

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering

**Supporting Multiple IPsec Security
Associations in Host Identity Protocol
Mobility and Multihoming**

Master's Thesis

Mika Kousa

Telecommunications Software and Multimedia Laboratory
Espoo 2006

Author:	Mika Kousa	
Name of the thesis:	Supporting Multiple IPsec Security Associations in Host Identity Protocol Mobility and Multihoming	
Date:	30th September 2006	Number of pages: 86
Department:	Department of Computer Science and Engineering	
Professorship:	T-110 Telecommunication Software And Applications	
Supervisor:	Professor Antti Ylä-Jääski	
Instructor:	Janne Lindqvist, M.Sc.	
<p>Mobility has become a standard feature for many of the hosts in the Internet. Some of these hosts have even multiple simultaneous network connections, which is called as multihoming. However, the protocols used widely today in the Internet were not originally designed with mobility and multihoming in mind. Also, applications are affected by the changed situation. All this has prevented from taking the full advantage of these concepts into normal daily use.</p> <p>Several new networking protocols or workarounds to the existing protocols have been suggested to improve the situation. This thesis presents one of these new protocols, the Host Identity Protocol (HIP). By using HIP as an example, we show how it can be used to address these problematic issues. HIP is also related to a new concept of identifier/locator split, which is closely related to mobility and multihoming.</p> <p>This thesis presents and proposes solutions to the areas, which must be taken care of when implementing a new network protocol supporting the new usage requirements. The solutions are presented both from the operating system and application level point of view.</p> <p>An analysis based on measurements on a HIP implementation concludes this thesis.</p>		
<p>Keywords: HIP, identifier/locator split, mobility, multihoming, IPsec</p>		

Tekijä:	Mika Kousa	
Työn nimi:	Monen IPsec-assosiaation toteuttaminen Host Identity protokollaan	
Päivämäärä:	30. syyskuuta 2006	Sivuja: 86
Osasto:	Tietotekniikan osasto	
Professuuri:	T-110 Tietoliikenneohjelmistot	
Työn valvoja:	Professori Antti Ylä-Jääski	
Työn ohjaaja:	DI Janne Lindqvist	
<p>Mobiliteetti on muodostunut melkein pä perusominaisuudeksi useimmille Internetissä oleville verkottuneille laitteille. Joillakin näistä laitteista on useampikin kuin vain yksi liityntä verkkoon, jota kutsutaan nimityksellä <i>multihoming</i>. Nykyisin Internetissä käytössä olevat tietoliikenneprotokollat eivät kuitenkaan ole alunperin suunniteltu mobiliteettia tai multihomingia huomioon ottaen. Muuttunut tilanne vaikuttaa myös sovelluksiin. Puutteet estävät mobiliteetin ja multihomingin täysimittaisen hyödyntämisen.</p> <p>Useita uusia protokollia ja korjausehdotuksia on ehdotettu nykytilanteen korjaamiseksi. Tämä diplomityö käsittelee yhtä tällaista uutta protokollaa, joka on Host Identity protokolla (HIP). HIP on esimerkkinä, miten sen avulla voidaan ratkaista mobiliteettiin ja multihomingiin liittyviä ongelmia. Lisäksi HIP liittyy uuteen käsitteeseen <i>identifier/locator</i>-jako, joka vastaavasti liittyy läheisesti mobiliteettiin ja multihomingiin.</p> <p>Tämä diplomityö esittelee ja ehdottaa ratkaisuja asioihin, jotka tulee ottaa huomioon toteutettaessa uutta tietoliikenneprotokollaa, jonka tarkoituksena on ratkaista mainittuja uusia vaatimuksia. Ratkaisut esitellään sekä käyttöjärjestelmän että sovellusten kannalta katsottuna.</p> <p>Diplomityö sisältää myös HIP-toteutuksen mittaustuloksiin perustuvan analyysin.</p>		
Avainsanat: HIP, identifier/locator-jako, mobiliteetti, multihoming, IPsec		

Contents

Abstract	ii
Tiivistelmä (in Finnish)	iii
Abbreviations	x
1 Introduction	1
1.1 From the Past to the Present	1
1.2 Problem Statement	2
1.3 Scope	2
1.4 Organization of the Thesis	2
2 Background	4
2.1 Multihoming	4
2.1.1 Site Multihoming	4
2.1.2 End-host Multihoming	5
2.1.3 Motivation for Multihoming	5
2.1.4 Problems with Multihoming	6
2.2 Mobility	8
2.2.1 Mobility Management	9
2.2.2 Mobility Levels	9
2.2.3 Mobility Categorization	10
2.2.4 Location Management	11
2.2.5 Handoff Management	11

