameter is covered by the signature of the HIP control packet that contains it. Therefore, it can only be added by the HIP end-host that generates the HIP control packet. The unsigned service offer is not covered by the signature in the HIP control packet, it is added by HIP-aware middleboxes or HIP end-hosts. Consequently, end-hosts can decide whether to use the signed or unsigned version of the parameter. An example in which an end-host may prefer to use the unsigned parameter is the use of pre-created R1 packets which should include a service offer parameter that depends on properties of the Initiator.

The service provider can determine whether the client acknowledges the service offer by checking the presence of a service acknowledgement parameter with a matching service identifier in the next packet. The service acknowledgement parameter contains the hash of the service offer, allowing the service provider to verify that the user has accepted the terms of service as added by the service provider in the service offer. Replying with the hash of the complete service offer parameter ensures that the client adheres to all conditions of the service offer and that the unsigned service offer parameter was delivered without modification in transit. Additionally, the service provider should verify the validity of the signature in the HIP control packet. In order to shelter against DoS attacks, end-hosts and middleboxes can utilize the puzzle mechanisms specified in [92] for end-hosts and [41] for middleboxes.

Middleboxes or end-hosts may require certificates that state that the host is entitled to perform certain actions (e.g., connect to a host, use a certain link, use a certain service) [42] (see also Section 2.4.5). The certificate parameter allows HIP hosts to transmit certificate information within HIP control packets. However, a host may possess multiple certificates and therefore it must decide which certificate to transmit.

End-hosts and middleboxes can require the client to present a certificate by adding a service offer parameter to the next packet addressed to the client. Setting the *initiator certificate* (CEI) bit set indicates that a certificate is required and should be sent on the consequent control packet in order to get service. The type of certificate can be transmitted in the Service Description (SD) field.

Likewise, an end-host or middlebox can inform a HIP host that additional authentication measures (e.g., password authentication [136]) must be performed during or after the base exchange. By setting the *required* (REQ) and *forwarding* (FOR) bits, the middlebox or end-host can express that forwarding of payload packet will not be performed until the authentication is completed. The exact type of authentication is expressed in the variable SD field.

If the end-host fails in providing sufficient credentials to the service provider it can respond with a NOTIFICATION with BLOCKED_BY_POLICY if the service provider is an end-host or a NOTIFICATION with BLOCKED_BY_POLICY_M if the service provider is a middlebox to signal the error. The policy reason for not serving or setting up an association in this case would be a missing or insufficient certificate.

2.4.5 Certificates

Digital certificates bind a piece of information to a public key by means of a digital signature, and thus, enable the holder of a private key to generate cryptographically verifiable statements. HIP has a container to transport X.509.v3 and SPKI certificates [42]. This container is an organizational parameter that can be grouped to transmit semantically grouped certificates in a systematic way.

CERT Parameter

The CERT parameter is a container for certain types of digital certificates. It does not specify any certificate semantics. However, it defines supplementary parameters that help HIP hosts to transmit semantically grouped CERT parameters in a more systematic way. The specific use of the CERT parameter for different use cases is intentionally not discussed in this document, because it is specific to a concrete use case. Hence, the use of the CERT parameter will be defined in the documents that use the CERT parameter.

The CERT parameter is covered and protected, when present, by the HIP SIGNATURE field and is a non-critical parameter.

The CERT parameter can be used in all HIP packets. However, using it in the I1 packet is not recommended, because it can increase the processing times of I1s, which can be problematic when processing storms of I1s. Each HIP control packet may contain multiple CERT parameters. These parameters may be related or unrelated. Related certificates are managed in Cert groups. A Cert group specifies a group of related CERT parameters that should be interpreted in a certain order (e.g., for expressing certificate chains). For grouping CERT parameters, the Cert group and the Cert count field must be set. Ungrouped certificates exhibit a unique Cert group field and set the Cert count to 1. CERT parameters with the same Cert group number in the group field indicate a logical grouping. The Cert count field indicates the number of CERT parameters in the group.

CERT parameters that belong to the same Cert group may be contained in multiple sequential HIP control packets. This is indicated by a higher Cert count than the amount of CERT parameters with matching Cert group fields in a HIP control packet. The CERT parameters must be placed in ascending order, within a HIP control packet, according to their Cert group field. Cert groups may only span multiple packets if the Cert group does not fit the packet. A HIP packet must not contain more than one incomplete Cert group that continues in the next HIP control packet.

The Cert ID acts as a sequence number to identify the certificates in a Cert group. The numbers in the Cert ID field must start from 1 up to Cert count.

The Cert group and Cert ID namespaces are managed locally by each host that sends CERT parameters in HIP control packets.

Table 2.1: Supported certificate formats.

Cert format	Type number
X.509.v3	1
SPKI	2
Hash and URL of X.509.v3	3
Hash and URL of SPKI	4
LDAP URL of X.509.v3	5
LDAP URL of SPKI	6
Distinguished Name of X.509.v3	7
Distinguished Name of SPKI	8

The next sections outline the use of HITs in X.509.v3 and in SPKI certificates. X.509.v3 certificates and the handling procedures are defined in [23]. The wire format for X.509.v3 is the Distinguished Encoding Rules format as defined in [59]. The SPKI, the handling procedures, and the formats are defined in [31].

Hash and URL encodings (3 and 4 2.1) are used as defined in Section 3.6 of [60]. Using hash and URL encodings results in smaller HIP control packets than by including the certificate(s), but requires the receiver to resolve the URL or check a local cache against the hash.

Lightweight Directory Access Protocol (LDAP) URL encodings (5 and 6 2.1) are used as defined in [125]. Using LDAP URL encoding results in smaller HIP control packets but requires the receiver to retrieve the certificate or check a local cache against the URL.

DN encodings (7 and 8 in 2.1) are represented by the string representation of the certificate's subject DN as defined in [57].Using the DN encoding results in smaller HIP control packets, but requires the receiver to retrieve the certificate or check a local cache against the DN.

X.509.v3 Certificate Object and Host Identities

If needed, HITs can represent an issuer, a subject, or both in X.509 v3. HITs are represented as IPv6 addresses as defined in [95]. When the HI is used to sign the certificate, the respective HIT must be placed into the Issuer Alternative Name (IAN) extension using the GeneralName form iPAddress as defined in [23]. When the certificate is issued for a HIP host, identified by a HIT and HI, the respective HIT must be placed into the Subject Alternative Name (SAN) extension using the GeneralName form iPAddress, and the full HI is presented as the subject's public key info as defined in [23] (see Figure2.13).

The following examples illustrate how HITs are presented as issuer and subject in the X.509 v3 extension alternative names.

```
Format of X509v3 extensions:
    X509v3 Issuer Alternative Name:
        IP Address:hit-of-issuer
    X509v3 Subject Alternative Name:
        IP Address:hit-of-subject

Example X509v3 extensions:
    X509v3 Issuer Alternative Name:
        IP Address:2001:14:6cf:fae7:bb79:bf78:7d64:c056
```

```
          X509v3 Subject Alternative Name:
              IP Address:2001:1c:5a14:26de:a07c:385b:de35:60e3
```

As another example, consider a managed PKI environment in which the peers have certificates that are anchored in (potentially different) managed trust chains. In this scenario, the certificates issued to HIP hosts are signed by intermediate CAs up to a root CA. In this example, the managed PKI environment is neither HIP aware, nor can it be configured to compute HITs and include them in the certificates.

In this scenario, it is recommended that the HIP peers have and use some mechanism of defining trusted root CAs for the purpose of establishing HIP communications. Furthermore it is recommended that the HIP peers have and use some mechanism of checking peer certificate validity for revocation, signature, minimum cryptographic strength, etc., up to the trusted root CA.

When HIP communications are established, the HIP hosts not only need to send their identity certificates (or pointers to their certificates), but also the chain of intermediate CAs (or pointers to the CAs up to the root CA, or to a CA that is trusted by the remote peer. This chain of certificates must be sent in a Cert group as specified in Section 2.4.5. The HIP peers validate each other's certificates and compute peer HITs based on the certificate public keys.

SPKI Cert Object and Host Identities

When using SPKI certificates to transmit information related to HIP hosts, HITs need to be enclosed within the certificates. HITs can represent an issuer, a subject, or both. In the following we define the representation of those identifiers for SPKI given as S-expressions. Note that the S-expressions are only the human-readable representation of SPKI certificates. Full HIs are presented in the public key sequences of SPKI certificates (see Figure2.14).

As an example the HIT of a host is expressed as follows:

```
Format:  (hash hit hit-of-host)
Example: (hash hit 2001:13:724d:f3c0:6ff0:33c2:15d8:5f50)
```

Revocation of Certificates

Revocation of X.509 v3 certificates is handled as defined in Section 5 in [48]. Revocation of SPKI certificates is handled as defined in Section 5 in [31].

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN=Example issuing host, DC=example, DC=com
        Validity
            Not Before: Mar 11 09:01:39 2011 GMT
            Not After : Mar 21 09:01:39 2011 GMT
        Subject: CN=Example subject host, DC=example, DC=com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c0:db:38:50:8e:63:ed:96:ea:c6:c4:ec:a3:36:
                    62:e2:28:e9:74:9c:f5:2f:cb:58:0e:52:54:60:b5:
                    fa:98:87:0d:22:ab:d8:6a:61:74:a9:ee:0b:ae:cd:
                    18:6f:05:ab:69:66:42:46:00:a2:c0:0c:3a:28:67:
                    09:cc:52:27:da:79:3e:67:d7:d8:d0:7c:f1:a1:26:
                    fa:38:8f:73:f5:b0:20:c6:f2:0b:7d:77:43:aa:c7:
                    98:91:7e:1e:04:31:0d:ca:94:55:20:c4:4f:ba:b1:
                    df:d4:61:9d:dd:b9:b5:47:94:6c:06:91:69:30:42:
                    9c:0a:8b:e3:00:ce:49:ab:e3
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Issuer Alternative Name:
                IP Address:2001:13:8d83:41c5:dc9f:38ed:e742:7281
            X509v3 Subject Alternative Name:
                IP Address:2001:1c:6e02:d3e0:9b90:8417:673e:99db
    Signature Algorithm: sha1WithRSAEncryption
        83:68:b4:38:63:a6:ae:57:68:e2:4d:73:5d:8f:11:e4:ba:30:
        a0:19:ca:86:22:e9:6b:e9:36:96:af:95:bd:e8:02:b9:72:2f:
        30:a2:62:ac:b2:fa:3d:25:c5:24:fd:8d:32:aa:01:4f:a5:8a:
        f5:06:52:56:0a:86:55:39:2b:ee:7a:7b:46:14:d7:5d:15:82:
        4d:74:06:ca:b7:8c:54:c1:6b:33:7f:77:82:d8:95:e1:05:ca:
        e2:0d:22:1d:86:fc:1c:c4:a4:cf:c6:bc:ab:ec:b8:2a:1e:4b:
        04:7e:49:9c:8f:9d:98:58:9c:63:c5:97:b5:41:94:f7:ef:93:
        57:29
```

Figure 2.13: a X.509.v3 certificate with encoded HITs.

```
(sequence
  (public_key
    (rsa-pkcs1-sha1
      (e #010001#)
      (n |yDwznOwXOw+zvQbpWoTnfWrUPLKW2NFrpXbsIcH/QBSLb
          k1RKTZhLasFwvtSHAjqh22OW8gRiQAGIqKplyrDEqSrJp
          OdIsHIQ8BQhJAyILWA1Sa6f5wAnWozDfgdXoKLNdT8ZNB
          mzluPiw4ozc78p6MHElH75Hm3yHaWxT+s83M=|
      )
      )
    )
    (cert
      (issuer
        (hash hit 2001:15:2453:698a:9aa:253a:dcb5:981e)
      )
      (subject
        (hash hit 2001:12:ccd6:4715:72a3:2ab1:77e4:4acc)
      )
      (not-before "2011-01-12_13:43:09")
      (not-after "2011-01-22_13:43:09")
    )
    (signature
      (hash sha1 |h5fC8HUMATTtKOcjYqIgeN3HCIMA|)
      |u8NTRutINI/AeeZgN6bngjvjYPtVahvY7MhGfenTpT7MCgBy
      NoZglqH5Cy2vH6LrQFYWxOMjWoYwHKimEuBKCNd4TK6hrCyAI
      CIDJAZ7OTyKXgONwDNWPOmcc3lFmsih8ezkoBseFWHqRGISIm
      MLdeaMciP4lVfxPY2AQKdMrBc=|
  )
)
```

Figure 2.14: A SPKI certificate with encoded HITs. The example has been indented for readability.

Error signaling

If the Initiator does not send the certificate that the Responder requires the Responder may take actions (e.g., blocking the connection). The Responder may signal this to the Initiator by sending a HIP NOTIFY message with NOTIFICATION parameter error type CREDENTIALS_REQUIRED.

If the verification of a certificate fails, a verifier may signal this to the provider of the certificate by sending a HIP NOTIFY message with NOTIFICATION parameter error type INVALID_CERTIFICATE.

```
NOTIFICATION PARAMETER - ERROR TYPES     Value
------------------------------------     -----


CREDENTIALS_REQUIRED                      48

The Responder is unwilling to set up an association
as the Initiator did not send the needed credentials.

INVALID_CERTIFICATE                       50

Sent in response to a failed verification of a certificate.
Notification Data contains 4 octets, in order Cert group,
Cert count, Cert ID, and Cert type of the certificate
parameter that caused the failure.
```

Security Considerations

Certificate grouping allows the certificates to be sent in multiple consecutive packets. This might allow similar attacks as IP-layer fragmentation allows, for example sending of fragments in wrong order and skipping some fragments to delay or stall packet processing by the victim in order to use resources (e.g., Central Processing Unit (CPU) or memory).

It is not recommended to use grouping or hash and URL encodings when HIP aware middleboxes are anticipated to be present on the communication path between peers because fetching remote certificates require the middlebox to buffer the packets and to request remote data. This makes these devices prone to DoS attacks. Moreover, middleboxes and responders that request remote certificates can be used as deflectors for distributed denial of service attacks.

Differences in MOBIKE and HIP mobility and multihoming

MOBIKE allows both of the peers to change their attachment points but does not support simultaneous mobility. HIP also allows both of the peers to change their attachment points and HIP is also able to use RVS service to support simultaneous mobility. MOBIKE is so better suited for client mobility in client-server models and in HIP the peers can move more freely.

In MOBIKE the peers can be multihomed but only one of the SAs may be active at a time. In HIP the multihoming behavior is the same, but research is being done to support load-balancing like features using multiple SAs at the same time [38].

In MOBIKE return routability tests are optional and done based on a policy decision. In HIP the return routability tests are mandatory.

2.5 Summary

This chapter gave the needed overview to the techniques and concepts needed for the rest of the dissertation. We started from the basic necessities and moved towards HIP.

We gave an overview on the DNS and described the current day behavior and some of the suggested systems, that may or may not, replace or supplement the DNS. From the resolution of the identifiers we moved forward by giving an overview of the current day specifications defining mobility, i.e, MIPv4 and MIPv6.

From connectivity and mobility we moved to the techniques and concepts needed to provide the security between communicating hosts with IPsec. Furthermore, we gave overview on the concepts needed to secure the mobile hosts with MOBIKE. Security overview included basics for certificates, i.e, X.509.v3 and SPKI.

From MOBIKE we move to the HIP. There are similarities between MOBIKE and HIP which are most likely caused by the fact that the protocols were developed close together in time and they shared some of the developers. However, HIP is a state of the art host identification protocol that separates the identifier from the locator and in effect decouples the transport layer form the Internet layer.

The identifier-locator separation can pose problems to the current day applications and frameworks. For this reason we now continue with the analysis of the current day open source applications and frameworks in order to see how they use the networking APIs.

Chapter 3

Statistics and Empirical Experience with Sockets API

Network applications are typically developed with frameworks that hide the low level networking details. The motivation is to allow developers to focus on application specific logic rather than networking level issues, such as name resolution, reliability, asynchronous processing and quality of service. In this chapter, we characterize statistically how open-source applications use the Sockets API and identify key application requirements based on our analysis. The analysis considers five fundamental questions: naming with end-host identifiers, name resolution, multiple end-host identifiers, multiple transport protocols and security. We discuss the significance of these findings for network application frameworks and their development. As two of our key findings, we contribute the discovery of problems with OpenSSL initialization in C-based applications and a multihoming issue with UDP in all of the analyzed four frameworks.

a

3.1 Introduction

The Sockets API is basis of all networking applications. While number of applications using it directly is large, a number of application use it indirectly through intermediate libraries or frameworks to hide all of the intricacies of the low-level Sockets API. Nevertheless, it is then the intermediaries that still have to interface with the Sockets API.

The Sockets API is important for all network applications either directly or indirectly but has been studied little. To fill this gap, we have conducted a statistical analysis on the usage of Sockets API to characterizing how today's network applications behave in Ubuntu Linux. In addition to merely characterize the trends, we investigated also certain programming pitfalls pertaining the Sockets API.

As a result, we report ten main findings and how they impact a number of relatively new sockets API extensions. To mention few examples, the poor adoption of a new DNS look up function slows down the migration path for the APIs of HIP and IPv6 source address selection. OpenSSL library is initialized incorrectly in many applications, thus causing potential security vulnerabilities. The management of the dual use of TCP and User Datagram Protocol (UDP) transports, and dual use of the two IP-address families, creates redundant complexity in applications.

To escape the unnecessary complexity of the Sockets API, some applications utilize network application frameworks. However, the frameworks are themselves based on the Sockets API and, therefore, subject to the same scrutiny as applications using the Sockets API. For this reason, it is natural to extend the analysis for frameworks.

We chose four example frameworks based on the Sockets API and analyzed them manually in the light of the Sockets API findings. Since frameworks can offer high-level abstractions and do not have to mimic the Sockets API, we organized the analysis of the frameworks top-down and along generalized dimensions of end-host naming, multiplicity of names and transports, name look up and security. As a highlight of the analysis, we discovered a persistent problem with multiplicity of names in all of the four frameworks. To be more precise, the problem was related to multihoming with UDP.

In this chapter, we describe how to solve some of the discovered issues in applications using the Sockets API. We also characterize some of the inherent limitations of the Sockets API, for example, related to complexity. As the API is very difficult to change, we suggest solutions for frameworks instead. Application utilizing the frameworks can then inherit the improvements indirectly.

3.2 Background

In this section, we first introduce the parts of the Berkey Sockets and the POSIX APIs that are essential to understand the results described in this chapter. Then, we briefly introduce four network application frameworks built on top of the two APIs.

3.2.1 The Sockets API

The Sockets API is the de facto API for network programming due to its availability for various operating systems and languages. As the API is rather low level and does not support object-oriented languages well, many networking libraries and frameworks offer additional higher-level abstractions to hide the intricacies of the Sockets API.

Unix-based systems typically provide an abstraction of all network, storage and other devices to the applications. The abstraction is realized with *descriptors* which are also sometimes referred as to *handles*. The descriptors are either file or socket descriptors. Both of them have different, specialized accessor functions even though socket descriptors can be operated with some of the file-oriented functions.

When a socket descriptor is created with the *socket()* function, the transport protocol has to be fixed for the socket. In practise, *SOCK_STREAM* constant fixes the transport protocol to TCP and *SOCK_DGRAM* constant to UDP. For IPv4-based communications, an application uses a constant called *AF_INET*, or its alias *PF_INET*, to create an IPv4-based socket. For IPv6, the application uses correspondingly *AF_INET6* or *PF_INET6*.

Name Resolution

An application can look up names from DNS by calling *gethostbyname()* function and *gethostbyaddr()* function. The former looks up the host information from the DNS by its symbolic name (forward look up) and the latter by its numeric name, i.e., IP-address (reverse look up). While both of these functions support IPv6, they are obsolete and their modern replacements are the *getnameinfo()* and *getaddrinfo()* functions.

Delivery of Application Data

A client-side application can start sending data immediately after creation of the socket; however the application typically calls the *connect()* function to associate the socket with a certain destination address and port. The *connect()* call also triggers the TCP handshake for sockets of *SOCK_STREAM* type. Then, the networking stack automatically associates a source address and port if the application did not choose them explicitly with the *bind()* function. Finally, a *close()* call terminates the socket gracefully and, when the type of the socket is *SOCK_STREAM*, the call also initiates the shutdown procedure for TCP.

Before a server-oriented application can receive incoming datagrams, it has to call a few functions. Minimally with UDP, the application has to define the port number and IP-address to listen to by using *bind()*. Typically, TCP-based services supporting multiple simultaneous clients prepare the socket with a call to the *listen()* function for the following *accept()* call. By default, the *accept()* call blocks the application until a TCP connection arrives. The function "peels off" a new socket descriptor from existing one that can then be used to handle the particular connection.

A constant *INADDR_ANY* is used with *bind()* to listen for incoming datagrams on all network interfaces and addresses of the local host. This wildcard address is typically employed in server-side applications.

An application can deliver and retrieve data from the transport layer in multiple alternative ways. For instance, the *write()* and *read()* functions are file-oriented functions but can also be used with socket descriptors to send and receive data. For these two file-oriented functions, the Sockets API defines its own specialized functions.

For datagram-oriented networking with UDP, the *sendto()* and the *recvfrom()* functions can be used. Complementary functions *sendmsg()* and *recvmsg()* offer more advanced interfaces for applications [128]. They operate on the scatter arrays (multiple non-consecutive I/O buffers instead of just one) and support also so called ancillary data that refers to meta-data and information related to network packet headers.

While the primary purpose of the socket calls is to send and receive data, they also implement access control. The *bind()* and *connect()* limit ingress (but not egress) network access to the socket by setting the allowed local and remote destination end point. Similarly, the *accept()* call effectively constrains remote access to the newly created socket by allowing communications only with the particular client. Functions *send()* and *recv()* are typically used for connection-oriented networking, but can also be used with UDP to limit remote access.

Customizing Networking Stack

The Sockets API provides certain default settings for applications to interact with the transport layer. The settings can be altered in multiple different ways.

With "raw" sockets, a process can basically create its own transport-layer protocol or modify the network-level headers. A privileged process creates a raw socket with socket type *SOCK_RAW*.

A more constrained way to alter the default behavior of the networking stack is to set socket options with *setsockopt()*. As an example of the options, the *SO_REUSEADDR* socket option can be used to disable the default "grace period" of a locally reserved transport-layer port. By default, consecutive calls to *bind()* with the same port fail until the grace period has passed. Especially during the development of a networking service, this grace period is usually disabled for convenience because the developed service may have to be restarted quite often for testing purposes.

Another way to influence the stack is to configure the underlying low-level networking devices. Two alternative APIs exist for this purpose. First, *ioctl()* is a function that allows, e.g., a socket to be set to non-blocking mode, to query addresses of network interfaces, and to manipulate ARP tables and routes. Second, the *fcntl()* function offers a more portable, albeit more constrained, set of features when compared to *ioctl()*.

3.2.2 Sockets API Extensions

Basic Socket Interface Extensions for IPv6 [35] defines additional data structures and constants, including *AF_INET* and *sockaddr_in6*. It also defines the new resolver functions, *getnameinfo()* and *getaddrinfo()*, as the old ones, *gethostbyname()* and *gethostbyaddr()*, by now are inadequate. The older ones are not thread safe and offer too little control over the resolved addresses. The specification also defines IPv6-mapped IPv4 addresses to improve IPv6 interoperability.

The Advanced Sockets API for IPv6 [128] defines IPv6 extensions for, e.g., diagnostic and routing software. The specification also introduces ancillary options for the *sendmsg()* and *recvmsg()* interface.

An IPv6 application can typically face a choice of multiple source and destination IPv6 pairs to choose from. Picking a pair may not be a simple task and some of the pairs may not even result in a working connectivity. IPv6 Socket API for Source Address Selection [97] defines extensions that restrict the type of the resulting address to, for instance, public or temporary IPv6 addresses. The extensions include both new socket options and new flags for the *getaddrinfo()* resolver. The extensions mainly affect client-side connectivity but can affect also at the server side when UDP is being used.

The Stream Control Transmission Protocol (SCTP) [130] implements a similar set of services to TCP and UDP. In a nutshell, SCTP offers a reliable, congestion-aware, message-oriented, in-sequence transport protocol. The minimum requirement to enable SCTP in an existing application is to change the protocol type in *socket()* call to SCTP. However, the application can only fully harness the benefits of the protocol by utilizing the *sendmsg()* and *recvmsg()* interface. Also, the protocol supports sharing of a single socket descriptor for multiple simultaneous communication partners; this requires some additional logic in the application.

The Datagram Congestion Control Protocol (DCCP) is similar to TCP but does not guarantee in-order delivery. An application can use it - with minor changes - by using *SOCK_DCCP* constant when a socket is created.

In this chapter, we use a simplified term "multihoming" to describe hosts with multiple IP-addresses typically introduced by multiple network interfaces. Multihoming is becoming interesting because most of the modern handhelds are equipped with e.g. 3G and WLAN interfaces. Multihoming is supported by SCTP, HIP [90] and Site Multihoming by IPv6 Intermediation (SHIM6) [96] both solve multihoming related networking issues. In addition to end-host multihoming support, HIP offers support for end-host mobility, IPv4 networks and applications and NAT traversal. By contrast, SHIM6 is mainly a multihoming solution. From the API perspective, SHIM6 offers backwards compatible identifiers for IPv6 - in the sense that they are routable at the network layer - where as the identifiers in HIP are non-routable. HIP has its own optional APIs for HIP-aware applications [70] but both protocols share the same optional multihoming APIs [69].

Name-based Sockets are a work-in-progress at the Internet Engineering Task Force (IETF) standardization forum. While the details of the specification [135] are rather immature and the specification still lacks official consent of the IETF, the general idea is to provide extensions to the Sockets API in order to replace IP-addresses with DNS-based names. In this way, the responsibility for the management of IP-addresses is pushed down in the stack, away from the application layer.

3.2.3 NAT Traversal

Private address realms [127] were essentially introduced by NATs but also Virtual Private Networks (VPNs) and other tunneling solutions can make use of private addresses. Originally, the concept of virtual address spaces was created to alleviate the depletion of the IPv4 address space, perhaps, because it appeared that most client hosts did not need publicly-reachable addresses. Consequently, NATs also offer some security as a side effect to the client side because it discards new incoming data flows.

Private address realms have introduced multiple problems to applications. First, private addresses, as presented by most of the deployed NAT devices, are valid only in the context of the particular network. Thus, connections initiated by the client side are successfully translated by the NAT but connections initiated by the server side typically fail. This is especially problematic for P2P applications, but constrains also the design of client-server protocols. Second, most of the deployed NAT devices support a very narrow number of protocols over IP (TCP, UDP, ICMP). Third, only IPv4 is usually supported by the existing NAT devices. Fourth, networks comprising of private addresses overlap with each other. This is problematic for applications and users because a single IP-address can map to different services or devices depending on the network. Fifth, many of the solutions [1, 142, 118, 50] to work around NATs introduce additional complexity to the application developers which may also be visible to the user.

To work around NATs, Teredo [50] offers NAT traversal solution based on a transparent tunnel to the applications. The protocol tries to penetrate through NAT boxes to establish a direct end-to-end tunnel but can resort to triangular routing through a proxy in the case of an unsuccessful penetration.

3.2.4 Transport Layer Security

Transport Layer Security (TLS) [132] is a cryptographic protocol that can be used to protect communications above the transport layer. TLS, and its predecessor Secure Socket Layer (SSL), is the most common way to protect TCP-based communications over the Internet.

In order to use SSL or TLS, the C/C++ application is usually linked to a library implementing the protocol(s) such as OpenSSL or GNU's Not Unix! (GNU) TLS. The application then calls the APIs of the TLS/SSL-library instead of using the APIs of the Sockets API. The functions of the library are wrappers around the Sockets API, and are responsible for securing the data inside the TCP stream.

3.2.5 Network Frameworks

The Sockets API can be cumbersome and too error-prone to be programmed directly. It is also very flat by its nature because it was not designed to accommodate object-oriented languages. For these reasons, a number of libraries and frameworks have been built to hide the details of the Sockets API and to introduce object-oriented interfaces. The Adaptive Communication (ACE) [122] is one such framework.

ACE simplifies the development of networking applications because it offers high-level networking APIs based on software patterns observed in well-written software. Among other things, ACE includes network patterns related to connection establishment and service initialization in addition to facilitating concurrent software and distributed communication services. It supports asynchronous communications by inversion of control, i.e., the framework takes over the control of the program flow and calls registered functions of the application when needed.

Boost::Asio is a open source C++ library that offers high-level networking API to simplify development of networking applications. Boost::Asio aims to be portable, scalable, and efficient but most of all it provides a starting point for further abstraction. Several Boost C++ libraries have already been included into the C++ Technical Report 1 and into C++11. In 2006 a networking proposal based on Asio was submitted for possible inclusion into the upcoming Technical Report 2.

The Java provides an object-oriented framework for the creation and use of sockets. *Java.net* package (called Java.net from here on) supports TCP (*Socket* class) and UDP (*Datagram* class). These classes define how to communicate over an IP network.

Twisted is a modular, high-level networking framework for python. It is based on inversion of control and asynchronous messaging. Twisted has built-in support for multiple application-layer protocols, including Internet Relay Chat (IRC), Secure SHell (SSH) and HTTP. Twisted focus on service-level functionality supports writing of adaptible functionality that can be run on top of several of application-layer protocols.

3.3 Materials and Methods

We collected information related to the use of Sockets API usage in applications. In this chapter, we refer to this information as *indicators*. An indicator refers to a constant, structure or function of the C language. We analyzed the source code for indicators in a static way (based on keywords) rather than dynamically [1]. The collected set of indicators was limited to networking-related keywords obtained from the keyword indexes of two books [129, 113].

We gathered the material for our analysis from all of the released LTS releases of Ubuntu: Dapper Drake 6.06, Hardy Heron 8.04, Lucid Lynx 10.04. Table 3.1 summarizes the number of software packages gathered per release. It should be noted that the "patched" row expresses how many applications were patched by Ubuntu.

We used sections "main", "multiverse", "universe" and "security" of the Ubuntu source code repository. The material was gathered on Monday 7th of March 2011 and consisted of open-source software written using the C language. Since our study was confined to networking applications, we selected only software in Ubuntu Linux in the categories of "net", "news", "comm", "mail", and "web" (in Lucid, the last category was renamed "httpd").

	Dapper	Hardy	Lucid
Total	1,355	1,472	1,147
Patched	1,222	1,360	979
C	**721**	**756**	**710**
C++	57	77	88
Python	126	148	98
Ruby	19	27	13
Java	9	10	8
Other	423	454	232

Table 3.1: Number of packages per release version.

We did not limit or favor the set of applications e.g. based on any popularity metrics [2]. It was in our goals to find general trends in network applications, so we analyzed all network applications in the scope. There are also different definitions on what is popular. Choosing the best one is difficult and leads to biased results in any case.

In our study, we concentrated on the POSIX networking APIs and Berkeley Sockets API because they form the de facto, low-level API for all

[1] Authors believe that a more dynamic or structural analysis would not have revealed any important information on the particular issues that were investigated

[2] We performed an outlier analysis in which we compared the whole set of applications to the most popular applications (100 or more installations in Ubuntu popularity contents). We discovered that the statistical "footprint" of the popular applications is different from the whole. However, the details are omitted because this contradicted our goals and also due to space efficiency reasons.

networking applications. Also, we extended the API analysis to OpenSSL to study the use of security as well. All of these three APIs have bindings for high-level languages, such as Java and Python, and can be indirectly used from network application frameworks and libraries. As the names of the bindings used in other languages differs from those used in C language, we excluded other languages from this study.

From the data gathered, we calculated sums and means for the occurrences of each indicator. We also calculated a separate reference number. This latter was formed by introducing a binary value to denote whether a software package used a particular indicator (1) or not (0), independent of the number of occurrences. The reference number for a specific indicator was collected from all software packages, and these reference numbers were then summed and divided by the number of packages to obtain a *reference ratio*. In other words, the reference ratio describes the extent of an API indicator with one normalized score.

The reference ratio indicates capability rather than 100% guarantee that the application will use the specific indicator for all its runs. When compared with the total occurrences of an indicator, the "flattened" reference ratio shows certain advantages. For instance, it can better describe whether a certain indicator is completely missing from a number of applications. Taking a concrete example, let us compare memory allocations and deallocations. The source code of an application can be organized in such a way that all memory deallocations occur in a wrapper function so that the application has many memory allocations but only a single direct deallocation. Thus it would appear that the application is misbehaving if only the total number of allocations are compared with total number of deallocations. This does not occur with the flattened reference ratios, albeit it gives a very coarse-grained metric.

In our results, we show also reference ratios of combined indicators that were calculated by taking an union or intersection of indicators, depending on the use case. With combined indicators, we used tightly coupled indicators that make sense in the context of each other.

3.4 Results and Analysis

In this section, we show the most relevant statistical results. We focus on the findings where there is room for improvement or that are relevant to the presented Sockets API extensions. Then, we highlight the most significant patterns or key improvements for the networking applications. Finally, we derive a set of more generic requirements from the key improvements and see if they are met in four different network application frameworks built on top of the Sockets or POSIX APIs.

3.4.1 Core Sockets API

In this section, we characterize how applications use the basic or "core" Sockets API. Similarly as in Section 3.2, the topics are organized into discussion on IPv6, DNS, transport protocols and customization of the networking stack. In the last section we describe a multihoming issue related to UDP.

In the described results, the reference ratios of functions of indicators are usually shown inside brackets. All numeric values are from Ubuntu Lucid unless otherwise mentioned. Figures 3.1, 3.2, and 3.3 illustrate the 20 most frequent indicators respectively for functions, structures, and constants. The following sections analyze the most interesting cases in more detail.

Circa 30 percent of the software did not explicitly reference any critical networking functions (*socket()*, *socketpair()*, *bind()*, *connect()*, *recv()* or *send()*) but references other related sockets functions. Some of such software appeared to call Sockets API functions indirectly through wrapper functions through another library, or consisted of browser plugins or Graphical User Interface (GUI) software which did not directly control the Berkeley Sockets but rather processed application data. Also, despite all of our attempts to incline the software selection towards networking applications, some packages contained auxiliary software, such as configuration file generators and converters.

A number of networking-related functions are deprecated, including and *inet_aton()*, *inet_ntoa()*, *gethostbyname()*, *gethostbyaddr()*, *getservbyname()* and *getservbyport()*. The proportion of applications referencing any of these deprecated functions is quite high in Ubuntu Lucid (58.9%). The proportion of thread-safe invocations of the *gethostbyname()* in multi-threading applications was low (15.0%) [3].

IPv6

According to the usage of AF and PF constants 39.3% were IPv4-only applications, 0.3% IPv6-only, 26.9% hybrid and 33.5% did not reference either of the constants. To recap, while the absolute use of IPv6 was not high, the relative proportion of hybrid applications supporting both protocols was quite high.

[3] An application can delegate all DNS requests to a single thread but the effect of this is negated by the usage of reference ratio as explained in Section 3.3

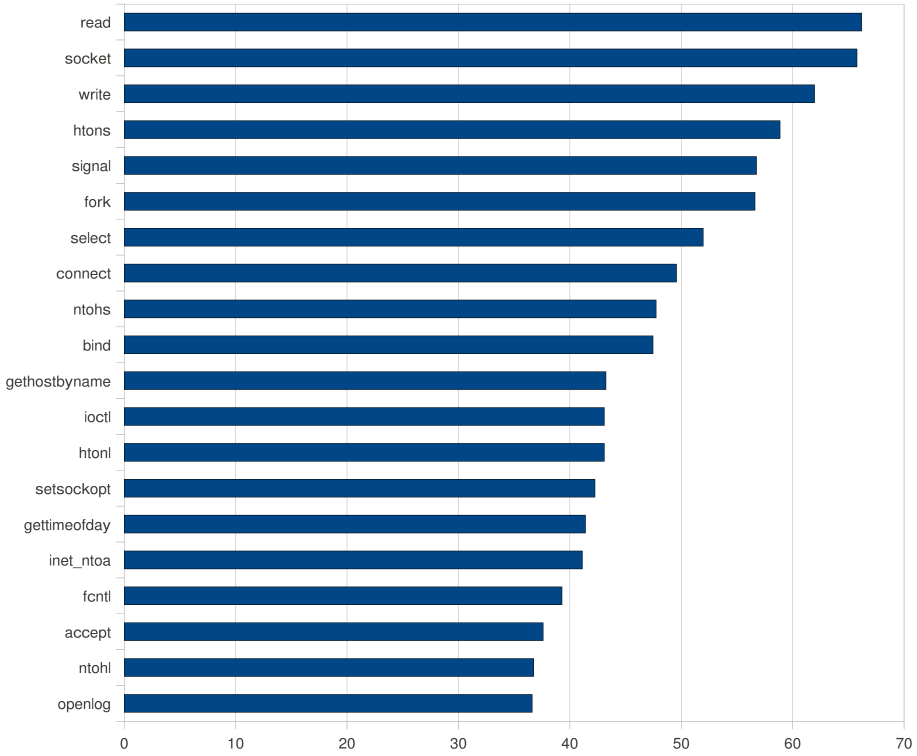


Figure 3.1: The most frequent reference ratios of functions in Ubuntu Lucid Lynx

Name Resolution

The obsolete DNS name-look-up functions were referenced more than their modern replacements. The obsolete forward look-up function *gethostbyname()* was referenced roughly twice more than its modern replacement *getaddrinfo()*. Two possible explanations for this are either that the developers have for some reason preferred the obsolete functions, or they have neglected to modernize their software.

The reference ratio of the obsolete *gethostbyname()* was declining slowly (-2.2%) and *getaddrinfo()* slowly inclining (4.4%) on the average between different Ubuntu LTS releases.

Packet Transport

Connection and datagram-oriented APIs were roughly as popular. Based on the usage of *SOCK_STREAM* and *SOCK_DGRAM* constants, we accounted for 25.1% TCP-only and 11.0% UDP-only applications. Hybrid applica-

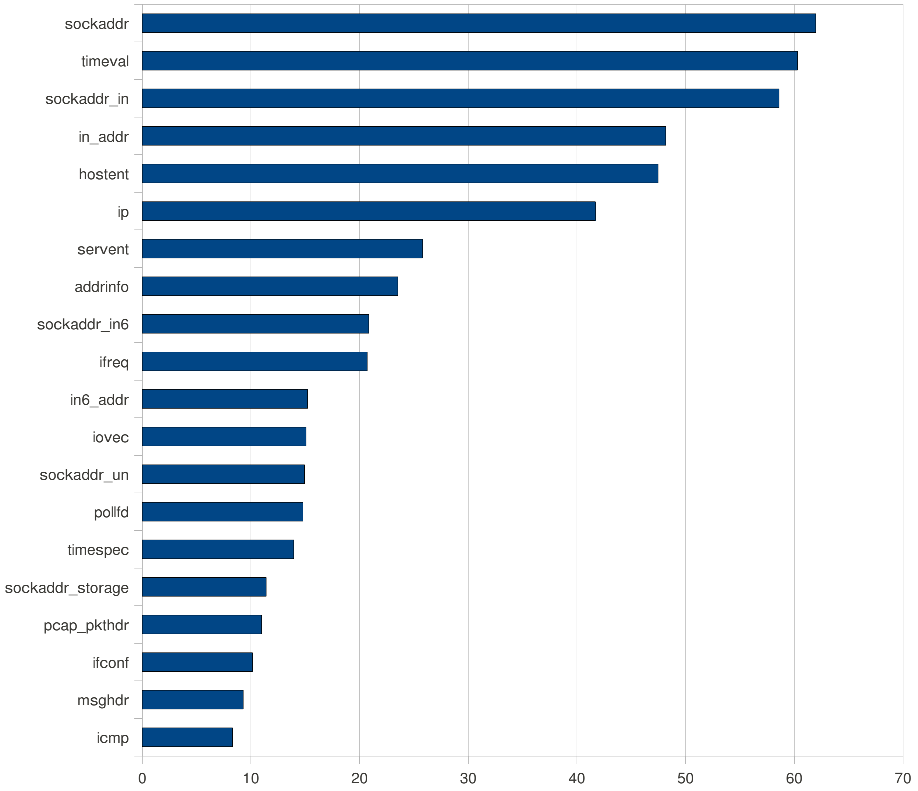


Figure 3.2: The most frequent reference ratios of structures in Ubuntu Lucid Lynx

tions supporting both protocols accounted for 26.3% - which leaves 37.6% of the applications that used neither of the constants. By combining the hybrids with TCP-only applications, the proportion of applications supporting TCP is 51.4% and, correspondingly, 37.3% for UDP. It should not be forgotten that typically all network applications implicitly access DNS over UDP by default.

Timing

While none of the applications reference the *aio_read()* and related functions, a popular way to implement this is through the popular *select()* function (52.0%). This signifies that a majority of the software remains interactive instead of blocking for long time periods. The use of an alternative to avoid blocking, constant *O_NONBLOCK* (34.2%), was not as popular as with the *select()* function. Counting together all packages that were referencing *select()*, *poll()*, *pollfd*, *pselect()* or *O_NONBLOCK* indicators, the number of

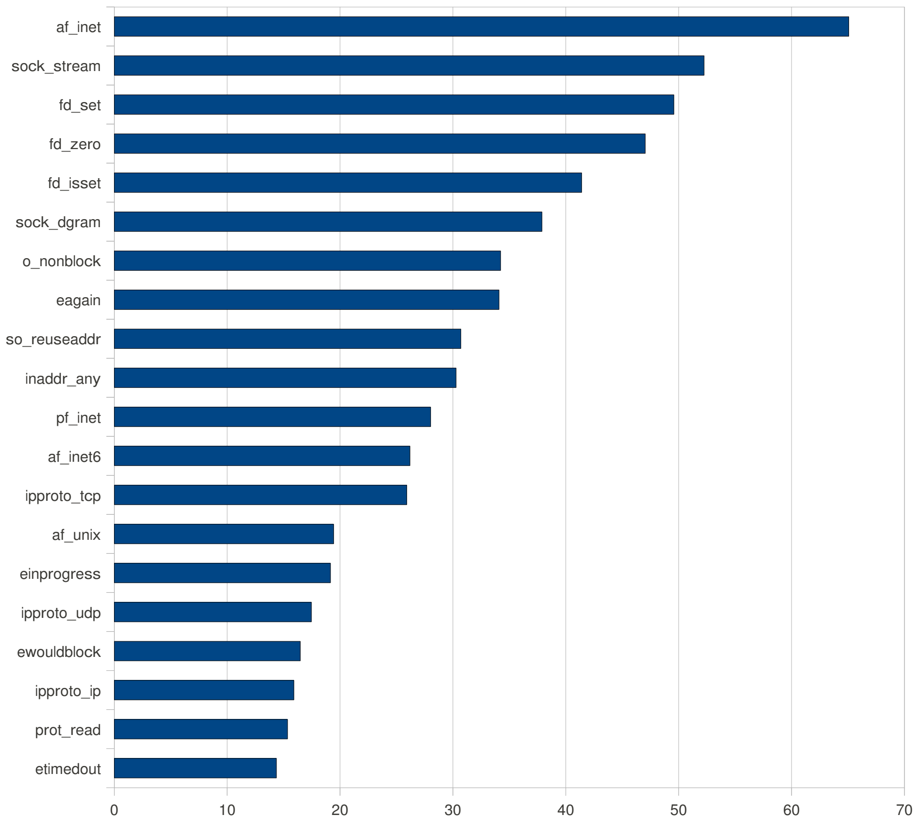


Figure 3.3: The most frequent reference ratios of constants in Ubuntu Lucid Lynx

non-blocking applications was 61.2%.

We observed that 64.6% of the network applications were operating in microsecond precision and 25.3% in nanosecond. The figure for microseconds was obtained by taking all applications referencing *select()* or *timeval* indicators. For nanoseconds, we counted all applications referencing *pselect()*, *poll()*, *timespec*, *clock_settime()* or *clock_gettime()* indicators. A scan through the nanosecond applications revealed that the majority of them could be categorized as server-side or diagnostics software.

Performance

Parallel computation was pervasive as indicated by the popularity of the POSIX *fork()* function (56.6%). Light-weight threading with *pthread_create()* function was not as popular (16.1%) as forking. One way to explain the differences in popularity is that forking isolates the processes better than threading. While the isolation includes a compromise in performance, it provides fault-tolerance against programming errors as the whole application is not doomed to fail, just a single process. Also, separate processes may be easier to comprehend and debug (e.g., less race condition with locking).

It has been suggested in the literature [117] that sockets introduce unnecessary overhead for the sending and retrieving of network data. The allegation is that the Sockets APIs should provide more direct access to the datagram (e.g., up to the network card) to avoid the overhead of copying of the data between different buffers (so called zero-copy scheme). While such functionality may be justified especially at the server side in fast networks, our statistical data does not support this demand as only 11.8% of the software were using similar functionality in the form of the POSIX *mmap()* call.

During our experience in engineering of a networking software [106], we noticed that *MSG_PEEK* (5.8%) flag has a relatively large and undocumented performance penalty of a roughly 25 ms per received datagram on Linux. We believe that the overhead cannot be explained merely by context switching. This operation allows the application to "preview" incoming data from a socket without the networking stack discarding the data for further read operations. In the implementation, we used the peek operation to preview the packet header of an incoming HIP datagram to figure out how large payload was supposed to arrive. However, we discontinued using the peek operation due to its overhead and instead just read maximum size datagrams. This is perfectly acceptable as socket read operations are allowed to return smaller amounts of data than requested.

Customizing Networking Stack

While the Sockets API provides transport-layer abstractions with certain system-level defaults, many applications preferred to customize the networking stack or to override some of the parameters. The combined reference ratio of *SOCK_RAW*, *setsockopt()*, *pcap_pkthdr* and *ipq_create_handle()* indicators was 51.4%. In other words, the default abstraction or settings of the sockets API are not sufficient for the majority of the applications.

It is worth mentioning that we conducted a brute-force search to find frequently occurring socket options sets. As a result, we did not find any recurring sets but merely individual socket options that were popular.

Multihoming and UDP

In this section, we discuss a practical issue related to UDP-based multihoming, but one which can be fixed in most applications by the correct use of *SO_BINDTODEVICE* (2.3%) socket option. The issue affects UDP-based applications accepting incoming connections from multiple interfaces or addresses.

On Linux, we have reason to believe that many UDP-based applications may not handle multihoming properly for initial connections. The multihoming problem for UDP manifests itself only when a client-side application uses a server address which does not correspond to the address used in the default route for the server. The root of the problem lies in egress datagram processing at the server side. The actual problem occurs when the client sends a "request" message to the server and the server does not send a "response" using the exactly same address pair that was used for the request. Instead, this sloppy server implementation responds to the client without specifying the source address, and the networking stack invariably chooses always the wrong source address - meaning that the client drops the response as it appears to be arriving from a previously unknown IP-address.

A straightforward fix is to modify the server-side processing of the software to respect original IP-address, and thus prevent the network stack from routing the packet incorrectly. In other words, when the server-side application receives a request, it should remember the local address of the received datagram and use it explicitly for sending the response.

Explicit source addressing can be realized by using the modern *sendmsg()* interface. However, a poorly documented alternative that can also be used is the *sendto()* function using a socket option called *SO_BINDTODEVICE*. This function is necessary because *bind()* can be used to specify the local address only for the ingress direction and not the egress.

We discovered this problem by accident with the iperf, nc and nc6 software. We have offered fixes to maintainers of these three pieces of software. Nevertheless, the impact of the problem may be larger as a third of the software in our statistics supports UDP explicitly. To be more precise, the lack of *SO_BINDTODEVICE* usage affects 45.7% (as an upper bound) of the UDP-capable software, which in practice accounts for a total of 121 applications. This figure was calculated by finding the intersection of all applications not using *sendmsg()* and *SO_BINDTODEVICE*, albeit still using *sendto()* and *SOCK_DGRAM*. We then divided this by the number of applications using *SOCK_DGRAM*.

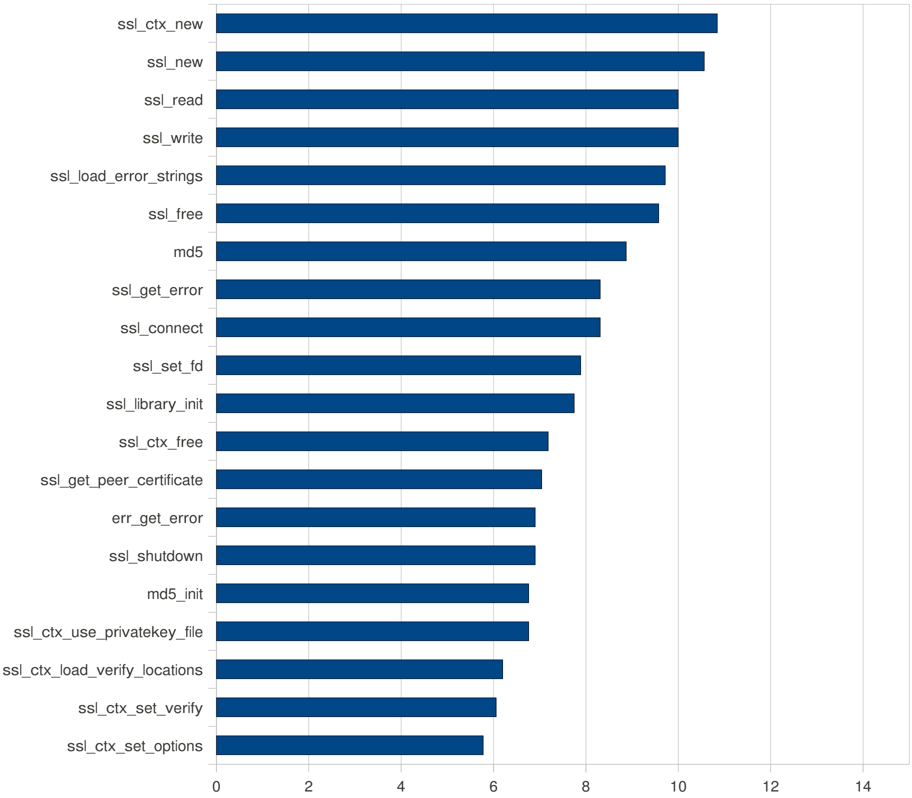


Figure 3.4: The most frequent reference ratios of SSL indicators in Ubuntu Lucid Lynx

3.4.2 Sockets API Extensions

In this section, we show and analyze statistics on SSL. Then we show how widely various other Sockets API extensions are adopted and explain how the adoption of the core Sockets API indicators impacts also the adoption of the extensions.

Security: SSL/TLS Extensions

Figure 3.1 illustrates some of the most frequent SSL indicators and the following sections analyze the most interesting cases in more detail.

Roughly 10.9% of the software in the data set used OpenSSL and 2.1% GNU TLS. In this section, we focus on OpenSSL, which is more popular. The applications using OpenSSL consisted of both client and server software. The majority of the applications using OpenSSL (54%) consisted of email, news and messaging software. The minority included network security and diagnostic, proxy, gateway, http and ftp server, printing, database and browser software.

Unless separately mentioned, we will, for convenience, use the term SSL to refer both TLS and SSL protocols; we will favor the OpenSSL naming scheme of the APIs. In this section, we only present reference ratios relative to the applications using OpenSSL because this is more meaningful from the viewpoint of the analysis. In other words, the percentages account only the OpenSSL-capable applications (77) and not the whole set of applications.

The reference ratios of SSL options remained roughly the same throughout the various Ubuntu releases. The use of SSL options in Ubuntu Lucid is illustrated in fig 3.5.

The use of *SSL_get_verify_result()* function (37.7%) indicates that a substantial proportion of SSL-capable software has interest in obtaining the results of the certificate verification. The *SSL_get_peer_certificate()* function (64.9%) is used to obtain the certificate sent by the peer.

The use of the *SSL_CTX_use_privatekey_file()* function (62.3%) implies that a majority of the software is capable of using private keys stored in files. A third ((27.3%) of the applications uses the *SSL_get_current_cipher()* function to request information about the cipher used for the current session.

The *SSL_accept()* function (41.6%) is an SSL equivalent for *accept()*. The reference ratio of *SSL_connect()* function (76.6%), an SSL equivalent for *connect()*, is higher than for *ssl_accept()* (41.6%). This implies that the data set includes more client-based applications than server-based. Furthermore, we observed that *SSL_shutdown()* (63.6%) is referenced in only about half of the software that also reference *SSL_connect()*, indicating that clients leave dangling connections with servers.

We noticed that only 71.4% of the SSL-capable software initialized the OpenSSL library correctly. The correct procedure for a typical SSL application is that it should initialize the library with *SSL_library_init()* function (71.4%) and provide readable error strings with *SSL_load_error_strings()* function (89.6%) before any SSL action takes place. However, 10.4% of the SSL-capable software fails to provide adequate error handling.

Only 58.4% of the SSL-capable applications seed the Pseudo Random Number Generator (PRNG) with *RAND_load_file()* (24.7%), *RAND_add()* (6.5%) or *RAND_seed()* (37.7%). This is surprising because incorrect seeding of the PRNG is considered a common security pitfall.

Roughly half of the SSL-capable software set the context options for SSL with *SSL_CTX_set_options* (53.3%); this modifies the default behavior of the SSL implementation. The option *SSL_OP_ALL* (37.7%) enables all

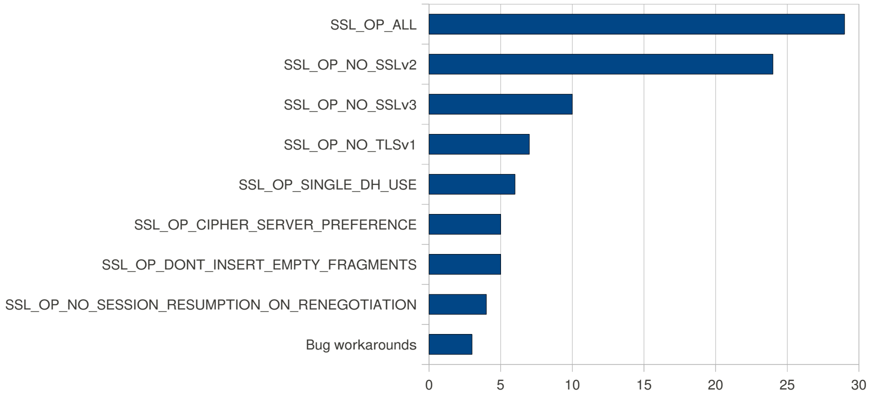


Figure 3.5: The number of occurrences of the most common SSL options

bug fixes.

SSL_OP_NO_SSLV2 option (31.2%) turns off Secure Socket Layer version 2 (SSLv2) and *SSL_OP_NO_SSLV3* (13.0%) turns off the support for Secure Socket Layer version 3 (SSLv3). The two options were usually combined so that the application would just use Transport Layer Security version 1 (TLSv1).

SSL_OP_SINGLE_DH_USE (7.8%) forces the implementation to re-compute the private part of the D-H key exchange for each new connection. With the exception of low-performance CPUs, it is usually recommended to turn on this option since it improves security.

The option *SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS* (6.5%) disables protection against an attack on the block-chaining ciphers. The countermeasure is disabled because some of the SSLv3 and TLSv1 implementations are unable to handle it properly.

37.7% of the SSL-capable software prefers to use only TLSv1 (*TLSv1_client_method()*) and 20.1% of the SSL-capable software prefers to fall back from TLSv1 to SSLv3 when the server does not support TLSv1. However, the use of *SSL_OP_NO_TLSV1* option indicates that 7% of the software is able to turn off TLSv1 support completely. *SSL_OP_CIPHER_SERVER_PREFERENCE* is used to indicate that the server's preference in the choosing of the cipher takes precedence. *SSL_OP_NO_SESSION_RESUMPTION_RENEGOTIATION* indicates the need for increased security as session resumption is disallowed and a full handshake is always required. The remaining options are various workarounds for bugs.

As a summary of SSL results, it appears that SSL-capable applications are interested of the details of the security configuration. However, some applications initialize OpenSSL incorrectly and trade security for backwards compatibility.

IPv6-Related Extensions

During the long transition to IPv6, we believe that the simultaneous coexistence of IPv4 and IPv6 still represents problems for application developers. For example, IPv6 connectivity is still not guaranteed to work everywhere. At the client side, this first appears as a problem with DNS look-ups if they are operating on top of IPv6. Therefore, some applications may try to look up simultaneously over IPv4 and IPv6 [143]. After this, the application may even try to call *connect()* simultaneously over IPv4 and IPv6. While these approaches can decrease the initial latency, they also generate some additional traffic to the Internet and certainly complicate networking logic in the application.

At the server side, the applications also have to maintain two sockets: one for IPv4 and another for IPv6. We believe this unnecessarily complicates the network processing logic of applications and should be abstracted away by network-application frameworks.

A more immediate solution to the concerns regarding address duplication is RFC4291 [45], which describes IPv4-mapped IPv6 addresses. The idea is to embed IPv4 addresses in IPv6 address structures and thus provide a unified data structure format for storing addresses in the application.

Mapped addresses can be employed either manually or by the use of *AI_V4MAPPED* flag for the *getaddrinfo()* resolver. However, the application first has to explicitly enable the *IPV6_V6ONLY* socket option (0.1%) before the networking stack will allow the IPv6-based socket to be used for IPv4 networking. By default, IPv4 connectivity with IPv6 sockets is disallowed in Linux because they introduce security risks [89]. As a bad sign, of the total six applications referencing the *AI_V4MAPPED* flag, only one of them set the socket option as safe guard.

The mapped addresses could be also criticized for fixing applications to IPv6-based structures instead of promoting agility for variable types of address structures.

Also Teredo-based addresses could be used for avoiding the duplication at the client side. However, no API figures can be given because Teredo is a transparent solution on Linux. On Windows, application has to enable it separately with a socket option.

Authors' experience in using mapped addresses is that they are only usable for implementing internal data structures and should be converted before use with the Sockets API. We have observed in Ubuntu Linux that they cannot be used with UDP at all. With TCP, such addresses work only for receiving packets, and not for sending at all. Fortunately, it appears the amount of applications employing these "broken" addresses is quite small.

The Advanced Sockets API for IPv6 [128] appears largely unused as the number of IPv6 applications remains still limited. Similarly, the use of ancillary options appears constrained by the unpopularity of the *sendmsg()* and *recvmsg()* interfaces. Nevertheless, 65% of the applications employing the interfaces were also exploiting ancillary options. A majority of such applications could be categorized as network diagnostics, tunneling, proxying or routing-related software.

The constants introduced by the IPv6 Socket API for Source Address Selection [97] are available in Ubuntu Lucid even though the support is incomplete. The flags to extend the *getaddrinfo()* resolver and the proposed auxiliary functions remain unavailable and only source address selection through socket options is available. Nevertheless, we calculated the proportion of IPv6-capable client-side applications that make choices related to source address selection. As an upper bound, 66.9% percent of the applications choose source addresses explicitly based the dual use of *connect()* and *bind()*. This means meaning that a majority of IPv6 applications might be potentially interested of the extensions for IPv6 Socket API for Source Address Selection.

While the use of this API is limited by the number of IPv6 applications, we still tried to estimate this. We calculated the number of applications specifying the source address explicitly by calculating the proportion of applications referencing *bind()* but not referencing the *INADDR_ANY* constant, which results in 18.6% of the applications. Thus, this does not give yet a very promising future for adoption of the extensions.

Other Protocol Extensions

The use of SCTP was very minimal in our set of applications and only three applications used SCTP. *Netperf* is a software used for benchmarking the network performance of various protocols. *Openser* is a flexible Session Initiation Protocol (SIP) proxy server. Linux Kernel SCTP tools (*lksctp-tools)* provides userspace tools for testing SCTP functionality.

As with SCTP, DCCP was also very unpopular. It was referenced only from a single software package, despite it being easier to embed in an application by merely using the *SOCK_DCCP* constant in the socket creation.

As described earlier, HIP and SHIM6 have optional native APIs. Both of the protocols can be used transparently by legacy applications. This might facilitate their deployment when compared with the mandatory changes in applications for SCTP and DCCP.

The APIs for HIP-aware applications [70] may also face a similar slow adoption path as the APIs require a new domain type for sockets. While *getaddrinfo()* resolver can conveniently fill in any domain types, the success of this new DNS resolver (23.5%) is still challenged by the deprecated *gethostbyname()* (43.3%). SHIM6 does not face the same problem as it works without any changes to the resolver and connections can be transparently "upgraded" to SHIM6 during the communications.

The shared multihoming API for HIP- and SHIM6-aware applications [69] may have a smoother migration path. The API relies heavily on socket options and little on ancillary options. This strikes a good balance because *setsockopt()* is familiar to application developers (42.8%) and *sendmsg()* / *recvmsg()* with its ancillary option is not embraced by many (7%).

A Summary of the Sockets API Findings and Their Implications

Table 3.2 highlights ten of the most important findings in the Sockets APIs. Next, we go through each of them and argue their implications to the development of network applications.

Core Sockets API		
1	IPv4-IPv6 hybrids	26.9%
2	TCP-UDP hybrids	26.3%
3	Obsolete DNS resolver	43.3%
4	UDP-based apps with multihoming issue	45.7%
5	Customize networking stack	51.4%
OpenSSL-based applications		
6	Fails to initialize correctly	28.6%
7	Modifies default behavior	53.3%
8	OpenSSL-capable apps in total	10.9%
Estimations on IPv6-related extensions		
9	Potential misuse with mapped addresses	83.3%
10	Explicit IPv6 Source address selection	66.9%

Table 3.2: Highlighted indicator sets and their reference ratios

Finding 1. The number of hybrid applications supporting both IPv4 and IPv6 was fairly large. While this is a good sign for the deployment of IPv6, the dual addressing scheme doubles the complexity of address management in applications. At the client side, the application has to choose whether to handle DNS resolution over IPv4 or IPv6, and then create the actual connection with either family. As IPv6 does not even work everywhere yet, the client may initiate communications in parallel with IPv4 and IPv6 to speed up the process. Respectively, server-side applications have to listen for incoming data flows on both families.

Finding 2. The hybrid applications using both TCP and UDP amounted as much as TCP-only applications. Thus, many application developers seem to write many application protocols to be run on with both transports. While it is possible to write almost homogeneous code for the two transports, the Sockets API favors different functions for the two. This unnecessarily complicates the application code.

Finding 3. The obsolete DNS resolver was referenced twice as much than the new one. This has negative implications on the adoption of new Sockets API extensions dependent on the new resolver. As concrete examples, native APIs for HIP and source address selection for IPv6 may experience a slow adoption path.

Finding 4. We discovered a UDP multihoming problem at the server side based on our experiments with three software included in the data set. As an upper bound, we estimated that the same problem affects 45.7% of the UDP-based applications.

Finding 5. Roughly half of the networking software is not satisfied with the default configuration of networking stack and alters it with socket options, raw sockets or other low-level hooking. However, we did not discover any patterns (beside few popular, individually occurring socket options) to propose as new compound profiles for applications.

Findings 6, 7 and 8. Roughly every tenth application was using OpenSSL but surprisingly many failed to initialize it appropriately, thus creating potential security vulnerabilities. Half of the OpenSSL-capable applications were modifying the default configuration in some way. Many of these tweaks improved backwards compatibility at the expense of security. This opens a question why backwards compatibility is not well built into OpenSSL and why so many “knobs” are even offered to the developer[4].

Finding 9. IPv6-mapped IPv4 should not be leaked to the wire for

[4] One of the reasons may be that we need a way to turn on or off features depending on the peer’s SSL/TLS implementation, as some of the implementations of SSL/TLS are considered “broken” as they do not implement or implement incorrectly some of the bugs and/or functionalities of SSL/TLS.

security reasons described earlier and socket option *IPV6_V6ONLY* would prevent this leakage. However, only one out of total six applications using mapped addresses were using the socket option. Despite the number of total applications using mapped address in general was small, this is an alarming sign because the number can grow as the number of IPv6 applications increases.

Finding 10. IPv6 source address selection lets typically a client application to choose the type of an IPv6 source address instead of explicitly choosing one particular address. The extensions are not used yet, but we estimated the need for them with our set of applications. Our coarse-grained estimate is that two out of three IPv6 applications might utilize the extensions.

We have now characterized current trends with C-based applications using Sockets API directly and highlighted ten important findings. Of these, we believe findings 3, 4, 6 and 9 can be directly used to improve the existing applications in the data set. We believe that most of the remaining ones are difficult to improve without introducing changes to the Sockets API (findings 1, 2, 5) or without breaking backwards compatibility (finding 7). Also, some applications may not need security at all (finding 8) and the adoption of extensions (finding 10) may just take some time.

As some of the findings are difficult to adapt to the applications using Sockets API directly, perhaps indirect approaches as offered by network application frameworks may offer easier migration path. For example, the first two findings are related to management of complexity in the Sockets API and frameworks can be used to hide such complexity from the applications using the framework.

3.4.3 Network Application Frameworks

In this section, we investigate four network application frameworks based the Sockets API. In a way, these frameworks are just another "application" using the Sockets API and, thus, similarly susceptible to the same analysis as the applications in the previous sections. However, benefits of improving a single framework transcend to numerous applications as frameworks are "hotspots" for sets of applications. The Sockets API may be difficult to change, but it is easier to change the details how a framework implements the complex management of the Sockets API behind its high-level APIs.

Generic Requirements for Modern Frameworks

Instead of applying the highlighted findings described in Section 3.4.2 directly to network application frameworks, we make some modifications. Firstly, to have a more top-down approach, we reorganize the analysis into more high-level themes describing end-host naming, look up, multiplicity of names and transport protocols and security. We also believe that the reorganization may be useful for extending the analysis in the future.

Secondly, we arrange the highlighted findings according to their theme with three changes. First, a high-level framework does not have to follow IP-address oriented layout of the Sockets API and, thus, we investigate the use of symbolic host names as well. Second, the reconfiguration of the stack (finding 5) was popular but we could not suggest any significant improvements on it, so it is omitted. Third, we split initiating of parallel connectivity with IPv4 and IPv6 as their own requirements for both transport connections and DNS look ups.

The following list reflects the Sockets API findings as requirements for network application frameworks:

R1: End-host naming

R1.1 Does the API of the framework support symbolic host names in its APIs, i.e., does the framework hide the details of hostname-to-address resolution from the application? If this is true, the framework conforms to a similar API as proposed by Name Based Sockets as described earlier. A benefit of this approach is that implementing requirements R1.2, R2.2, R3.1 and 3.3 becomes substantially easier.

R1.2 Are the details of IPv6 abstracted away from the application? In general, this requirement facilitates adoption of IPv6. More specifically, it could be used for supporting of NAT traversal based on Teredo transparently in the framework.

R1.3 IPv6-mapped address should not be present on the wire for various security reasons. Thus, the framework should manually convert mapped addressed to regular IPv4 addresses before they are passed to any Sockets API calls or use the *AI_V4MAPPED* option as a safe guard to prevent such leakage.

R2: Look up of end-host names

R2.1 Does the framework implement DNS look ups with *getaddrinfo()*? This important because, e.g., Native HIP API and IPv6 source address selection extensions are dependent on this particular function.

R2.2 Does the framework support parallel DNS look ups over IPv4 and IPv6 as a latency optimization?

R3: Multiplicity of end-host names

R3.1 IPv6 source address selection is not adopted yet but is the framework modular enough to support it especially at the client side? As a concrete example, the framework should support adding new parameters to its counterpart of *connect()* call to support preferences for source address type.

R3.2 Does the server-side multihoming for UDP work properly? As described earlier, the framework should use *SO_BINDTODEVICE* option or *sendmsg()/recvmsg()* interfaces in a proper way.

R3.3 Does the framework support parallel *connect()* over IPv4 and IPv6 as a latency optimization?

R4: Multiplicity of transport protocols

R4.1 Are TCP and UDP easily interchangeable? "Easy" here means that the developer merely changes one class or parameter but the APIs are the same for TCP and UDP. It should be noted that this has also implications on the adoption of SCTP and DCCP.

R5: Security

R5.1 Does the framework support SSL/TLS?

R5.2 Does the SSL/TLS implementation provide reasonable defaults so that the developer does not have to set the details of the security?

R5.3 Is the SSL/TLS implementation initialized correctly?

ACE

ACE 6.0.0 provides a class for denoting one end of a transport-layer session called *ACE_INET_Addr* that can be initiated both based on a symbolic host name and a numeric IP-address. The support for IPv6 is thus transparent if the developer has chosen to resort solely on host names and uses

AF_UNSPEC in the instantiation of the class. Typically, the IP-addresses are also specified using strings which provides are more unified interface with host names. ACE allows storing of IPv4 addresses in the IPv6 mapped format internally but reverses them to the normal IPv4 format before returning them to the requesting application or using on the wire.

ACE supports *getaddrinfo()* function and resorts to *getnameinfo()* only when the Operating System (OS) (e.g., Windows) does not support *getaddrinfo()*.

ACE supports both connected (class *ACE_SOCK_CODgram*) and connectionless communications (class *ACE_SOCK_Dgram*) with UDP. We tested the UDP multihoming problem with test software included in the ACE software bundle. We managed to repeate the UDP multihoming problem with connected sockets which means that the ACE library shares the same bug as iperf, nc and nc6 as described earlier. Unsurprisingly, disconnected UDP communications did not suffer from this bug because ACE does not fix the remote communication end-point for such communications with *connect()*. It should be also noted that a separate class, *ACE_Multihomed_INET_Addr*, supports multiaddressing natively.

A client can connect to a server with TCP with class *ACE_SOCK_Connector* in ACE. The instantiation of the class supports flags which could be used for extending ACE to support IPv6 source address selection in a backwards compatible manner. While instantiation of connected UDP communications does not have a similar flag, it still includes few integer variables used as binary arguments that could be overloaded. Alternatively, new instation functions with different method signature could be defined in C++. As such, ACE seems modular to adopt IPv6 source address selection with minor changes.

For basic classes, ACE does not provide support for accepting communications simultaneously for both IPv4 and IPv6 at the server side. Class *ACE_Multihomed_INET_Addr* has to be used to support such behaviour more seamlessly but it can be used both at the client and server side.

Changing of the transport protocol in ACE is straightforward. Abstract class *ACE_Sock_IO* defines the basic interfaces for sending and transmitting data. An application instantiates *ACE_Sock_Stream* class to use TCP or *ACE_SOCK_Dgram* to use UDP which both implement the abstract class. While both TCP and UDP-specific classes supply some extra transport-specific methods, switching from one transport to another occurs merely by renaming the type of the class at the instantiation assuming the application does not need the transport-specific methods.

ACE supports SSL albeit it is not as interchangeable as TCP with

UDP. ACE has wrappers around *accept()* and *connect()* calls in its Acceptor-Connector pattern. This hides the intricacies of SSL but all of the low-level details are still configurable when needed. SSL is initialized automatically and correctly.

Boost::Asio

Boost::Asio 1.47.0 provides a class for denoting one end of a transport-layer session called *endpoint* that can be initiated through resolving a hostname or numeric IP. By default the resolver returns a set of endpoints that may contain both IPv4 and IPv6 addresses (IPv6 addresses are queried if IPv6 loopback is present). These endpoints can be given directly to the connect() which makes sequential connects to the address found in the endpoint set until it succeeds. The support of IPv6 is thus transparent if the developer has chosen to rely on host names. Boost::Asio allows storing the IPv4 addresses in the IPv6 mapped form. By default the mapped format is used only when the developer explicitly defines the query protocol to IPv6 and the query results contain no IPv6 addresses. The mapped format is used internally and converted to IPv4 before using it on the wire.

Boost::Asio uses POSIX *getaddrinfo()* if the underlying OS has support for *getaddrinfo()*, on systems such as Windows (older than XP) and cygwin Boost::Asio emulates the getaddrinfo by using *gethostbyaddr()* and *gethostbyname()*. By default, Boost::Asio does not support client-side IPv6 source address selection when connecting. Boost::Asio's connect would be able to benefit from the source address preference ordered endpoint set created by an extended *getaddrinfo()* described in [97] Section 7.

Boost::Asio does not support parallel IPv4 and IPv6 queries or does not provide support for simultaneous communications for both IPv4 and IPv6.

We tested the UDP multihoming problem with example software provided with the Boost::Asio. We managed to repeat the UDP multihoming problem with connected sockets which means that the Boost::Asio library shares the same bug as iperf, nc and nc6 as described earlier.

Boost::Asio defines basic interfaces for sending and transmitting data. An application instantiates *ip::tcp::socket* to use TCP or *ip::udp::socket* to use UDP. While both classes provide extra transport-specific methods, switching from one transport to another occurs merely by renaming the type of the class at the instantiation assuming the application does not need the transport-specific methods.

Boost::Asio supports SSL and TLS. The initialization is wrapped into the SSL context creation. In Boost::Asio, the library initialization is actually done twice as *OpenSSL_add_ssl_algorithms()* is a synonym of *SSL_library_-*

init() and both are called sequentially. PRNG is not automatically initialized with *RAND_load_file()*), *RAND_add()* or *RAND_seed()*. Although Boost::Asio implements class *random_device* which can be easily used in combination with *RAND_seed* to seed the PRNG.

Java.net

Java.net in OpenJDK Build b147 allows creation of a socket with various options. An application can resolve the address from a hostname, create the socket, and connect the socket to the given hostname with one function call or use the more traditional way of resolving the hostname to a set of addresses, creating the socket and connecting the socket to one of the addresses from the address set. The internal presentation of *InetAddress* can hold an IPv4 or IPv6 address and thus is transparent if the developer relies on the hostnames.

Java supports v4_mapped address format as an internal presentation. However, when used in transport it is converted to IPv4 address.

Java.net checks the existence of the constant *AF_INET6*, and that a socket can get an IPv6 address and creates the *InetAddr* factory accordingly. If java.net sees that the IPv6 is supported it uses the *getaddrinfo()* for resolution, otherwise it will use *gethostbyname()* for resolution. Parallel DNS queries simultaneously over IPv4 and IPv6 are not supported out-of-the-box. However, the SIP communicators ParallelResolver package [5] can be easily used to implement the support.

We tested the UDP multihoming problem with example software provided with the java.net. We managed to repeate the UDP multihoming problem with connected sockets which means that the java.net library shares the same bug as iperf, nc and nc6 as described earlier.

Java.net prefers TCP as the notion of socket always means TCP socket. If the developer needs UDP sockets the developer has to instantiate *DatagaramSocket*. Changing of the protocol is not easy because TCP in java uses streams for input and output, and UDP in java uses DatagramPacket objects as the sending means.

Java.net supports SSL, and TLS and the details of the usage are hidden inside the abstraction, although it is possible to influence the behaviour by changing options.

[5] net.java.sip.communicator.util.dns.ParallelResolver

Twisted

In Twisted 10.2, hostnames can be passed to TCP-based connections directly. However, the situation is asymmetric with UDP. A UDP-based application has to first resolve the hostname into an IP-address manually.

Surprisingly, the IPv6 support is mostly missing from Twisted. It supports resolving of IPv6 records from the DNS. Also, some methods and classes include “4” postfix to fix certain functions to IPv4 but, other than this, IPv6 is essentially missing. Mapped addresses are not a concern due to lack of proper IPv6 support.

The twisted framework uses *gethostbyname()* but has also its own implementation of DNS, both for the client and server side. As IPv6 support is missing, the framework cannot support parallel look ups.

Introducing IPv6 source address selection to Twisted would be relatively straightforward, assuming IPv6 support is eventually implemented. For example, Twisted method equivalents for *connect()* accept hostnames and the methods could be adapted to include a new optional argument to specify source address preferences. However, parallel connections over IPv4 and IPv6 remain unsupported due to lack of proper IPv6 support.

Twisted inherits the UDP multihoming issue from the Sockets API. We observed this by trying it with a couple of client and server UDP applications from the Twisted documentation.

TCP and UDP are quite interchangeable in Twisted when Endpoint class is used because it provides read and write abstraction. However, two discrepancies exists. First, Creator class creates discrepancy with a TCP-specific naming conventions in method *connectTCP()*. Second, applications cannot read or write UDP datagrams directly using host names but have to first resolve them into IP-addresses.

Twisted supports TLS and SSL in separate classes. SSL or TLS can be plugged into an application with relative ease due to modularity of the framework. The details of the security are configurable but Twisted provides defaults for applications that do not need special configurations. All of the details of TLS/SSL initialization, including seeding of the PRNG, are hidden from the application.

The initialization of the SSL/TLS is handled correctly inside the initialization of the *SSL* sub module. Also the error strings for PRNG are initialized correctly in *RAND* sub module. Although, *RAND* sub module does not automatically do the seeding, it is very easy to implement.

A Summary of the Framework Results

We summarize how the requirements are met by each of the four frameworks in Table 3.3. As it can be seen, the frameworks meet the requirements in different ways but some requirements are consistent in all. IPv6 source address selection (R1.3) seems easy to adopt in all frameworks and TLS/SSL is well managed (R5.1, R5.2, R5.3) in all frameworks. Conversely, all frameworks fail to support parallel IPv4/IPv6 connection initialization for clients (R3.3) and fail in UDP multihoming R3.2 as well.

Req.	ACE	Boost::Asio	Java.net	Twisted
End-host naming				
R1.1	✓		✓	(✓)
R1.2	✓	✓	✓	
R1.3	✓	✓	✓	✓
Look up of end-host names				
R2.1	✓	✓	✓	
R2.2			(✓)	
Multiplicity of end-host names				
R3.1	✓	(✓)		✓
R3.2				
R3.3				
Multiplicity of transport protocols				
R4.1	✓	✓		(✓)
Security				
R5.1	✓	✓	✓	✓
R5.2	✓	✓	✓	✓
R5.3	✓	✓	(✓)	✓

Table 3.3: Summary of the requirements for the frameworks

3.5 Related Work

At least three other software-based approaches to analyze applications exist in the literature. Camara et al. [25] developed software and models to verify certain errors in applications using the Sockets API. Ammons et al. [7] have investigated machine learning to reverse engineer protocol specifications from Sockets API based source code. Palix et al. [101] have automatized finding of faults in the Linux kernel [101] and conducted a longitudinal study.

We did not focus on the development of automatized software tools but rather on the discovery of a number of novel improvements to applications and frameworks using the Sockets API. While our findings could be further automatized with the tools utilized by Camara, Ammons and Palix et al., we believe it is in the scope of another paper already due to space limitations.

IP-address literals refers to addresses hard-coded to the application or addresses obtained without DNS resolution. Arkko et al. [12] describe their experiences from a transition to a NATted IPv6-only network. One of the problems in the transition originated from the IPv4-address literals embedded in applications. The experiment was cross-platform by its nature and considered different applications, such as web, email, games, instant messaging and VoIP.

Aside from internal unit tests, we did not find any IP-address literals in our set of software. Arkko et al. found literals especially in instant messengers and games. However, their findings do not conflict with ours because our study did not involve any games or the Windows OS. Also, they do not list the exact names of the software used, nor do they explain whether the address literals were built into the software or discovered by some other means such as configuration files, which were excluded from our study.

3.6 Summary

In this chapter, we showed results based on a statistical analysis of open-source network software. Our aim was to understand how the Sockets API and its extensions are used by current applications and application frameworks, and to examine how well the current frameworks support the observed usage scenarios and patterns. Thus, our aim was to conduct an empirical analysis that could be used to characterize network applications and create better frameworks to support their development. We reported ten interesting findings that included security, IPv6, and configuration related issues. Based on the findings we concluded that the Sockets API usage is heterogeneous and that it is difficult to introduce general modifications to the way applications utilize networking features. We partly addressed the extent of this challenge by suggesting fixes on security and UDP multihoming support. For example, we discovered that 28.6% of the C-based network applications in Ubuntu are vulnerable to attacks because they fail to initialize OpenSSL properly. Then, we turned to network application frameworks in order to be able to introduce the desired changes to multiple applications.

Our specific contributions are the findings based on a comprehensive statistical analysis of open-source software, the implications of the findings for updating and developing network applications, and the subsequent analysis of network application frameworks with the aim of a more flexible common network code that can be updated with new features. We investigated four frameworks and we analyzed their properties along different generalized dimensions of end-host naming, multiplicity of names and transports, name look up and security. A key finding in the frameworks was a multihoming problem with UDP-based connectivity that we verified with all four frameworks. With the suggested technical solution, frameworks can support better the pervasive multiaccess devices of today.

Chapter 4

Secure Identifier Resolution

Many efforts of the network research community focus on the introduction of a new identifier to relieve the IP-address from its dual role of end-host identifier and routable locator. This identifier-locator split introduces a new identifier between human readable domain names and routable IP-addresses. Mapping between identifiers and locators requires additional name mapping mechanisms because their relation is not trivial. Despite its popularity and efficiency, the DNS is not a perfect choice for performing this mapping because identifiers are not hierarchically structured and mappings are frequently updated by users. In this chapter we discuss the features needed to resolve flat identifiers to locators in a secure manner. In particular, we focus on the features and the performance that identifier-locator split protocols require from a mapping system. To this end, we consider a mapping system for an identifier-locator split based mobility solution and evaluate its performance.

4.1 Introduction

In the evolution of the Internet, IP-addresses initially served hosts as their identifiers *and* their routable locators. Although this dual role simplified many design decisions in the communication stack, it has been called into question because it hampers dynamics and flexibility in today's networks. Identifier-locator split protocols address this problem by limiting IP-addresses to being pure locators and by introducing a new identifier above the network layer. This new identifier is often not routable and serves purely for host identification. With this split, these protocols support current requirements in the Internet, including security, mobility, and multihoming.

While the identifier-locator split has clear benefits, it also introduces its own problems. In addition to the existing DNS mappings, identifiers must be resolved to one or more locators. Despite the similarity of both resolution steps, the identifier-locator split introduces requirements that the current name resolution architecture cannot handle in practice. First, the pattern of the requests changes from *name-to-locator* to *name-to-identifier* and *identifier-to-locator*, where the identifier may belong to a flat namespace and the locator may change frequently. Second, the system must support fast mapping updates for mobile hosts. Third, secure user-generated updates must be supported.

The current DNS was designed for an Internet that consisted of stationary nodes. As such, the DNS was built for frequent reads and occasional updates. In contrast, mobile hosts need to update their location in the identifier mapping system quickly to stay reachable. Such updates pose new challenges to performance and security since the DNS is mainly an administered environment in which end hosts typically do not have direct write access to their DNS records.

In this chapter we present an architecture that maps FQDNs to EIDs using the DNS, and that maps the EIDs of a host to its Routable LOCators (RLOCs) using a DHT. Our contribution consists of an in-depth analysis of the problem domain and the design of a secure resolution architecture for identifiers and locators for mobile users. As a proof of concept, we present practical experience with an implementation of the resolution architecture.

4.2 System Requirements

The introduction of end-host identifiers changes the way names are resolved at the beginning of a communication session. With the identifier-locator split, hosts have to resolve FQDNs to EIDs and EIDs to RLOCs. An essential question is whether the existing DNS name resolution infrastructure can cope with this task and how an alternative system should function. There are four problem areas that a name resolution structure for locator-identifier split mappings must tackle: a) In most cases the EIDs are based on a flat and often cryptographic namespace (e.g., the EID can be the hash of a public key of an asymmetric key pair identifying the host). It is known that DNS does not cope well with data that has no hierarchical structure. b) The architecture has to support user-generated and user-updated mappings. In the current DNS, names are mapped to a relevant authority controlling a subspace of the namespace. In some cases there is no authority for the user to turn to. For example, in HIP, the identifiers are self-created by the users, and in most cases the users do not belong to any organization that grants them modification rights to a DNS sub-domain, such that they could store and update their mappings. c) The architecture has to be secure; for example, it has to prevent attackers from forging identities and mappings of clients. Additionally, the system must prevent attackers from flooding the resolution system with bogus mappings to drown valid mappings. d) Finally, the system has to operate in an efficient manner.

4.2.1 Support for Flat Namespaces

The nature and structure of identifiers depends on the chosen identifier-locator split protocol. Identifiers can be divided into two categories. The first category represents human-readable identifiers at the application layer. Domain names, the most prominent human-readable identifier, are managed and structured in a hierarchical way, reflecting the hierarchy prevalent in the management of networked systems. The second category represents binary identifiers that may or may not be organized in a hierarchical way. These identifiers can consist of any sequence of bits without taking human readability into account. In the network community, there is a trend towards cryptographic identifiers to provide inherent security when addressing a host or service. An example of such a cryptographic identifier is HIP's HIT, a form of public key fingerprint. Such cryptographic names have little or no hierarchical structure, making it difficult to assign the management of the identity to an organization as the DNS does for human-readable names.

4.2.2 Rapid Mapping of User-generated Updates

Using DNS to store all the required mapping information (domain name, identifier, and locator) would suffice for stationary nodes under administrative management, but would not be a good choice for mobile nodes and for users who do not have modification rights to the DNS. To allow fast mapping updates for mobile hosts, which need to change their IP-address mapping rapidly to stay reachable, the DNS RRs would have to use low TTL values, or caching would have to be disallowed. However, high TTLs and caching are cornerstones of the scalability and performance of the DNS. Abandoning them for a considerable proportion of entries would seriously degrade the performance of the system as a whole.

This chapter considers the proposition that mobile nodes with access to the DNS should use it to map FQDNs to EIDs, and as the research community has adopted DHTs to handle flat namespaces, the chapter furthermore assumes that EIDs should be resolved using DHTs. DHTs do not employ hierarchical caching and thus allow for immediate mapping updates. However, using a DHT results in a higher communication overhead within the name mapping system (c.f. Section 4.4).

4.2.3 Securing Mapping Updates

The DNS, as a hierarchical and administered name resolution system, is widely regarded as secure. Even without cryptographic protections like DNSsec [11], fraudulent behavior requires access to the DNS infrastructure itself and is typically limited to a single compromised sub-domain. Tampering with DNS entries on a global scale requires considerable effort. In addition, DNSsec protects the system against spoofing attacks in which a malicious user tries to forge an answer from the DNS or tries to claim that the queried name does not exist.

However, as discussed before, the DNS system was not designed for large amounts of fast *user-generated* updates. Besides technical challenges, security issues arise when allowing users to modify the contents of the mapping system. The name lookup at the beginning of a communication session is a vulnerable phase. Tampering with it may allow direct as well as indirect DoS attacks (e.g., by invalidating the locator mapping or by redirecting traffic addressed to a popular host to a victim). Therefore, the system must be protected regardless of the resolution system.

We illustrate the arising issues using the example of OpenDHT[1] as a system that allows user-generated updates. OpenDHT has been proposed as one choice for a collaboratively managed DHT [5] (see Section 4.5 for further examples). OpenDHT is a publicly available DHT service running

[1] In May 2009, the maintainer of OpenDHT informed the community that the service would be discontinued. However, since it was a widely used service for years, we still use OpenDHT as a practical example of a DHT.

in PlanetLab, a world-wide testbed of several hundred servers. In contrast to other DHT systems, users do not have to run a local DHT node to be able to access it. OpenDHT does not require registration to insert and look up data. The open access philosophy of OpenDHT matches the requirements for global name resolution well, because requiring each end host to sign up for a name mapping service hardly matches the principles of the Internet. Available storage and bandwidth in OpenDHT are shared among all users [114].

OpenDHT stores one or several values under each key (e.g., the EID in a name lookup system). This convention is prone to flooding and index poisoning attacks [80]. In these attacks the malicious user stores as much false or random information under the attacked key as possible, thereby effectively drowning the original value. This allows malicious users to present seemingly correct information in the DHT. Index poisoning in an identifier-locator mapping service can even be used to mount distributed DoS attacks against victim hosts. Consider a case in which a malicious user uploads the victim's locator under the identifiers of some popular services. This would redirect the traffic destined for the services to the victim's system, thereby overloading its downlink.

There are two possible solutions for this problem: a) The DHT is agnostic with respect to the contents it stores and leaves it to the end host to implement security or b) the DHT enforces the correctness of mappings and updates to mappings before accepting them. Solution a) can be achieved by attaching additional authentication information to the stored mappings (e.g., digital signatures). EIDs based on a cryptographic namespace (e.g., HITs in HIP) simplify this approach because each host can use its EID to sign its locator set. Such signatures would enable a querier to identify the correct value among a set of forged locators. However, in an index poisoning attack, it would also mean that the querier would have to verify the signatures of many returned locators until it identifies a valid entry among the flood of bogus mappings. In contrast, in solution b), DHT nodes would verify the authenticity of the signatures before accepting a new key-value pair. This method requires replay protection to prevent attackers from republishing properly signed but outdated locator mappings.

4.3 Resolution System Design

This section summarizes the previous discussion and proposes a secure name resolution architecture in which clients map EIDs to locators using a DHT. We assume that the EIDs are derived from public keys (as HITs in

HIP are) and that hosts can prove the possession of an EID by using their private keys. We use HIP as an example of a host-based identifier-locator split protocol because it includes all of the security features that our proposed architecture requires. Moreover, we show how the security features of HIP support the requirements listed in the previous section.

4.3.1 General Design

Since the current DNS is sufficient to store the long-lasting mappings from FQDNs to EIDs and these mappings may be independent of the locator-split protocol, we treat the first name resolution step as an orthogonal issue and assume that appropriate measures are taken to ensure secure operation (e.g., by employing DNSsec). However, note that the FQDNs are resolved to EIDs instead of locators.

In the second name resolution step, a DHT is used to map the EIDs to locators. We assume that the EID (or a value that can be securely derived from it) is used as the key in the DHT. The value stored under the key consists of the public key of the host, its locators, a sequence number, and a signature created with the private key of the host. The signature and the sequence number prove to the clients and the DHT nodes that the locator mapping is authentic.

In the previous section, we noted that DoS and replay protection measures are needed to protect the DHT. This requires a challenge-response mechanism for verifying that the host owns the public-keys related to the EID for which it updates the locator mapping. If this verification succeeds, the mapping is stored; otherwise it will be dropped.

4.3.2 An Identifier Resolution System for HIP

In essence, the HIP BEX is a four-way handshake and key negotiation phase to create an IPsec security association between hosts. The BEX verifies that the peer owns the private key that correspond to the public key that was used to create their identities. The BEX also includes puzzle protection against DoS attacks and other flooding attacks. In our approach, we use the BEX as the challenge-response mechanism for verifying the ownership and freshness of the locators to be stored in the DHT.

In order to initiate the BEX, the *initiator* (the host that initiates the communication) needs to know the HIT of the *responder* (e.g., a server) and a way to map the HIT to an IP-address. Currently HIP offers two different ways to perform the resolution. In our solution, both of these approaches are used. Firstly, DNS can be used to store HIP-related identifiers using

HIP RRs [94] protected by signatures. This allows for translation of FQDNs to HIs. The resolver then issues an *A* query to map the HIT into the host's IP-addresses. Alternatively, HIP can utilize the HIP DHT interface [5].

We use the client authentication in the HIP BEX to prevent attackers from inserting forged locators for EIDs of other hosts into the DHT. By requiring a HIP connection between the client and the DHT node, the client has to prove that it is uploading mappings for its legitimate host identity – the key in the DHT. Implementing this authentication check is simple and can be done through the standard Berkeley Sockets API. The HIT, as an IPv6-compatible identifier, can be used directly by any IPv6-capable DHT. The authentication, as well as the basic DoS protection, are handled on the HIP layer. The only modification required to the DHT is a test for equality of the HIT and the key in the *put* message. The self-certifying property of the HIT obsoletes further authentication measures like client certificates or user registration.

Depending on the security architecture of the DHT, a client should either perform the BEX with a DHT gateway node, or with the node storing the EID and locator information if the DHT system cannot be regarded as secure. Such protection is only possible if the DHT is used exclusively for HIT-to-locator resolution, because a general-purpose DHT is not able to guarantee the cryptographic binding between a host and the updated keys.

4.4 Evaluation

In this section we study the feasibility of the system by analyzing our proof-of-concept implementation. Our prototype uses the Host Identity Protocol as the identifier-locator split protocol. HIP was chosen because it readily includes cryptographic identifiers that can be used for authentication, a built-in challenge-response mechanism, and a cryptographic puzzle mechanism for DoS prevention. We implemented an OpenDHT Interface [5] for the *HIPL* implementation[2] to store the identifier-locator mappings.

We begin our performance evaluation by analyzing the number of required cryptographic operations and messages for inserting a new entry into the DHT based on our HIP-centric security solution. We focus on the performance of the resolution system and not on the performance of mobile nodes. We also provide measurements of the processing times for the Bamboo DHT[3] and for OpenLookup v2,[4] to show the resolution performance

[2] HIPL, `http://www.infrahip.net`, 22.9.2012

[3] `http://sites.google.com/site/lxpworkroom/bambooipv6version`, 22.9.2012

[4] `http://openlookup.net/`, 22.9.2012

Table 4.1: Computational complexity of cryptographic operations in HIP.

	operations per sec.	ms/operation
HMAC(MD5)	42267	0.02
SHA-1	30809	0.03
DSA signature	1887	0.53
DSA verify	1645	0.61
RSA signature	890	1.12
RSA verify	18502	0.05
DH key generation	51	19.64

of a real system.

4.4.1 Feasibility

Performance is a primary concern for a resolution system that employs online public-key operations. The HIP BEX is dominated by the processing times for creating and verifying the public key signatures. Hence, we measured the performance of these operations to estimate the number of clients that a server is capable of serving per second. The cryptographic operations are only relevant for write operations because reads do not require authentication. Hence, reads will be drastically faster. Moreover, we do not consider routing overhead in the DHT in this measurement either. The goal of the analysis is not to give an accurate estimation of the expected performance of a world-wide system, but rather to provide a general impression of the feasibility of using HIP for securing a DHT system for HIT-to-IP mapping.

We used a quad core Intel Xeon 5130 running at 2 GHz with 2 GB of main memory to perform the cryptographic calculations. However, the cryptographic measurements used only one core. We conducted the cryptography tests with the OpenSSL 0.9.8g speed test. The results of the tests are shown in Table 4.1. We used 1024-bit keys because that is the default size for the Rivest Shamir Adleman (RSA) and Digital Signature Algorithm (DSA) keys in the HIP for Linux implementation, which we used for latency measurements. For the hash functions, we used the maximum block size of 8192 bytes.

Based on the information presented in Tables 4.1 and 4.2, we can calculate that it will take circa 20.8 ms to complete the cryptographic calculations needed in the BEX on the server side (1 RSA signature, 1 RSA verification, and 1 D-H key generation). This amounts to 192 key updates processed by each DHT node per second. Considering a system comparable to OpenDHT, which consisted of about 150 nodes on average, the system could process about 28,800 updates per second. These estimations show that even a moderately-sized system can support a substantial number of mobile devices.

4.4.2 Resolution and Update Delay

The resolution and update delay of the system is a crucial factor for clients because the resolution step precedes every communication to each host for which the HIT-to-IP mapping is not known. The update delay determines how long the locator information in a DHT stays outdated upon a change of the locator. In our system, read accesses are protected by a signature in the DHT resource record while write accesses are protected using HIP between the client and the DHT. Using HIP prolongs the update process by the time required for cryptographic processing plus two Round-Trip Times (RTTs) for establishing the HIP association to the gateway or DHT node.

We measured the mean latency of an IPv6-enabled Bamboo DHT and OpenLookup v2 to determine the resolution performance of these systems. OpenLookup v2 implements the same XML-RPC client interface as the Bamboo DHT, but it is not strictly speaking a DHT, and it does not share data with OpenDHT. OpenLookup v2 is an administratively decentralized system based on full data replication.

Bamboo DHT and OpenLookup v2 were running on the same hardware as described in Section 4.4.1. We used a laptop with Intel Core 2, 2 GHz CPU processor with 2 GB of main memory as the client. All machines involved in our measurements were located in our local Gigabit network with a mean round-trip latency of 0.88 ms (std.dev. 0.03 ms). Since we only modified the lookup and update API, we focus on the communication between the end host and the resolution system. For this reason, we made our measurements using a Bamboo DHT configuration containing just one node. Thus the results obtained do not reflect the expected total lookup time of a world-wide deployment requiring routing within the DHT.

Figure 4.1 shows that OpenLookup v2 performs slightly better in the lookup operations in comparison to the Bamboo DHT. The larger mean

Table 4.2: Cryptographic and communication overhead of the HIP BEX.

	Verify		Sign		DH key	
	PK	HMAC	PK	HMAC	Generation	# of msgs
Initiator	2	1	1	1	1	2
Responder	1	1	1	1	1	2

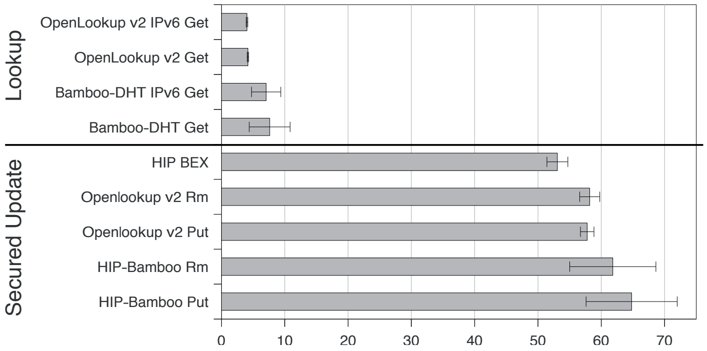


Figure 4.1: Latencies of update and get operations (in ms).

values and standard deviations for the Bamboo DHT are due to an unnecessary periodic delay caused by the queue management in the iterative lookup procedure, resulting in an extra delay for a small fraction of the requests (approx. 10%). Since the shortest latencies of the Bamboo DHT (IPv4 3.8 ms, IPv6 4.0 ms) match the shortest lookup times of OpenLookup v2 (IPv4 4.2 ms, IPv6 4.0 ms) we assume that both systems achieve a similar level of performance under realistic conditions.

The tested systems do not support updates of keys. However, by deleting a key and re-inserting it with new locator information, updates can be achieved. The latency of updating a record in the system is 116.4 ms (±0.8 ms) for OpenLookup v2 and 127.6 ms (±5 ms) for the Bamboo DHT. In fig 4.1 we also show the latencies of the HIP BEX in the test environment and the latencies of a Bamboo DHT utilizing HIP. Utilizing HIP in the Bamboo DHT and in OpenLookup v2 did not add latency, other than the 53 ms (±3.3 ms) caused by the BEX. Otherwise, the Bamboo DHT and OpenLookup v2 utilizing HIP performed as expected from the results in the *get* case. As pointed out in the previous section, the main cause of the delay introduced by HIP is the cryptographic operations during the BEX. In contrast to our estimation of the additional cryptographic load on the DHT gateways, the additional delay of 53 ms in fig 4.1 also includes the processing time of the client, the packet processing, and the network latency. Our measurement focused on the performance of the gateway and do not take into account the higher RTTs between the client and the gateway under realistic conditions. As expected, HIP introduces a notable delay for updates; however, at the same time it eliminates the possibility of index poisoning and forged locator updates, without requiring additional administrative measures like user registration.

4.5 Related Work

Mathy et al. [86] describe how LISP-DHT serves as an efficient and secure mapping service for the LISP 3 variant. LISP-DHT requires every Autonomous System (AS) to have its own DHT node to serve identities in the AS. In LISP-DHT, security is based mainly on the assumption that the architecture is administered and joining the DHT requires a valid X.509.v3 certificate. To improve efficiency, LISP-DHT proposes the use of a Stealth DHT, where client nodes may acquire DHT routing information (but do not take responsibility for any data segment on the ring). In this way, the stealth nodes can inject lookups into the system in a more efficient way than by always directing queries via a gateway node. LISP reduces latencies by caching and so hinders mobility. Mobility in LISP and its influence on name resolution are currently under design [33].

In the Node Identity architecture [4], the node identities are the public keys of public-private key pairs. Name resolution in the Node Identity architecture uses the DNS to map FQDNs to EIDs, while EIDs are mapped to locators using a global DHT shared by all node identity routers. The Node Identity architecture is similar to LISP in the sense that both are network-based and need customized routers to work. The security of the mappings in the architecture is only briefly addressed by stating that the security is inherited from registration security. However, the registration security is not discussed in detail.

DHT-MAP [81] proposes a mapping system, useful for LISP and similar protocols. The difference relative to other solutions is that EIDs are mapped in the DHT to the address of a server that handles the resolution to a host's real RLOC. Mobility is supported by allowing the mobile host to register with the resolution server of the access network to which it attaches. In this way, DHT-MAP avoids triangular routing and the concept of a *home network*. Luo et al. [81] state that their approach may allow EID spoofing attacks, and they suggest a challenge-response mechanism similar to the mechanisms provided by HIP.

Baumgart [15] proposes a distributed two-stage name resolution service (P2PNS) built on top of a DHT. That paper presents requirements and solutions similar to ours but does not discuss mobility. In P2PNS, flooding attacks are hampered by introducing computational puzzles that have to be solved before the mappings can be inserted. As an additional feature, the number of values under a key is restricted. When a key is queried from P2PNS, it is queried in parallel from all replicas that have the key and its value. Based on the received values, the issuer of the query makes a majority decision.

4.6 Summary

In this chapter, we presented a discussion about identifier resolution for identifier-locator split protocols and pointed out the shortcomings of the current DNS. Based on our observations, we described an architecture for secure identifier-locator mappings based on a distributed hash table. In particular, we discussed three core problems of name resolution for host-based identity locator split protocols: a) support for flat namespaces, b) rapid user-generated updates, and c) the security of the mappings.

We address these problems by implementing secure key updates based on the cryptographic properties of the identifiers in the HIP. Our system works with user-generated identities and does not require any user management or the deployment of a global PKI system because it makes use of the self-certifying identities in HIP. With its identity concept and IPv6 compatibility, HIP integrates nicely into existing lookup systems and enhances their security features with DoS resilience and authenticated locator updates. Our performance analysis of the HIP-enabled DHT API demonstrates the feasibility of the architecture and indicates that employing HIP as a security solution provides acceptable performance with considerably increased security.

Chapter 5

Separating Friends from Spitters

Undesired mail, such as commercials, is called spam in mail services. Spit is what spam is for mail services, unsolicited communications. The difference in spit and spam is that spam can be checked before the delivery to the recipient and spit can be reliably detected only after the call is made.

VoIP community has adopted a more peer-to-peer aproach, in which the registrars and proxies are located on the participating nodes, rather than on separate servers. Industry has also been quite keen in the developement of such approach, especially ones that use identifier-locator split protocols. The lack of centralized authorities and the usage of long trust paths makes detecting spit even harder a task.

In this chapter we describe a system to disseminate information of friendships that are based on an end-host based identifier-locator split protocol. In our solution the existing *buddy lists* are used to introduce one-hop connections in the system.

In our system we rather detect friends than spit or spitters. Moreover, our solution can be generalized for situations where the before-hand inspection of the content is impossible or otherwise hard to implement.

5.1 Introduction

Everyone knows how annoying it is to open a mailbox and see it littered with unsolicited mail. We have seen this problem grow in proportion over the years and now we see how it spreads across different mediums. Everything that draws in large crowds of users will eventually draw in hordes of spammers in a form or another.

VoIP is one of the many new technologies that draws in users as well as spammers. Spam over Internet Telephony (SPIT), the equivalent to spam in VoIP, is more intrusive as a call effectively disrupts what ever the user was doing at the moment and the spam does not. Blacklisting is the most prominent way to fight against spam but it has its downsides and there is no sure way of knowing if the call is SPIT before answering it.

In VoIP, SIP is the signalling protocol used to create the sessions between clients. In order to get a more scalable and less vulnerable SIP, the IETF and the networking industry have been designing a pure peer-to-peer alternative that does not need any centralized servers, i.e., P2PSIP. As the architecture moves away from the centralization, the research community has proposed the usage of trust paths in order to identify friends [44].

In this chapter we argue that trust paths longer than one-hop are too long to retain trust. Moreover, we argue that hiding of the path structure from the search result, for privacy reasons, enables Sybil attacks without any fear of retaliation for the attackers.

Our solution is based on host based identifier-locator split with self-certifying cryptographic identities. These identities are used as the entities in the certificates that are used to communicate the one-hop paths between participants. Moreover, we show how this information can be used in the GUI to provide more information for the users to make better trust decisions.

It should be noted that the solution is also valid for other situations where the before-hand inspection of the content is impossible or inconvenient. First, the content such as live audio and video streams cannot be inspected before receiving it as it does not exist prior to the receival. Second, the inspection of a large file transfer can be impossible, difficult or even unwanted waste of resources as the file has to be transfered and stored on the inspecting host or middlebox.

5.2 Background

SIP is a signaling protocol for conferencing, telephony, instant messaging and presence. SIP can create, modify and terminate two- or multi-party sessions. In SIP the user equipment (User Agent (UA)) is the network endpoint that creates or receives the calls. The actual architecture comprises of three elements: proxy servers, registrars, and redirect servers. Proxies handle the routing of SIP messages between UAs and they can implement access control. Registrars maintain the location information for the UAs, i.e., registrars translate SIP Uniform Resource Identifiers (URIs) into one or more IP-addresses. Redirect servers can be used to redirect SIP session invitations to external domains.

For scalability and security reasons the SIP research community has introduced a peer-to-peer alternative for SIP called the P2PSIP [51]. The infrastructure elements in SIP are defined as logical entities and are usually co-located on the same hardware. This distinction makes it easier to move the infrastructure elements to the UAs as P2PSIP does. The HIP [91, 92, 37] has been proposed to be used for the connection maintenance and transport protocol for the P2PSIP because of its support for mobility, multihoming, NAT traversal, and security features [18, 17]. For VoIP applications NAT traversal is a major concern and by offloading it to HIP the connection management in VoIP applications becomes simpler. Moreover, by using HIP VoIP applications gain support for transparent mobility without any modification to the application.

It could be argued that why to use host identities in access controlling, as the modern operating systems are multi-user systems. First, in our opinion most of the machines, such as laptops, smart phones, etc. are more personal than ever and they even require additional authentication methods, such as Personal Identification Number (PIN) codes on smart phones, and account passwords on laptops, etc. Second, the usage of lower level identifiers to access control the connection avoids the need for deep packet inspection on higher layers. For example, using SIP URIs for access control would need inspection of SIP URIs in the SIP messages.

5.3 Requirements for the trust paths

WOT originates from the PGP and is the starting point for most trust path related solutions. WOT is a decentralized trust model that represents the trust that the users have for each other. A path through the WOT from the initiator to the responder of the communication is presented as a token of trust to the responder. We argue that long trust paths are complicated and do not represent real trust.

First, long trust paths increase complexity of the solutions and can have unforeseen problems. Heikkilä et al. allow long trust paths scheme in their Pathfinder [44] approach for protection against spit. Pathfinder uses one or more centralized servers as privacy protected search engines. The information that the Pathfinder servers use is protected with a hash scheme. However, we argue that by protecting the privacy of the links the Pathfinder allows Sybil attacks. A malicious user can set up a node that acts trustworthily among other nodes. While, acting nice, the node then links the real spitters to the WOT. Now, when the spitter tells the Pathfinder to search for a path from itself to the target, the path is found. What makes this effective is the fact that the malicious user connecting the spitters to the network does not have to fear of punishment as the path is hidden along the responsible node.

Second, long trust paths do not represent real trust. To be trustworthy, long trust paths require that every node on the path are honest and do not lie in their statements. The longer the path grows, the harder it becomes to trust every node's decision on the path. In a country, such as Finland, it may be possible to connect the author to the President with a reasonably short path. However, it does not say anything about the trust between the President and the author in either way. In our opinion trust degrades quite fast, in the matter of few hops.

We propose, based on the discussion above that, only one-hop trust paths are sufficient.

5.4 Our solution

HIP BEX creates an IPsec tunnel between the participating hosts and the traffic is transported inside the tunnel between the hosts. Due to the nature of IPsec being a host-to-host connection, we access control the traffic coming from the tunnel by using the HITs from the packet headers. This way we bind the tunnel for certain usage, VoIP in our case.

In our prototype we used OpenLookup v2 as our credential storage system. Although, any storage that implements anonymous key-value storage service with the XML-RPC interface can be used, such as Bamboo DHT [1] or OpenLookup v1 [2]. The storage does not have to be a DHT but could be a centralized system or even less centralized system than DHT. But it

[1] `http://bamboo-dht.org`, 22.9.2012

[2] `http://openlookup.net/`, 22.9.2012

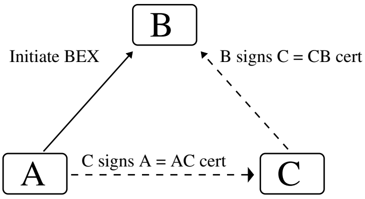


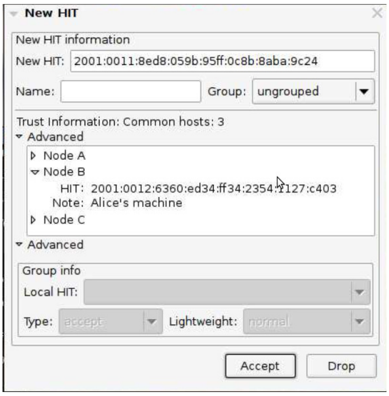
Figure 5.1: Forming of a trust relation ship between host A and host B by presenting the certificate given to host C to host A.

should be noted that by using a centralized system it may become a bottleneck for the system and in less centralized systems the revocation of the certificates may become overly complicated.

Upon an incoming connection from an unknown host the user is prompted with a dialogue, in which a question is asked, whether to accept or drop the new connection. If the connection is accepted, a certificate is created and uploaded into the storage system. This certificate's semantic statement is that the subject has trusted the host enough to accept a connection from the host. Upon subsequent connections from the subject, the certificates can be checked and the user is not bothered with the dialog.

The certificate contains the HITs of the participants as the issuer and subject. Moskowitz et al. [92] say that HIT collision maybe possible, while improbable. For this reason the certificates contain also the full HIs. The certificate contains also a short time frame in which the certificate is valid. This makes the revocation easier, as the issuer can just stop renewing and uploading the certificate. For the host the information contained in the certificate is enough but in the *new HIT* dialogue of HIPL the HITs are also presented to the user. Long hexadecimal strings are hard for the user to recognize and for this reason we added an issuer given name into the certificate. Moreover, we think that because the names are given by first-hop friends they most probably have a meaning for the receiver. As the certificates do not contain any location information they are also suitable for mobile clients as there is no need to update the certificates in the system upon mobility events.

In our system malicious users can try to lie about their trust but the lying would be noticed easily and the lying friend could be punished. Moreover, the system could have an additional rating (e.g., a floating point value from 0 to 1, where 1 is complete trust) for the trust that could be increased or decreased based on the observed behaviour. In practice this rating could be enforced by increasing the puzzle sizes and/or by throttling the connection by limiting the bandwidth of host with low ratings and vice versa for



New HIT information

New HIT: 2001:0011:8ed8:059b:95ff:0c8b:8aba:9c24

Name: Group: ungrouped

Trust Information: Common hosts: 3

Node A

Node B

HIT: 2001:0012:6360:ed34:ff34:2354:1127:c403

Note: Alice's machine

Node C

Group info

Local HIT:

Type: accept Lightweight: normal

Figure 5.2: Acceptance dialogue presented to the user upon an incoming connection. The dialogue includes our modifications.

hosts with high ratings. In the end, the meaning of the rating is left as a local policy, this way the hosts can make independent choices on the level of enforcement. In the worst case the host could deny all connections from the subject and remove its certificate from the system. By tying the used puzzle size in the BEX to the used trust rating, the responder could also choose the puzzle sizes for the initiators based on their ratings. The used computational cycles to solve the puzzle would constitute a payment for the service needed by the initiator of the connection.

HIPL [3] implementation has an identity management GUI that filters all HIP based control traffic through it. Incoming connections are filtered on the receive of I1 control packets. Incoming connection prompts the user to make a decision on accepting or dropping the connection. The GUI is used also used to group known HIs in to groups and give them group based attributes. GUI can also be used to show dropped HI.

During our implementation efforts we changed the triggering point for the access control of incoming HIP control packets. Previously the trigger point was in the receiving of I1 control packets and we moved the trigger point after the handling of I2 control packet. At this stage the user would not see prompts for all easily forged I1 control packets. Instead the responder will see the incoming connection prompt only after the initiator has solved the puzzle successfully. Moreover, the signature in the I2 control packet has proved that the initiator actually owns the corresponding public key, from which the used HIT was created.

In the initialization phase, i.e., when the host starts the identity man-

[3] http://www.infrahip.net, 22.9.201

agement GUI, the hosts upload the certificates to the used storage service. The upload can happen sequentially or in parallel but for our purposes uploading sequentially was adequate. In the example, host B has previously accepted a connection from the host C and uploaded the certificate to the storage system (CB cert in the Figure 5.1). Moreover, host C has previously accepted a connection from host A and has uploaded the certificate to the storage system (AC cert in the Figure 5.1). When the initiator starts the connection it queries all the possible concatenations of its own HIT used in I1 and I2 control packets as the destination HIT and its friend's HITs and tries to find one or more suitable certificates to be presented to the responder in the BEX. The certificates are stored by using the hash of the concatenation of the issuer and subject HITs from the certificate as the key. This way we retain some privacy as the key cannot be directly guessed. If only the issuers HIT was used the malicious host could easily gather the friend list of a host. Using the hash of the concatenation of HITs the key is obfuscated so the malicious user has to guess what are the HITs of the friends and while the malicious user would guess one friend it would not reveal other friends of the host.

In our example the found trust path is the following:

$$B \Rightarrow C \Rightarrow A$$

In the scope of the certificates we have the following:

$$ACcertificate \Rightarrow CBcertificate$$

In our example, the host A finds two certificates: one given by host B for host C, and one given by host C for host A. Host A transports the certificate AC in the BEX to the host B. There is no need to transport the certificate CB as the host C is already in the friend list of host B and is trusted.

Upon receiving the I2 control packet host B verifies its signature and checks that the puzzle is solved correctly. If these checks were performed successfully the identity management GUI prompts the user asking to accept or drop the connection (see Figure 5.2). In this prompt, in addition to the used HITs, the user is presented with the user friendly name of host A given to it by host C.

5.5 Evaluation

We measured the mean latency of OpenLookup v2, using both IPv4 and IPv6, to determine the query performance of the system. We used a quad

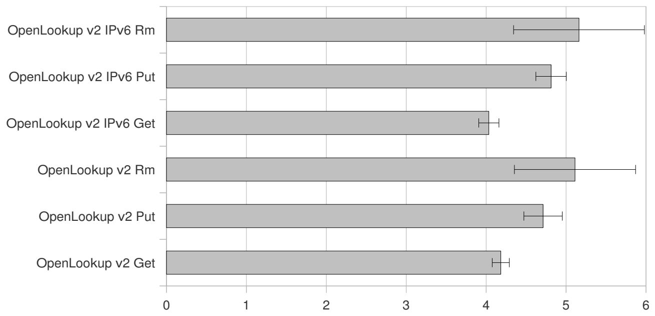


Figure 5.3: Average latencies measured from the storage systems in milliseconds.

core Intel Xeon 5130 running at 2 GHz with 2 GB of main memory as the server and we used a laptop with Intel Core 2, 2 GHz CPU with 2 GB of main memory as the client. All machines involved in our measurements were located in our local Gigabit network with a mean round-trip latency of 0.88 ms (std.dev. 0.03 ms). We concentrated more on the query performance and not on the update performance, since the friendships seldom change and the frequency to refresh the credentials in the storage can be even a week. The measurements were done by querying random keys with values containing certificates of varying sizes.

From the results depicted in the fig 5.3 it can be easily calculated that an initiator, with 100 friends, can sequentially query the system in maximum of circa 418 milliseconds. In the Internet the RTT times between the client and the server increase the latency. With RTT of, for example, 70 milliseconds between the client and the server will the total time be circa 7418 milliseconds. In our opinion even the longer latencies are acceptable because it bothers only the initiator of the connection. Moreover, the query performance can be optimized by querying in parallel and by caching the results locally on the client and thus avoiding subsequent queries of credentials.

In our experiments we noticed that certificates can pose a size problem for the control packets. If multiple suitable certificates are found, we could send multiple certificates in the control packets to the responder so that the information could be prompted to the user. However, the average size of a I2 control packet, using 1024 bit RSA keys, is circa 850 bytes and it occupies most of the minimum Maximum Transmission Unit (MTU) of IPv6 (1024 bytes) and exceeds the minimum MTU of IPv4 (512 bytes). When we add one or more certificates to the I2 control packet, the size will exceed even the IPv6 minimum MTU. This makes it very probable that the packets are fragmented on the wire. The solution for the size problem is left for further study.

5.6 Summary

In this chapter we presented a discussion on how to separate friends from spitters, using SIP as an example. Based on our observations, the discussion identifies two problems in the proposed trust path solutions: a) trust path solutions that hide the path details from the users allows Sybil attacks, b) real life trust does not extend over multiple hops.

We addressed these problems by using the self-certifying cryptographic identities of HIP to create one-hop trust paths to be used to access control the incoming calls. Our solution is in a sense a distributed white list based on host identifiers that identify the incoming connections from friends. We also provided measurements from live storage systems in order to estimate the time required to gather sufficient trust information and discussed about the size issue caused by the addition of certificates to the control packets.

Chapter 6

Secure and Efficient IPv4/IPv6 Handovers Using Host-Based Identifier-Locator Split

Internet architecture is facing at least three major challenges. First, it is running out of IPv4 addresses. IPv6 offers a long-term solution to the problem by offering a vast amount of addresses but is neither supported widely by networking software nor has been deployed widely in different networks. Second, end-to-end connectivity is broken by the introduction of NATs, originally invented to circumvent the address depletion. Third, the Internet architecture lacks a mechanism that supports end-host mobility and multihoming in a coherent way between IPv4 and IPv6 networks.

We argue that an identifier-locator split can solve these three problems based on our experimentation with the Host Identity Protocol. The split separates upper layer identifiers from lower network layer identifiers, thus enabling network-location and IP-version independent applications.

Our contribution consists of recommendations to the present HIP standards to utilize cross-family mobility more efficiently based on our implementation experiences. To the best of our knowledge we are also the first ones to show a performance evaluation of HIP-based cross-family handovers.

6.1 Introduction

The IPv6 address space is drastically larger than for IPv4, but IPv6 has not experienced a wide-scale deployment yet. Concurrent use of both addressing families causes problems for both network software and management due to non-uniform addressing. Existing legacy software is hard-coded to use IPv4 addresses and some of it can never be updated to support IPv6 due to the applications proprietary nature. The fact that IPv4 address space is almost exhausted does not make things any easier because a host might acquire only an IPv6 address in future networks. As a consequence, proprietary network software may have trouble to access the Internet in the future.

End-to-end communication between two hosts is not guaranteed anymore, even considering protocols for traversing NATs. To make things even more complicated, end-host mobility arises as a new requirement for the Internet. Users are used to staying continuously in contact with each other using cellular phones and may also want the same with other portable devices. Users may want to benefit from access technologies, such as WLAN and 3G, available on phones and other devices. Multiaccess is desirable for users, for example, to reduce monetary costs, to assess benefits from device proximity, or to obtain a faster connection. Even though cellular networks support mobility transparently, the same does not apply to WLAN mobility.

In the current Internet, an IP-address both identifies and locates a host. However, this binding breaks when the address of the host changes. This is a problem both for relocating the mobile host and for maintaining long-term transport layer connections, which break upon such a mobility event.

The identifier-locator split decouples the host identifier from its topological location. The new host identifier is present at the transport and upper layers to provide applications a fixed identifier independent of network location. The identifier-locator split introduces a layer between the transport and the network layers, and translates the identifiers dynamically into routable addresses and vice versa.

The concept of the HIP [92, 67] is based on identity-locator split. It provides security, global end-host mobility, multihoming, NAT traversal, and Rendezvous/Relay services. The HIP specification [93] describes end-host mobility and multihoming but handovers across IP families are left for further study. In this chapter, we describe HIP-based cross-family handovers based on our implementation experimentation and performance evaluation. Compared to previous work [145, 98, 55, 74, 144], we focus on Linux rather than the Berkeley Software Distribution (BSD) networking stack.

6.2 Related Work

In MIP [108, 52], each node has a home address that identifies the node independently of its location. When the mobile is not located in its home address, the mobile node informs its HA on its current address (CoA). Datagrams destined to the mobile node are tunneled to its current address through its home agent. MIPv6 includes an optimization that allows end-hosts to route MIPv6-related traffic directly between them without such a triangular routing through the home agent. IPsec and MOBIKE [27] [32] can be used to protect MIP traffic.

The MOBIKE protocol offers mobility functionality similar as in HIP. For example, the LOCATOR is similar to ADDITIONAL_*_ADDRESS (where * is IPv4 or IPv6) and the return routability test is similar as in HIP. The MOBIKE standards allow the mobile node to send additional addresses of different family than those currently in use [134].

A MIPv4 extension [133] introduces dual stack mobility by tunneling IPv6 over IPv4. This approach needs dual stack HA and triangular routing to offer movement between IPv4 and dual stack networks. Cross-family handovers, where nodes move from IPv4 network to IPv6 networks or vice versa, is left somewhat unclear in the specification.

Teredo is an IPv6-over-IPv4 tunneling protocol that includes a mechanism to avoid triangular routing [50]. Teredo uses UDP encapsulation and encodes additional information into the IPv6 addresses. Teredo defines a dedicated IPv6 prefix (2001:0::/32) for the tunnel which can be used by any IPv6-capable networking software.

SHIM6 [96] is a layer 3 multihoming protocol that offers locator agility for the transport protocols. SHIM6 has multiple similarities when compared with HIP. For example, the protocol formats are identical and the initial handshake is similar. At the time of writing SHIM6 did not have specification for the usage of IPv4. In our opinion, our work with cross-family handovers is beneficial also for the SHIM6, when the usage of IPv4 is standardized for SHIM6.

Jokela et al. [54] first discussed about cross-family handovers in HIP but showed no performance or implementation evaluation. Their primary environment was FreeBSD, while we have implemented cross-family handovers for the Linux networking stack. Furthermore, we specifically focus on the fault tolerance aspects of handovers rather than load balancing.

6.3 Cross-family IPv4/IPv6 Handovers

6.3.1 Scope of HIP Handovers

In this chapter, a handover refers to a change in the locator set of an end-host. When the locator set changes, the end-host can perform a handover procedure to sustain HIP and upper layer connectivity. A vertical handover describes end-host movement between different link-layer access technologies, such as WLAN and Universal Mobile Telecommunications System (UMTS), and a horizontal handover refers to movement within the same type of access technology devices. HIP can support both vertical and horizontal handovers because it operates above link layer. The focus of this chapter is on end-to-end handovers even though HIP facilitates also end-to-middle operation using a HIP proxy [121].

In a Make-Before-Break (MBB) handover an end-host obtains a new locator before it loses its current address. In a Break-Before-Make (BBM) handover, the end-host loses its current address before it obtains a new address. The latter results in a gap in connectivity during which the end-host is not reachable which causes disruption to existing connections at the transport layer.

6.3.2 Cross-Family Handovers

HIP specifications [92, 93] offer a possibility to include LOCATOR parameters in the R1 and I2 packets. However, these two documents explain only the load balancing case with the preferred bit set. When a host sets the preferred locator, its peer is forced to switch to it immediately. We argue that the preferred bit complicates handling of alternative locators and a host should prefer sending its locators in the base exchange with all preferred bits unset.

When a host receives locators with all preferred bits unset, they should be considered as alternative addresses for the peer. The host does not have to use these locators immediately, but can use them for fault tolerance or load balancing purposes. This aids also cross-family handovers because then two communicating hosts know all the available addresses of each other.

At the Responder side, the LOCATOR parameter could be placed into the R2 packet instead of the R1. The LOCATOR in the R2 packet facilitates mobile devices to serve as Responders better. For instance, a mobile node could disable an expensive link until the base exchange completes. Also, this is beneficial for a mobile node employing precreated pools of R1 packets. As the R1 signature covers the Responder's IP-address, it does not have to recreate its pools upon address changes.

Using the LOCATOR parameter in the base exchange benefits also HIP NAT traversal [71], which forbids preferred bits in NATted environments and assumes the LOCATOR to be placed in the R2 packet. De la Oliva et al. [26] also proposed a scheme for sending all the locators early in the communication in order to maximize the fault tolerance.

6.3.3 Peer Locator Learning

It is possible for a host to delay the exposure of additional locators to host's peer for, e.g., privacy reasons to avoid exposing of the topology of the corporation of the end-host. Alternatively, the end-host can even be unaware of some of its locators in NATted environments [71] where the peers of the end-host observe the address of a NAT middlebox and not actual end-host address. In either case, a correspondent node should be able to inform about its additional locators after the base exchange without sending additional locators. As an example, let us consider that two hosts have established the base exchange over IPv6 without exchanging additional locators. Then, one of the hosts becomes mobile and moves to an IPv4-only network. The mobile node informs its correspondent node about its new location with an UPDATE. Now, the correspondent node can choose to break connectivity for privacy reasons or send an echo request from its previously unadvertised IPv4 address.

The use of unadvertized addresses is not defined in the HIP mobility specification [93]. To achieve better flexibility, we propose that correspondent node should be able to send echo requests from previously unadvertised addresses and the mobile node should reply to them with echo responses. We refer this as *peer locator learning.*

As a second example of scenario, NAT middleboxes alter source addresses of UDP encapsulated HIP packets and the end-host sending the packets may be unaware of this. As a consequence, the packet receiver learns a new address of the originating host that was not advertised in the included LOCATOR parameter.

Peer locator learning is depicted in Figure 6.1. In step 1 the MN changes its attachment point to the network and obtains one IPv4 address and one IPv6 address after which in step 2 MN sends the new locator set to its CN. Upon receiving the locator set, the CN starts the return routability tests and sends one echo request to the IPv4 address and one echo request to the IPv6 address. When the MN receives the echo request from the CN's IPv4 address, it checks the locator lists it has for the active CNs and sees that the locator is already known and sends an echo response to the CN (see Figure 6.1 step 3). Upon receiving the echo request from its IPv6 locator, the MN checks its locator lists for the active CNs and does not find the used locator (see Figure 6.1 step 4). Finally, MN adds this locator to the list and starts connectivity tests for the locator, and sends an echo response to the CN (see Figure 6.1 step 5).

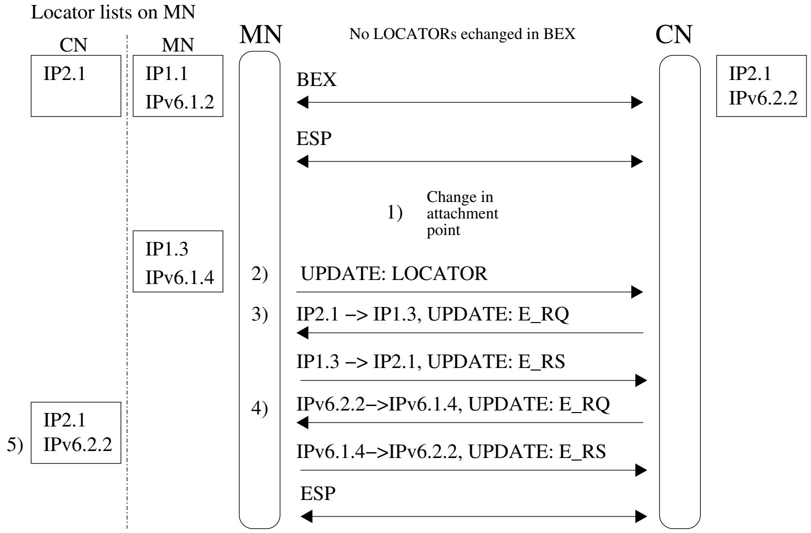

Locator lists on MN
CN
MN
IP2.1
IP1.1
IPv6.1.2
MN
No LOCATORs echanged in BEX
CN
IP2.1
IPv6.2.2
BEX
ESP
1)
Change in attachment point
IP1.3
IPv6.1.4
2)
UPDATE: LOCATOR
3)
IP2.1 –> IP1.3, UPDATE: E_RQ
IP1.3 –> IP2.1, UPDATE: E_RS
4)
IPv6.2.2–>IPv6.1.4, UPDATE: E_RQ
IPv6.1.4–>IPv6.2.2, UPDATE: E_RS
5)
IP2.1
IPv6.2.2
ESP

Figure 6.1: Peer locator learning example case, where the end-hosts exchange no LOCATORs in the BEX and the MN learns the IPv6 address of its CN from the return routability tests

Peer locator learning is also beneficial in cases such as simultaneous end-host mobility. In this case both end-hosts move simultaneously and lose connectivity due to the fact that the end-hosts do not know where to send the UPDATE control packet. This is generally solved with third party rendezvous service as described by Hobaya et al. [46].

In their paper, Hobaya et al. also describe a situation where the simultaneous end-host mobility ends up in confusion when the UPDATE procedures on the end-hosts get interleaved and leaves the connection in an asymmetric state. Hobaya et al. provide also a solution for the problem by enforcing UPDATE retransmissions. In their examples one of the end-hosts is able to send data to the other, but not vice versa, because the other end-host mistakenly cleared its retransmission buffers. With peer locator learning, the end-host unable to the send data could find a new locator from IP header of the received UPDATE control packet. After the end-host has discovered the new address, it could trigger the return routability tests and, as a result, both of the nodes could continue communicating. We believe that the peer locator learning technique would result in faster handovers.

6.3.4 Teredo Experiments

We wanted to validate that our implementation works in the presence of NATs with Teredo. In general, basic Teredo-based connectivity was successful in our experimentation. We discovered some problems as well, for example, when the mobile node moved into an IPv6-only network and could not derive a Teredo address in the absence of an IPv4 address. Another problem was that the mobile node sent an UPDATE packet to the Teredo address of the correspondent node, but the local router did not know what to do with the non-routable Teredo address. In order to work, this case would have required a Teredo relay in the network of the mobile node or a global IPv6 address for the corresponding node.

Miredo, the Teredo implementation for Linux, decreased the throughput due to the tunneling overhead and unoptimized implementation. Especially in MBB handovers, it took 30 seconds at the maximum for the Miredo software to notice a mobility event that required changing the topology-dependent Teredo address. HIP daemon reacted instantly by sending an UPDATE packet advertising the old but unfortunately already invalid Teredo address. As a summary, there is room for performance improvements in the Miredo implementation.

6.3.5 Implementation of Cross-Family Handovers

This section discusses some of the issues we faced when implementing cross-family handovers with HIP. We chose the HIPL implementation [105] as our experimentation tool. Most of the changes involved only the UPDATE packet parameters. Only minor changes were required in the processing of R1 and I2 packets.

To make cross-family handovers possible, we implemented a new function to uniformly build LOCATOR parameters containing all the locators of the local host. Modifications also included introducing of LOCATOR parameter to R1 and I2 control packets.

The challenges we faced ranged from trivial to more complicated. A trivial problem was that base exchanges with locators triggered return routability tests before the state was ESTABLISHED on both sides. As a solution, we had to delay the triggering of address verifications (ECHO_-REQUESTS) to avoid unnecessary dropping of UPDATE control packets. A trickier problem originated from the sockets API that has separate raw sockets for IPv4 and IPv6. Raw sockets are needed for sending and receiving of HIP control packets. We experienced a problem where one of the socket buffers contained a base exchange packet and the other sockets buffer contained an UPDATE packet. We had to change the implementation to handle the packets in the correct order. This was just an optimization to the handovers because the problem could also be solved by just dropping the UPDATE packet and relying on retransmissions of the mobile node.

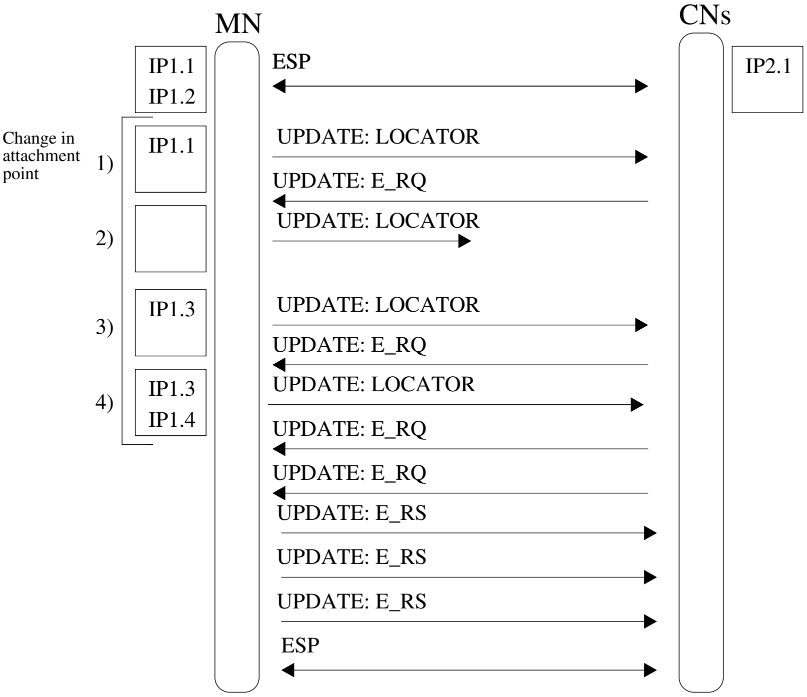


Figure 6.2: Results of triggering the handover too fast after a change in the addresses on a interface.

Performance measurements described in the Section 6.4 revealed features in HIPL that were too aggressive in their behavior and did not conform to the specifications.

First, the MN triggered the first UPDATE control packet immediately after a obtaining a new address. This resulted in an ICMP message informing the MN that the CN is unreachable because the network interface was not fully initialized. As a result from the destination unreachable error, the implementation queued the UPDATE control packet to the retransmission queue for ten seconds. It would have been beneficial to start with a low interval for the retransmission and increase it exponentially as described in the specification [92]. This is also discussed by Shütz et al. [124].

Second, relying to the address notifications from the kernel, i.e., netlink events, as the only indication for the handover and reacting too aggressively to it resulted in excess UPDATE control packets. This situation is depicted in Figure 6.2. For example, in a case where the MN has two addresses, reacting instantly on netlink messages would result into multiple UPDATE control packets unnecessarily. When the interface goes down, the addresses are deleted one by one and this results in two UPDATE control packets, one with LOCATOR containing one locator (see Figure 6.2 step 1) and a second, a so called zero address readdressing packet (see Figure 6.2 step 2). As a response to this, the CN tests at least the address from which the UPDATE packet was received for return routability. There is no guarantees that zero address readdressing packet will be sent. When the CN obtains the new attachment point and its interfaces are brought up the netlink informs about the new addresses (two in the example), the kernel informs about the addresses one at a time that will result in two more UPDATE control packets (see Figure 6.2 steps 3 and 4). At the CN side, this will unoptimally result in at least three more return routability tests despite that the address seen in the previous control packet would not have to be tested again.

To sum up, the second case leads to three update packets sent to the CN and the CN sends four return routability tests, while we could manage with one UPDATE control packet and trigger return routability tests for the addresses found in the locator set contained in the control packet. As a simple solution for this, we delayed the handover so that all the consequtive netlink events could be handled as a single event. We have not yet optimized the modified solution to its maximum.

This is not the best solution because it increases the latency of the handover in overall, while in the second case described in this Section it decreases the handover latency. In our opinion the trigger for the handovers should not be a simple solution that relies on one type of data. For example the netlink messages could be augmented by monitoring of the end-to-end connectivity.

Figure 6.3 depicts the growth of the TCP sequence number during a BBM handover. The base exchange is concluded at point of time T1. MN loses network connectivity at T2 and regains it at T3. At point of time T4 the interfaces are fully operational and HIP update procedure is triggered. We observed that, after the update procedure at T4, there is some extra latency before the first IPsec ESP packet is sent at T5.

This latency varied so that it may even out the difference in the overall latency presented in Section 6.4 in Tables 6.1 and 6.2. This finding conforms to the findings of Shütz et al. [124] where they found a similar period of inactivity after a period of disconnectivity. According to them, TCP waits for the current retransmission timeout to expire while the new address is obtained or the connectivity is otherwise restored before TCP tries to retransmit. Shütz et al. [124] suggest an improvement to this situation. Their solution also tries to minimize the period of inactivity by introducing a more aggressive way to enforce the retransmissions after an end-host receives or sends the last echo response in the update procedure. In our opinion, this feature is a welcome improvement to decrease the handover latency.

To improve the chances that transport layer survives connectivity loss automatically, we implemented a heartbeat probe. The heartbeat is used to monitor the connectivity between the hosts. The heartbeat is an ICMPv6 messages inside the ESP tunnel between two end-hosts. As a naive approximation, the implementation triggers the update procedure after n consecutive heartbeats are lost. Care has to be taken to avoid choosing a too long interval for the heartbeat to avoid TCP aborting the connection. Also, intermediary hosts, such as NAT boxes, may time out an idle ESP tunnel when the heart beat interval is too long.

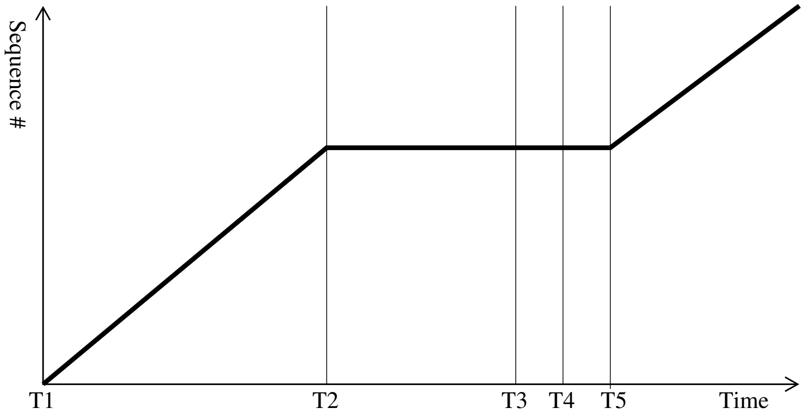


Figure 6.3: Sequence number generation during BBM handover.

For the heartbeats to work the end-host needs to know more than one address of its peer. An advantage in using Internet Control Message Protocol version 6 (ICMPv6) packets to implement the heartbeat is the fact that the heartbeat mechanism must be supported only on the end-host using the heartbeat. The end-host on the other side of the tunnel does not have to trigger heartbeats at all. It merely has to has to support replying to

ICMPv6 messages inside the tunnel.

The implementation currently sends the heartbeat on regular intervals. It could be optimized to send only when the IPsec tunnel is idle. However, sending heartbeats all the time and gathering the monitoring results from them is a better choice for multihoming cases as explained by Gurtov et al. [38].

6.4 Performance Measurements

In this section, we describe the measured impact of cross-family handovers. To avoid issues with TCP timeouts documented in detail elsewhere [124], we primarily measured UDP throughput.

We conducted the measurements on two identical laptops (Intel Core 2, 2 GHz CPU). We concentrated on the processing cost by minimizing the network latency (RTT 0.484 ± 0.143 ms), and therefore the laptops were connected via single Gigabit switch to each other. Both machines were running Ubuntu Jaunty Jackalope Linux with 2.6.28 kernel and HIPL release 1.0.3.

We triggered handovers using *ip* command from ip-tools package that allows manipulation of the network interfaces. In the test cases the CN sent UDP packets continuously to the MN. We used Wireshark to capture the traffic on the MN and to analyze the gathered data.The handover was measured to begin from the sending of the first UPDATE control packet with LOCATOR parameter (step 2 in Figure 2.12) and to cease when the first ESP is received using the new address (step 8 in Figure 2.12).

Tables 6.1 and 6.2 show that cross-family MBB and BBM handovers tend to last 8 milliseconds longer than handovers where the family does not change.

Table 6.1: Durations of intra-family handovers.

Direction	Duration, ms
MBB IPv4 to IPv4	53 ± 12
MBB IPv6 to IPv6	56 ± 6
BBM IPv4 to IPv4	41 ± 12
BBM IPv6 to IPv6	40 ± 6
Total average	47 ± 10

We observed a delay of 10 ms (±1) from sending the UPDATE control packet with LOCATOR parameter and receiving of the UPDATE control packet with ECHO_REQUEST (steps 2 - 4 in Figure 2.12). Handling of

Table 6.2: Durations of cross-family handovers.

Direction	Duration, ms
MBB IPv4 to IPv6	56 ± 6
MBB IPv6 to IPv4	53 ± 16
BBM IPv4 to IPv6	56 ± 8
BBM IPv6 to IPv4	54 ± 11
Total average	55 ± 11

the ECHO_REQUEST and creation of needed SAs took 19 ms (±5) in intra-family handovers and 40 ms (±8) in cross-family handovers (step 5 in Figure 2.12). The delay between sending of the ECHO_RESPONSE and the receiving of the first ESP packet (steps 6-8 in Figure 2.12) was 6 ms (±2). Most of the processing time was spent in processing of the UPDATE control packet with ECHO_REQUEST parameter as Pääkkönen et al. [100] have also observed. We suspect that the processing time was doubled in cross-family handovers due to unoptimized code.

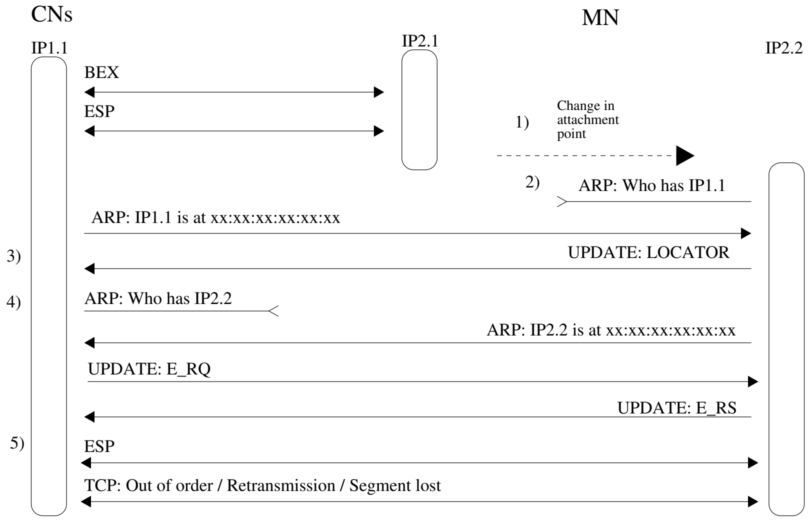

CNs
MN
IP1.1
IP2.1
IP2.2
BEX
ESP
1)
Change in attachment point
2)
ARP: Who has IP1.1
ARP: IP1.1 is at xx:xx:xx:xx:xx:xx
3)
UPDATE: LOCATOR
4)
ARP: Who has IP2.2
ARP: IP2.2 is at xx:xx:xx:xx:xx:xx
UPDATE: E_RQ
UPDATE: E_RS
5)
ESP
TCP: Out of order / Retransmission / Segment lost

Figure 6.4: An intra-family Break Before Make handover from IPv4 to IPv4 with ARP traffic before and after the handover.

In Figures 6.4 and 6.5, we depict the difference of moving from IPv4 to IPv4 and from IPv4 to IPv6 in the BBM handover case. The mobile node changes the attachment point and obtains a new locator (see Figure 6.4 step 1 and Figure 6.5 step 1). The major difference is the use of Address Resolution Protocol (ARP) messaging in IPv4 (see Figure 6.4 steps 2 and

4) versus ICMPv6 neighbor discovery (see Figure 6.5 step 2) in IPv6.

Operationally, ARP and ICMPv6 neighbor discovery do not differ much. In ARP, the end-host broadcasts a “Who has” message, containing the target IP-address to the network. The end-host possessing the address answers that the queried IP-address is at the specified link-local address. In IPv6 neighbor discovery, the end-host first announces to the nearest router that it listens to IPv6 broadcasts and excludes its own address from the broadcasts it wants to receive. Then, the end-host broadcasts a neighbor solicitation message asking who has the target IPv6 address. The end-host possessing the IPv6 address answers with neighbor advertisement containing the end-host’s link-local address.

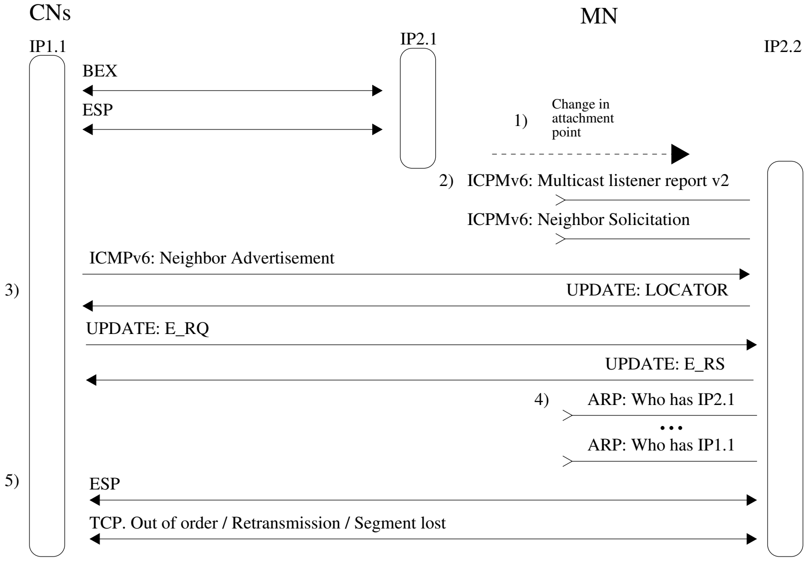

CNs
MN
IP1.1
IP2.1
IP2.2
BEX
ESP
1)
Change in attachment point
2)
ICPMv6: Multicast listener report v2
ICPMv6: Neighbor Solicitation
ICMPv6: Neighbor Advertisement
3)
UPDATE: LOCATOR
UPDATE: E_RQ
UPDATE: E_RS
4)
ARP: Who has IP2.1
•••
ARP: Who has IP1.1
5)
ESP
TCP. Out of order / Retransmission / Segment lost

Figure 6.5: Cross-family Break Before Make handover from IPv4 to IPv6 with the ICMPv6 neighbor discovery traffic and with ARP messages for the lost IPv4 connectivity.

In intra-family case with IPv4, the peer locator was discovered before and after the UPDATE procedure (see Figure 6.4 step 3). Intra-family handover with IPv6 or cross-family handover towards IPv6 did not incur neighbor discovery after the UPDATE procedure (see Figure 6.5 step 3). We observed that in cross-family handovers, from IPv4 to IPv6, the interface kept broadcasting ARP queries about the previously used IPv4 addresses while it did not have a working IPv4 address anymore. In our tests this behavior resulted in circa five broadcasted messages (see Figure 6.5 step 4), after which the ESP traffic started flowing again (see Figure 6.5

step 5). After the handover, TCP had to retransmit some of the data and some of it was received out of order. In overall, the amount of TCP retransmits did not differ in intra-family or cross-family handovers. The similar amounts of retransmitted TCP segments can be also explained. TCP handles the retransmissions on top of the IPsec ESP and TCP is not affected by the change of the address family, i.e., TCP is connected to a HIT and the handover is transparent to TCP. We also observed the Credit Based Authorization (CBA) (see [93] Section 3.3.1) operational in some intra-family cases. CBA allows IPsec ESP traffic to commence before the completion of return routability tests. We also observed that the CBA was too aggressive and resulted in ICMP reroute messages. We also performed the same tests with UDP which did not show any significant difference in the amount of lost packets.

6.5 Summary

Cross-family handovers can be used as a transition mechanism towards IPv6 now that IPv4 address space is almost depleted. In this chapter, we have shown three key contributions. 1) We described a shortcoming in current HIP mobility specifications preventing cross-family handovers and suggested a simple solution to it. 2) Our performance evaluation on our implementation indicates that HIP-based cross-family handovers perform as well as intra-family handovers. 3) Our approach is compatible with NATted networks because it can make use of Teredo-based end-to-end tunnels.

Chapter 7

Conclusion

The IP was designed in an age when the nodes in the Internet were stationary and few, the closest thing to mobility involved a truck. In the current day things have changed: every day more and more equipment needs an Internet connection and the equipment is very portable. For the Internet this means more addresses and IPv4 cannot handle the ever growing number of equipment needing connections. IPv6 was introduced to alleviate the shortage of the addresses. However, IPv6 has not been completely deployed and in the current day Internet is in transition state, parts of the Internet use IPv4, parts IPv6, and some parts use both families. Moreover, the IP-address of the machine is often seen as the identity of the machine, which in MN case is not that reliable, as the address may change according to the attachment point in the network. As a solution the identifier-locator split protocols were introduced to separate the meaning of the identifier from the locator, i.e., the IP-address. However, the introduction of the identifier-locator protocols brought along new problems: will the legacy application work, how to securely resolve the new identifiers with fast user-based updates, how to control access in home environments, and how to maintain the connection in the transition state Internet.

In this thesis we addressed the new problems that were not addressed in the specification of the current day identifier-locator protocols for the secure connectivity of MNs in the transition state Internet. Moreover, we concentrated on solving the problems with a host-based mapping and encapsulating based Identifier-Locator protocol, i.e., the HIP.

7.1 Summary of Contributions

We gathered extensive statistics from a modern OS, i.e., Ubuntu Linux, and from all of its LTS versions. From these statistics we characterized the OS's usage of the Berkeley Sockets API. We concentrated on five fundamental questions: how the end-hosts are named, name resolution, multiple end-host identifiers, transport protocol selection, and selection of security solutions. We reported ten interesting findings that included security, IPv6, and configuration related issues. For example, we found out that only 10.4% of the SSL/TLS capable software failed to provide adequate error handling. Moreover, we found out that 28.6% of the SSL/TLS capable failed to initialize the OpenSSL library incorrectly. Out of all of the SSL/TLS capable only 58.4% if the applications were seeding the PRNG before using it. This is surprising as the incorrect seeding of the PRNG is considered as a common security pitfall. From the statistics we showed that the UDP multihoming problem persists, even in the frameworks for network applications. The UDP multihoming problem manifests itself when a client uses the non-default address of the server to send the UDP message and the server answers with its default address, and so to the client the response seems to come from different entity, i.e., from different IP-address. Based on the findings we concluded that the Sockets API usage is heterogeneous and that it is difficult to introduce general modifications to the way applications utilize networking features. We partly addressed the extent of this challenge by suggesting fixes on security and UDP multihoming support.

Based on our observations on resolution systems, we described an architecture for secure identifier-locator mappings based on a distributed hash table. In particular, we discussed three core problems of name resolution for host-based identity locator split protocols: a) support for flat namespaces, b) rapid user-generated updates, and c) the security of the mappings. We address these problems by implementing secure key updates based on the cryptographic properties of the identifiers in the HIP. Our system works with user-generated identities and does not require any user management or the deployment of a global PKI system because it makes use of the self-certifying identities in HIP. With its identity concept and IPv6 compatibility, HIP integrates nicely into existing lookup systems and enhances their security features with DoS resilience and authenticated locator updates. Our performance analysis of the HIP-enabled DHT API demonstrates the feasibility of the architecture and indicates that employing HIP as a security solution provides acceptable performance with considerably increased security.

We designed a DHT-based system to distribute the information of friendships that is based on the self-certifying cryptographic identities of HIP. We create one-hop trust paths to be used to access control the incoming calls. We rather detect friends and friends-of-friends than spit or spitters. There are less friends than spitters, so the task is far simpler. The solution we described can be generalized for situations where the before-hand inspection of the content is impossible or otherwise hard to implement, for example middleboxes may not have the storage to delay large transfers.

It has been shown that identifier-locator split protocols can solve two major challenges that the Internet architecture is facing. First, end-to-end connectivity is broken due to NATs. Second, the Internet architecture lacks a mechanism that supports end-host mobility in the transition state Internet, i.e., cross-family handovers. We demonstrated that cross-family handovers can be used as an IPv6 transition mechanism now that the IPv4 address space is almost depleted. We described a shortcoming in current HIP mobility specifications preventing cross-family handovers and suggested a simple solution to it. Our performance evaluation with our implementation indicates that HIP-based cross-family handovers perform as well as intra-family handovers. Our approach is compatible with NATted networks because it can make use of Teredo-based end-to-end tunnels.

7.2 Future Work

In Chapter 3, our work covers only C-based software. It would be useful to examine also software written in other languages. However, we believe that it was important to inspect C-based software first because the bindings to the Sockets API in other languages are essentially written in C.

Our analysis covers only Ubuntu Linux. We believe that it would be useful to inspect also Windows applications by extending the analysis to binary executables.

We did not profile the use of the customization of the networking stack. This would be useful especially for application frameworks or even for providing new extensions for the sockets APIs.

Our statistics are very coarse grained and can be improved with further work. While we have confirmed the need for synchronous decoupling in pub-sub designs, the time and space decoupling arguments should be ratified with a more fine-grained source-code analyzer. Deepening the analysis on the (un)marshalling process of the application protocols could be used to further improve network application frameworks or to affirm the message-oriented design of pub-sub systems. Similarly our findings on SSL/TLS issues could be further analyzed with software analysis methods proposed by others [25, 7].

Based on the applications in Ubuntu Linux, we found the performance of the current Sockets API sufficient for most applications because *mmap()* was used only little. This assumption does not apply to esoteric environments, such as High-Performance Computing (HPC), sensor and cloud network as they have their own application bases. Nevertheless, we feel it would useful to extend the scope of the analysis to such environments.

In Chapter 6 we identified a problem in triggering of the handovers and as future work we intend to research solutions for triggering the handovers and find an optimal solution for the triggers.

Based on the experiences with the CERT-parameter in HIP we started an Internet Draft [43] that we plan to improve. The draft describes an extension for HIP that enables HIP end-hosts and HIP-aware middleboxes to announce services to HIP hosts during a BEX or HIP update.

References

[1] Understanding universal plug and play. whitepaper, Oct. 2007. Microsoft.

[2] A. Abdul-Rahman. The PGP Trust Model. In *EDI-Forum: the Journal of Electronic Commerce*, volume 10, pages 27–31, 1997.

[3] J. Adams. RFC 2144: The cast-128 encryption algorithm. Request for Comments 2144, Internet Engineering Task Force, Apr. 1997.

[4] B. Ahlgren, J. Arkko, L. Eggert, and J. Rajahalme. A node identity internetworking architecture. In *Proc. of INFOCOM 2006. 25th IEEE International Conference on Computer Communications.*, Apr. 2006.

[5] J. Ahrenholz. RFC 6537: Host Identity Protocol Distributed Hash Table Interface. Request for Comments 6537, Internet Engineering Task Force, Feb. 2012.

[6] P. Albitz and C. Liu. *DNS and BIND, Fourth Edition.* O'Reilly and Associates, 2001.

[7] G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '02, pages 4–16, New York, NY, USA, 2002. ACM.

[8] M. Andrews. RFC 2308: Negative Caching of DNS Queries (DNS NCACHE). Request for Comments 2308, Internet Engineering Task Force, Mar. 1998.

[9] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4033: DNS Security Introduction and Requirements. Request for Comments 4033, Internet Engineering Task Force, Mar. 2005.

[10] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4034: Resource Records for the DNS Security Extensions. Technical Report 4034, Internet Engineering Task Force, March 2005.

[11] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4035: Protocol Modifications for the DNS Security Extensions. Technical Report 4035, Internet Engineering Task Force, March 2005. Updated by RFC 4470.

[12] J. Arkko and A. Keränen. RFC 6586: Experiences from an IPv6-Only Network. Request for Comments 6586, Internet Engineering Task Force, Apr. 2012.

[13] D. Atkins and R. Austein. RFC 3833: Threat Analysis of the Domain Name System (DNS). Request for Comments 3833, Internet Engineering Task Force, Aug. 2004.

[14] R. Atkinson. ILNP Concept of Operations: draft-rja-ilnp-intro-11. Internet draft, Internet Engineering Task Force, July 2011. Expired.

[15] I. Baumgart. P2PNS: A Secure Distributed Name Service for P2PSIP. In *Proc. of 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, 2008.

[16] S. M. Bellovin. Problem areas for the ip security protocols. In *Proceedings of the Sixth Usenix Unix Security Symposium*, pages 205–214. AT and T Research, USENIX, July 1996.

[17] G. Camarillo. *A Service-enabling Framework for the Session Initiation Protocol (SIP).* PhD thesis, Aalto University, Espoo, 2011.

[18] G. Camarillo, P. Nikander, J. Hautakorpi, and A. Johnston. RFC 6079: HIP BONE: Host Identity Protocol (HIP) Based Overlay Networking Environment (BONE). Request for Comments 6079, Internet Engineering Task Force, Jan. 2011.

[19] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International workshop On Designing Privact Enhancing Technologies: Design Issues In Anonymity And Unobservability*, pages 46–66. Springer-Verlag New York, Inc., 2001.

[20] B. Cohen. The BitTorrent Protocol Specification, Version 11031., 2008. `http://bittorrent.org/beps/bep_0003.html`, 22.9.2012.

[21] U. D. O. Commerce. Des modes of operation. FIPS 81, U.S. Department Of Commerce, National Institute of Standards and Technology, Dec. 1980. Federal Information Processing Standards Publication, Category: Computer Security Subgategory: Cryptography.

[22] U. D. O. Commerce. Data encryption standard. FIPS 463, U.S. Department Of Commerce, National Institute of Standards and Technology, Oct. 1999. Federal Information Processing Standards Publication, Category: Computer Security Subgategory: Cryptography.

[23] D. Cooper, S. Santesson, S. Farrel, S. Boeyen, R. Housley, and W. Polk. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Request for Comments 5280, Internet Engineering Task Force, May 2008.

[24] R. Cox, A. Muthitacharoen, and R. Morris. Dns performance and the effectiveness of caching. In *In IEEE/ACM Transactions on Networking*, volume 2429, pages 155 – 165. Springer-Verlag, Mar. 2002.

[25] P. de la Cámara, M. M. Gallardo, P. Merino, and D. Sanán. Model checking software with well-defined apis: the socket case. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, FMICS '05, pages 17–26, New York, NY, USA, 2005. ACM.

[26] A. de la Oliva and M. Bagnulo. Fault tolerance configurations for HIP multihoming: draft-oliva-hiprg-reap4hip-00. Internet draft, Internet Engineering Task Force, July 2007. Expired.

[27] V. Devarapalli and P. Eronen. RFC 5266: Secure Connectivity and Mobility Using Mobile IPv4 and IKEv2 Mobility and Multihoming (MOBIKE). Request for Comments 5266, Internet Engineering Task Force, June 2008.

[28] drscholl. Napster Messages, 2000. `http://opennap.sourceforge.net/napster.txt`, 27.11.2011.

[29] D. Eastlake. RFC 2535: Domain Name System Security Extensions. Request for Comments 2535, Internet Engineering Task Force, Mar. 1999.

[30] D. Eastlake. RFC 4305: Cryptographic Algorithm Implementation Requirements for Encapsulationg Security Payload (ESP) and Authentication Header (AH). Request for Comments 4305, Internet Engineering Task Force, Dec. 2005.

[31] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. RFC 2693: SPKI Certificate Theory. Request for Comments 2693, Internet Engineering Task Force, Sept. 1999.

[32] P. Eronen. RFC 4555: IKEv2 Mobility and Multihoming Protocol (MOBIKE). Request for Comments 4555, Internet Engineering Task Force, June 2006.

[33] D. Farinacci, V. Fuller, D. Lewis, and D. Meyer. LISP Mobility Architecture: draft-meyer-lisp-mn-07. Internet draft, Internet Engineering Task Force, Apr. 2012. Work in progress.

[34] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID Separation Protocol (LISP): draft-ietf-lisp-23.txt. Internet draft, Internet Engineering Task Force, May 2012. Work in progress.

[35] R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens. RFC 3439: Basic Socket Interface Extensions for IPv6. Request for Comments 3493, Internet Engineering Task Force, Feb. 2003.

[36] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *In 1st USENIX/ACM Symposium on networked systems design and implementation (NSDI '04), San Francisco, CA, 2004*, 2004.

[37] A. Gurtov. *Host Identity Protocol (HIP): Towards the Secure Mobile Internet*. Wiley and Sons, 2008.

[38] A. Gurtov and T. Polishchuk. Secure Multipath Transport for Legacy Internet Applications. In *Proceedings of BROADNETS'09*, Sep 2009.

[39] D. Harkins and D. Carrel. RFC 2409: The internet key exchange (IKE). Request for Comments 2409, Internet Engineering Task Force, Nov. 1998.

[40] T. Heer. *Direct End-to-Middle Authentication in Cooperative Networks*. PhD thesis, Rheinisch-Westfaelische Technische Hochschule Aachen, 2011.

[41] T. Heer, R. Hummen, K. Wehrle, and M. Komu. End-host authentication for hip middleboxes: draft-heer-hip-middle-auth-04. Internet draft, Internet Engineering Task Force, Oct. 2011. Expired.

[42] T. Heer and S. Varjonen. RFC 6253: HIP Certificates. Request for comments, Internet Engineering Task Force, Aug. 2011.

[43] T. Heer, H. Wirtz, and S. Varjonen. Service identifiers for hip: draft-heer-hip-service-01. Internet draft, Internet Engineering Task Force, Sept. 2011. Expired.

[44] J. Heikkilä and A. Gurtov. Filtering SPAM in P2PSIP communities with web of trust. In *Proceedings of the MobiSec'09*, Jun 2009.

[45] R. Hinden and S. Deering. RFC 4291: IP Version 6 Addressing Architecture. Request for Comments 4291, Internet Engineering Task Force, Feb. 2006.

[46] F. Hobaya, V. Gay, and E. Robert. Host Identity Protocol extension supporting end-host simultaneous mobility. In *Proceedings of 2009 Fifth International Conference on Wireless and Mobile Communications*, pages 261–266, 2009.

[47] R. Housley. RFC 4309: Using advanced encryption standard (AES) CCM mode with IPsec encapsulating security payload (ESP). Request for Comments 4309, Internet Engineering Task Force, Dec. 2005.

[48] R. Housley, W. Ford, T. Polk, and D. Solo. RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile. Request for Comments 2459, Internet Engineering Task Force, Jan. 1999.

[49] R. Housley, W. Polk, W. Ford, and D. Solo. RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Request for Comments 3280, Internet Engineering Task Force, Apr. 2002.

[50] C. Huitema. RFC 4380: Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). Request for Comments 4380, Internet Engineering Task Force, Feb. 2006.

[51] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. Resource location and discovery (reload) base protocol: draft-ietf-p2psip-base-22. Internet draft, Internet Engineering Task Force, July 2012. Work in progress.

[52] D. Johnson, C. Perkins, and J. Arkko. RFC 3775: Mobility Support in IPv6. Request for Comments 3775, Internet Engineering Task Force, June 2004.

[53] P. Jokela, R. Moskowitz, and P. Nikander. RFC 5202: Using ESP Transport format with HIP. Request for Comments 5202, Internet Engineering Task Force, Apr. 2008.

[54] P. Jokela, P. Nikander, J. Melen, J. Ylitalo, and J. Wall. Host Identity Protocol: Achieving IPv4 - IPv6 handovers without tunneling. In *Proceedings of Evolute workshop 2003: Beyond 3G Evolution of Systems and Services*, University of Surrey, Guildford, UK, Nov 2003.

[55] P. Jokela, T. Rinta-Aho, T. Jokikyyny, J. Wall, M. Kuparinen, J. Melén, T. Kauppinen, and J. Korhonen. Handover performance with HIP and MIPv6. In *1st International Symposium on Wireless Communication Systems*, pages 324 – 328, 2004.

[56] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. Dns performance and the effectiveness of caching. In *In IEEE/ACM Transactions on Networking*, volume 10, pages 589 – 603. ACM Press, Oct. 2002.

[57] E. K. Zeilenga. RFC 4514: Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names. Request for Comments 4514, Internet Engineering Task Force, June 2006.

[58] G. Kan. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* O'reilly, 2001. Chapter 8: Gnutella.

[59] C. Kaufman. Recommendation X.690 Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). ITU-T X.690-0207, International Telecommunication Union, Dec. 2002.

[60] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet key exchange (ikev2) protocol. Request for Comments 5996, Internet Engineering Task Force, Sept. 2010.

[61] S. Kent. RFC 4301: Security Architecture for the Internet Protocol. Request for Comments 4301, Internet Engineering Task Force, Dec. 2005.

[62] S. Kent. RFC 4302: IP Aunthentication Header. Request for Comments 4302, Internet Engineering Task Force, Dec. 2005.

[63] S. Kent. RFC 4303: IP Encapsulating Security Payload (ESP). Request for Comments 4303, Internet Engineering Task Force, Dec. 2005.

[64] S. Kent and R. Atkinson. RFC 2401: Security Architecture for the Internet Architecture. Request for Comments 2401, Internet Engineering Task Force, Nov. 1998.

[65] S. Kent and R. Atkinson. RFC 2402: IP authentication header. Request for Comments 2402, Internet Engineering Task Force, Nov. 1998.

[66] S. Kent and R. Atkinson. RFC 2406: IP encapsulating security payload (ESP). Request for Comments 2406, Internet Engineering Task Force, Nov. 1998.

[67] A. Khurri, E. Vorobyeva, and A. Gurtov. Performance of Host Identity Protocol on lightweight hardware. In *MobiArch '07: Proceedings of the 2nd ACM/IEEE International Workshop on Mobility in the Evolving Internet Architecture*, pages 1–8, New York, NY, USA, Aug. 2007. ACM.

[68] J. Kohl and C. Neuman. RFC 1510: The Kerberos Network Authentication Service (V5). Request for Comments 1510, Internet Engineering Task Force, Sept. 1993.

[69] M. Komu, M. Bagnulo, S. Sugimoto, and K. Slavov. RFC 6316: Socket Application Program Interface (API) for Multihoming Shim. Request for Comments 6316, Internet Engineering Task Force, July 2011.

[70] M. Komu and T. Henderson. RFC 6317: Basic Socket Interface Extensions for Host Identity Protocol (HIP). Request for Comments 6317, Internet Engineering Task Force, jul 2011.

[71] M. Komu, T. Henderson, H. Tschofenig, J. Melen, and A. Keranen. RFC 5770: Basic Host Identity Protocol (HIP) Extensions for Traversal of Network Address Translators. Request for Comments 5770, Internet Engineering Task Force, Apr. 2010.

[72] M. Komu and J. Lindqvist. Leap-of-Faith Security is Enough for IP Mobility. In *Proceedings of the 6th Annual IEEE Consumer Communications and Networking Conference IEEE CCNC 2009*, Las Vegas, NV, Jan 2009.

[73] M. Komu, S. Varjonen, S. Tarkoma, and A. Gurtov. Sockets and Beyond: Assessing the Code of Network Applications. In *Aalto University publication series SCIENCE + TECHNOLOGY*, volume 46, 2011. ISSN 1799-490X (pdf).

[74] J. Korhonen. *IP Mobility in Wireless Operator Networks*. PhD thesis, University of Helsinki, 2008.

[75] H. Krawczyk. SKEME: A versatile secure key exchange mechanism for the Internet. In *Proceedings of the 1996 Internet Society Symposium on Network and Distributed System Security*, pages 114–127. IEEE Computer Society, Feb. 1996.

[76] J. Laganier and L. Eggert. RFC 5204: Host Identity Protocol (HIP) Rendezvous Extension. Request for Comments 5204, Internet Engineering Task Force, Apr. 2008.

[77] J. Laganier, T. Koponen, and L. Eggert. RFC 5203: Host Identity Protocol (HIP) Registration Extension. Request for Comments 5203, Internet Engineering Task Force, Apr. 2008.

[78] X. Lai. Detailed description and a software implementation of the ipes cipher. Technical report, Institute for Signal and Information Processing, ETH-Zentrum, 1991.

[79] B. Laurie, G. Sisson, R. Arends, and D. Blacka. RFC 5155: DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. Request for Comments 5155, Internet Engineering Task Force, Mar. 2008.

[80] J. Liang, N. Naoumov, and K. Ross. The index poisoning attack in p2p file sharing systems. In *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pages 1 – 12. IEEE, Apr. 2006.

[81] H. Luo, Y. Qin, and H. Zhang. A DHT-Based Identifier-to-Locator Mapping Approach for a Scalable Internet. In *IEEE Transactions on Parallel and Distributed Systems*. IEEE Computer Society, Feb. 2009.

[82] C. Madson and N. Doraswamy. RFC 2405: The EPS DES-CBC cipher algorithm with explicit IV. Request for Comments 2405, Internet Engineering Task Force, Nov. 1998.

[83] C. Madson and R. Glenn. RFC 2403: The use of HMAC-MD5-96 within ESP and AH. Request for Comments 2403, Internet Engineering Task Force, Nov. 1998.

[84] C. Madson and R. Glenn. RFC 2404: The use of HMAC-SHA-1-96 within ESP and AH. Request for Comments 2404, Internet Engineering Task Force, Nov. 1998.

[85] J. Manner and M. Kojo. RFC 3753: Mobility Related Terminology. Request for Comments 3573, Internet Engineering Task Force, June 2004.

[86] L. Mathy and L. Lannone. LISP-DHT: Towards a DHT to map identifiers onto locators. In *Proc. of ACM ReArch 2008*, Dec. 2008.

[87] D. Maughan, M. Schertler, M. Schneider, and M. Turner. RFC 2408: Internet security association and key management protocol (ISAKMP). Request for Comments 2408, Internet Engineering Task Force, Nov. 1998.

[88] Merkur. The eMule Project Homepage, 2004. `http://www.emule-project.net`, 27.11.2011.

[89] C. Metz and J. ichiro itojun Hagino. IPv4-Mapped Addresses on the Wire Considered Harmful: draft-itojun-v6ops-v4mapped-harmful-02. Internet draft, Internet Engineering Task Force, Oct. 2003. Expired.

[90] R. Moskowitz, P. Jokela, T. Henderson, and T. Heer. RFC 5201bis: Host identity protocol. Internet draft, Internet Engineering Task Force, Jan. 2011. Work in progress. Expires in July, 2011.

[91] R. Moskowitz and P. Nikander. RFC 4423: Host Identity Protocol (HIP) Architecture. Request for Comments 4423, Internet Engineering Task Force, May 2006.

[92] R. Moskowitz, P. Nikander, P. Jokela, and T. R. Henderson. RFC 5201: Host Identity Protocol. Request for Comments 5201, Internet Engineering Task Force, Apr. 2008.

[93] P. Nikander, T. Henderson, C. Vogt, and J. Arkko. RFC 5206: End-Host Mobility and Multihoming with the Host Identity Protocol. Request for Comments 5206, Internet Engineering Task Force, Apr. 2008.

[94] P. Nikander and J. Laganier. RFC 5205: Host Identity Protocol (HIP) Domain Name System (DNS) Extension. Request for Comments 5205, Internet Engineering Task Force, Apr. 2008.

[95] P. Nikander, J. Laganier, and F. Dupont. RFC 4843: An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID). Request for Comments 4843, Internet Engineering Task Force, Apr. 2007.

[96] E. Nordmark and M. Bagnulo. RFC 5533: Shim6: Level 3 Multihoming Shim Protocol for IPv6. Request for Comments 5533, Internet Engineering Task Force, June 2009.

[97] E. Nordmark, S. Chakrabarti, and J. Laganier. RFC 5014: IPv6 Socket API for Source Address Selection. Request for Comments 5014, Internet Engineering Task Force, Sept. 2007.

[98] S. Novaczki, L. Bokor, and S. Imre. Micromobility support in HIP: survey and extension of host identity protocol. In *Electrotechnical Conference. MELECON 2006. IEEE Mediterranean*, pages 651 – 654, May 2006.

[99] H. Orman. RFC 2412: The OAKLEY Key Determination Protocol. Request for Comments 2412, Internet Engineering Task Force, Nov. 1998.

[100] P. Paakkonen, P. Salmela, R. Aguero, and J. Choque. Performance analysis of HIP-based mobility and triggering. In *Proceedings of International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008*, Newport Beach, CA, Jun 2008.

[101] N. Palix, G. Thomas, S. Saha, C. Calvès, J. L. Lawall, and G. Muller. Faults in linux: ten years later. In *ASPLOS*, pages 305–318, 2011.

[102] J. Pang, J. Hendricks, A. Akella, R. De Prisco, B. Maggs, and S. Seshan. Availability, usage, and deployment characteristics of the domain name system. In *In Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, New York, NY, USA, 2004*. ACM Press, 2004.

[103] V. Pappas, D. Massey, A. Terzis, and L. Zhang. A comparative study of the dns design with dht-based alternatives. In *In the Proceedings of IEEE INFOCOM'06*. IEEE, Apr. 2006.

[104] K. G. Paterson and A. K. L. Yau. Cryptography in theory and practice: The case of encryption in ipsec. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, st. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*. Springer-Verlag, 2006.

[105] A. Pathak, M. Komu, and A. Gurtov. HIPL: Give a name to your linux box. In *Linux Journal*, Nov. 2009.

[106] A. Pathak, M. Komu, and A. Gurtov. Host Identity Protocol for Linux. In *Linux Journal*, Nov. 2009.

[107] R. Pereira and R. Adams. RFC 2451: The ESP CBC-mode cipher algorithms. Request for Comments 2451, Internet Engineering Task Force, Nov. 1998.

[108] C. Perkins and et al. RFC 3344: IP Mobility Support for IPv4. Request for Comments 3344, Internet Engineering Task Force, Aug. 2002.

[109] D. Piper. RFC 2407: The internet IP security domain of interpretation for ISAKMP. Request for Comments 2407, Internet Engineering Task Force, Nov. 1998.

[110] V. Ramasubramanian and E. Sirer. The design and imlementation of a next generation name service for the internet. In *In Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications SIGCOMM '04*, volume 34. ACM Press, 2004.

[111] V. Ramasubramanian and E. G. Sirer. Beehive: O(1)lookup performance for power-law query distributions in peer-to-peer overlays. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 8–8, Berkeley, CA, USA, 2004. USENIX Association.

[112] E. Rescorla. RFC 2631: Diffie-Hellman Key Agreement Method. Request for Comments 2631, Internet Engineering Task Force, June 1999.

[113] E. Rescorla. *SSL and TLS, Designing and Building Secure Systems.* Addison-Wesley, 2006. Tenth printing.

[114] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. In *Proceedings of ACM SIGCOMM 2005*, Aug. 2005.

[115] R. L. Rivest. The rc4 encryption algorithm. Technical report, RSA Data Security Inc., Mar. 1992.

[116] R. L. Rivest. The RC5 encryption algorithm, from dr. dobb's journal, january, 1995. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996.* Stallings, 1996.

[117] S. H. Rodrigues, T. E. Anderson, and D. E. Culler. High-Performance Local Area Communication With Fast Sockets. In *In Proceedings of the USENIX Technical Conference*, pages 257–274, 1997.

[118] J. Rosenberg. RFC 5245: Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. Request for Comments 5245, Internet Engineering Task Force, Apr. 2010.

[119] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *In: Middleware*, pages 329–350, 2001.

[120] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, Nov. 2001.

[121] P. Salmela and J. Melen. Host identity protocol proxy. In *E-business and Telecommunication Networks, Communications in Computer and Information Science*, pages 126 – 138, nov 2007.

[122] D. C. Schmidt. The adaptive communication environment: An object-oriented network programming toolkit for developing communication software. pages 214–225, 1993.

[123] B. Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings*, pages 191–204. Cambridge University, Dec. 1993. Printed by Springer-Verlag in 1994.

[124] S. Shütz, L. Eggert, S. Schmid, and M. Brunner. Protocol enhancements for intermittenly connected hosts. In *ACM SIGCOMM Computer Communication Review*, pages 5 – 18, July 2005.

[125] M. Smith and T. Howes. RFC 4516: Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator. Request for Comments 4516, Internet Engineering Task Force, Dec. 2006.

[126] H. Soliman and et al. RFC 5555: Mobile IPv6 Support for Dual Stack Hosts and Routers. Request for Comments 5555, Internet Engineering Task Force, June 2009.

[127] P. Srisuresh and K. Egevang. RFC 3022: Traditional IP Network Address Translator (Traditional NAT). Request for Comments 3022, Internet Engineering Task Force, Jan. 2001.

[128] W. Stevens, M. Thomas, E. Nordmark, and T. Jinmei. RFC 3542: Advanced Sockets Application Program Interface (API) for IPv6. Request for Comments 3542, Internet Engineering Task Force, May 2003.

[129] W. R. Stevens, B. Fenner, and A. M. Rudoff. *Unix Network Programming, Volume 1, The Sockets Networking API.* Addison-Wesley, 2004. Fourth printing.

[130] R. Stewart. RFC 4960: Stream Control Transmission Protocol. Request for Comments 4960, Internet Engineering Task Force, Sept. 2007.

[131] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. Technical notes, Laboratory for Computer Science, Massachusetts Institute of Technology, Jan. 2002.

[132] T.Dierks and E. Rescorla. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2. Request for Comments 5246, Internet Engineering Task Force, Aug. 2008.

[133] G. Tsirtsis, V. Park, and H. Soliman. RFC 5454: Dual Stack Mobile IPv4. Request for Comments 5454, Internet Engineering Task Force, Mar. 2009.

[134] G. Tsirtsis and H. Soliman. RFC 4977: Problem Statement: Dual Stack Mobility. Request for Comments 4977, Internet Engineering Task Force, Aug. 2007.

[135] J. Ubillos, M. Xu, Z. Ming, and C. Vogt. Name Based Sockets: draft-ubillos-name-based-sockets-03. Internet draft, Internet Engineering Task Force, Sept. 2010. Expired.

[136] S. Varjonen. HIP and User Authentication: draft-varjonen-hip-eap-00. Internet draft, Internet Engineering Task Force, July 2009. Expired.

[137] S. Varjonen and A. Gurtov. Separating friends from spitters. In *International reports on socio-informatics - Workshop Proceedings of the 9th International Conference on the Design of Cooperative Systems*, volume 7, 2010. COOP 2010.

[138] S. Varjonen, T. Heer, K. Rimey, and A. Gurtov. Secure Resolution of End-Host Identifiers for Mobile Clients. In *Proceedings of GLOBECOM '11*, 2011. This paper received the best paper award at the Next Generation Networking (NGN) Symposium.

[139] S. Varjonen, M. Komu, and A. Gurtov. Secure and efficient ipv4/ipv6 handovers using host-based identifier-locator split. In *Proceedings of the 17th international conference on Software, Telecommunications and Computer Networks*, SoftCOM'09, pages 111–115, Piscataway, NJ, USA, 2009. IEEE Press.

[140] S. Varjonen, M. Komu, and A. Gurtov. Secure and Efficient IPv4/IPv6 Handovers Using Host-Based Identifier-Locator Split. *Journal of Communications Software and Systems*, 6(1), 2010.

[141] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. RFC 136: Dynamic Updates in the Domain Name System (DNS UPDATE). Request for Comments 2136, Internet Engineering Task Force, Apr. 1997.

[142] D. Wing, S. Cheshire, M. Boucadair, R. Penno, and F. Dupont. Port Control Protocol (PCP): draft-ietf-pcp-base-27. Internet draft, Internet Engineering Task Force, Sept. 2012. Work in progress.

[143] D. Wing and A. Yourtchenko. RFC 6555: Happy Eyeballs: Success with Dual-Stack Hosts. Request for Comments 6555, Internet Engineering Task Force, Apr. 2012.

[144] J. Ylitalo. *Secure Mobility at Multiple Granularity Levels over Heterogeneous Datacom Networks.* PhD thesis, University of Helsinki, 2008.

[145] J. Ylitalo, J. Melén, P. Nikander, and V. Torvinen. Re-thinking Security in IP-Based Micro Mobility. In *Lecture Notes in Computer Science*, pages 318 – 329, 2004. ISBN 978-3-540-23208-7.

[146] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, Jan. 2004.

[147] B. Y. Zhao, J. Kubiatowicz, A. D. Joseph, B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, 2001.

TIETOJENKÄSITTELYTIETEEN LAITOS
PL 68 (Gustaf Hällströmin katu 2 b)
00014 Helsingin yliopisto

JULKAISUSARJA **A**

DEPARTMENT OF COMPUTER SCIENCE
P.O. Box 68 (Gustaf Hällströmin katu 2 b)
FIN-00014 University of Helsinki, Finland

SERIES OF PUBLICATIONS **A**

Reports may be ordered from: Kumpula Science Library, P.O. Box 64, FIN-00014 University of Helsinki, Finland.

A-2005-2 A. Doucet: Advanced Document Description, a Sequential Approach. 161 pp. (Ph.D. Thesis)

A-2006-1 A. Viljamaa: Specifying Reuse Interfaces for Task-Oriented Framework Specialization. 285 pp. (Ph.D. Thesis)

A-2006-2 S. Tarkoma: Efficient Content-based Routing, Mobility-aware Topologies, and Temporal Subspace Matching. 198 pp. (Ph.D. Thesis)

A-2006-3 M. Lehtonen: Indexing Heterogeneous XML for Full-Text Search. 185+3 pp. (Ph.D. Thesis)

A-2006-4 A. Rantanen: Algorithms for ^{13}C Metabolic Flux Analysis. 92+73 pp. (Ph.D. Thesis)

A-2006-5 E. Terzi: Problems and Algorithms for Sequence Segmentations. 141 pp. (Ph.D. Thesis)

A-2007-1 P. Sarolahti: TCP Performance in Heterogeneous Wireless Networks. 171 pp. (Ph.D. Thesis)

A-2007-2 M. Raento: Exploring privacy for ubiquitous computing: Tools, methods and experiments. 61+150 pp. (Ph.D. Thesis)

A-2007-3 L. Aunimo: Methods for Answer Extraction in Textual Question Answering. 127+18 pp. (Ph.D. Thesis)

A-2007-4 T. Roos: Statistical and Information-Theoretic Methods for Data Analysis. 82+75 pp. (Ph.D. Thesis)

A-2007-5 S. Leggio: A Decentralized Session Management Framework for Heterogeneous Ad-Hoc and Fixed Networks. 230 pp. (Ph.D. Thesis)

A-2007-6 O. Riva: Middleware for Mobile Sensing Applications in Urban Environments. 195 pp. (Ph.D. Thesis)

A-2007-7 K. Palin: Computational Methods for Locating and Analyzing Conserved Gene Regulatory DNA Elements. 130 pp. (Ph.D. Thesis)

A-2008-1 I. Autio: Modeling Efficient Classification as a Process of Confidence Assessment and Delegation. 212 pp. (Ph.D. Thesis)

A-2008-2 J. Kangasharju: XML Messaging for Mobile Devices. 24+255 pp. (Ph.D. Thesis).

A-2008-3 N. Haiminen: Mining Sequential Data – in Search of Segmental Structures. 60+78 pp. (Ph.D. Thesis)

A-2008-4 J. Korhonen: IP Mobility in Wireless Operator Networks. 186 pp. (Ph.D. Thesis)

A-2008-5 J.T. Lindgren: Learning nonlinear visual processing from natural images. 100+64 pp. (Ph.D. Thesis)

A-2009-1 K. Hätönen: Data mining for telecommunications network log analysis. 153 pp. (Ph.D. Thesis)

A-2009-2 T. Silander: The Most Probable Bayesian Network and Beyond. 50+59 pp. (Ph.D. Thesis)

A-2009-3 K. Laasonen: Mining Cell Transition Data. 148 pp. (Ph.D. Thesis)

A-2009-4 P. Miettinen: Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms. 164+6 pp. (Ph.D. Thesis)

A-2009-5 J. Suomela: Optimisation Problems in Wireless Sensor Networks: Local Algorithms and Local Graphs. 106+96 pp. (Ph.D. Thesis)

A-2009-6 U. Köster: A Probabilistic Approach to the Primary Visual Cortex. 168 pp. (Ph.D. Thesis)

A-2009-7 P. Nurmi: Identifying Meaningful Places. 83 pp. (Ph.D. Thesis)

A-2009-8 J. Makkonen: Semantic Classes in Topic Detection and Tracking. 155 pp. (Ph.D. Thesis)

A-2009-9 P. Rastas: Computational Techniques for Haplotype Inference and for Local Alignment Significance. 64+50 pp. (Ph.D. Thesis)

A-2009-10 T. Mononen: Computing the Stochastic Complexity of Simple Probabilistic Graphical Models. 60+46 pp. (Ph.D. Thesis)

A-2009-11 P. Kontkanen: Computationally Effcient Methods for MDL-Optimal Density Estimation and Data Clustering. 75+64 pp. (Ph.D. Thesis)

A-2010-1 M. Lukk: Construction of a global map of human gene expression - the process, tools and analysis. 120 pp. (Ph.D. Thesis)

A-2010-2 W. Hämäläinen: Efficient search for statistically significant dependency rules in binary data. 163 pp. (Ph.D. Thesis)

A-2010-3 J. Kollin: Computational Methods for Detecting Large-Scale Chromosome Rearrangements in SNP Data. 197 pp. (Ph.D. Thesis)

A-2010-4 E. Pitkänen: Computational Methods for Reconstruction and Analysis of Genome-Scale Metabolic Networks. 115+88 pp. (Ph.D. Thesis)

A-2010-5 A. Lukyanenko: Multi-User Resource-Sharing Problem for the Internet. 168 pp. (Ph.D. Thesis)

A-2010-6 L. Daniel: Cross-layer Assisted TCP Algorithms for Vertical Handoff. 84+72 pp. (Ph.D. Thesis)

A-2011-1 A. Tripathi: Data Fusion and Matching by Maximizing Statistical Dependencies. 89+109 pp. (Ph.D. Thesis)

A-2011-2 E. Junttila: Patterns in Permuted Binary Matrices. 155 pp. (Ph.D. Thesis)

A-2011-3 P. Hintsanen: Simulation and Graph Mining Tools for Improving Gene Mapping Efficiency. 136 pp. (Ph.D. Thesis)

A-2011-4 M. Ikonen: Lean Thinking in Software Development: Impacts of Kanban on Projects. 104+90 pp. (Ph.D. Thesis)

A-2012-1 P. Parviainen: Algorithms for Exact Structure Discovery in Bayesian Networks. 132 pp. (Ph.D. Thesis)

A-2012-2 J. Wessman: Mixture Model Clustering in the Analysis of Complex Diseases. 119 pp. (Ph.D. Thesis)

A-2012-3 P. Pöyhönen: Access Selection Methods in Cooperative Multi-operator Environments to Improve End-user and Operator Satisfaction. 211 pp. (Ph.D. Thesis)

A-2012-4 S. Ruohomaa: The Effect of Reputation on Trust Decisions in Inter-enterprise Collaborations. 214+44 pp. (Ph.D. Thesis)

A-2012-5 J. Sirén: Compressed Full-Text Indexes for Highly Repetitive Collections. 97+63 pp. (Ph.D. Thesis)

A-2012-6 F. Zhou: Methods for Network Abstraction. 48+71 pp. (Ph.D. Thesis)

A-2012-7 N. Välimäki: Applications of Compressed Data Structures on Sequences and Structured Data. 73+94 pp. (Ph.D. Thesis)