2.3	Identifier/Locator Split	14
2.4	Security	15
2.4.1	IPsec	15
2.4.2	Effects of Mobility and Multihoming to IPsec	16
3	Problem Statement	18
3.1	Research Problems	18
3.2	Elaboration of the Research Problems	19
3.2.1	Identifier/Locator Split	19
3.2.2	Implications of Identifier/Locator Split on Mobility and Multihoming	19
4	Related work	21
4.1	Mobile IPv6	21
4.1.1	Mobile IPv6 Components	22
4.1.2	Data Transfer in Mobile IPv6	22
4.1.3	Route Optimization and Registration of Correspondent Node	23
4.2	Hash Based Addresses	24
4.2.1	HBA address set	25
5	Host Identity Protocol	26
5.1	Introduction to Host Identity Protocol	26
5.2	Host Identity Namespace	27
5.2.1	Host Layer	27
5.2.2	Mobility and Multihoming with HIP	28
5.2.3	Representations of a Host Identifier	28
5.3	The HIP Base Exchange	29
6	Design of Implementation	31
6.1	Architecture of the Linux Networking Stack	31
6.1.1	Output Packet Processing	32
6.1.2	Input Packet Processing	33

6.1.3	Related Support Functionality	34
6.2	Required Changes to the Linux Operating System Kernel . . .	37
6.2.1	Network Input and Output	37
6.2.2	Storing of IPsec SAs	37
6.2.3	Caching of Data Structures	38
6.2.4	Issues on Lower And Higher Layers	39
6.2.5	Event Handling	39
6.3	Grouping of IP Addresses into IPsec Security Associations . .	40
6.3.1	Requirements	40
6.4	Selection of Outbound ESP SA	44
6.5	Dynamic Outbound Path Probing	45
7	Analysis	47
7.1	Measurement Goals	47
7.2	Measurement Non-Goals	48
7.3	Measurement Methods	49
7.3.1	Hardware	49
7.3.2	Software	49
7.3.3	Definitions of the Measured Targets	50
7.3.4	Definitions of the Measured Values	51
7.4	Measurement Results	52
7.4.1	Bulk Data Transfers	52
7.4.2	Common Network System Calls Used in Networking Applica- tions	53
7.5	Analysis of the Results	62
7.5.1	About the Measurement Timings	62
7.5.2	Bulk Data Transfers	62
7.5.3	System Calls	63
7.5.4	Using Different Host Key Types	64
8	Future work	66
8.1	Multiple IPsec Security Associations	66

8.2 Other Interesting Areas to Research	67
9 Conclusions	68
A Example applications	70
A.1 Client	70
A.2 Server	71

List of Figures

2.1	Site Multihoming	5
2.2	End-host Multihoming	5
2.3	Identity Layer	14
5.1	The HIP Base Exchange	30
7.1	Host A client, no Host Association, connect	59
7.2	Host A client, Host Association exists, connect	59
7.3	Host A client, no Host Association, send	60
7.4	Host A client, Host Association exists, send	60
7.5	Host A server, no Host Association, recv	61
7.6	Host A server, Host Association exists, recv	61

List of Tables

7.1	Data Transfer Times (in seconds)	52
7.2	Data Transfer Times (in seconds) (continued)	52
7.3	Data Transfer Rates (in MBps)	53
7.4	Data Transfer Rates (in MBps) (continued)	53
7.5	Host A, without HIP	54
7.6	Host B, without HIP	54
7.7	Host A, with HIP, no existing Host Association	55
7.8	Host B, with HIP, no existing Host Association	55
7.9	Host A, with HIP, Host Association exists	56
7.10	Host B, with HIP, Host Association exists	56
7.11	Absolute and relative changes on host A, no HIP vs. HIP . . .	57
7.12	Absolute and relative changes on host A, no HIP vs. HIP (con- tinued)	57
7.13	Absolute and relative changes on host B, no HIP vs. HIP . . .	58
7.14	Absolute and relative changes on host B, no HIP vs. HIP (con- tinued)	58

Abbreviations

AAA Authentication, Authorization and Accounting

AH Authentication Header

API Application Programming Interfaces

DHCP Dynamic Host Configuration Protocol

DNS Domain Name Service

ESP Encapsulating Security Payload

GPRS Global Packet Radio Service

HBA Hash Based Address

HIP Host Identity Protocol

HIPL Host Identity Protocol for Linux

IETF Internet Engineering Task Force

IP Internet Protocol

IPsec Internet Protocol security

ISP Internet Service Provider

LAN Local Area Network

MIP Mobile Internet Protocol

OSI Open System Interconnection

PAN Personal Area Network

PDA Personal Digital Assistant

QoS Quality of Service

RFC Request For Comments

SADB Security Association Database

SCTP Stream Control Transmission Protocol

SPD Security Policy Database

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol / Internet Protocol

UDP User Datagram Protocol

VoIP Voice over IP

WLAN Wireless Local Area Network

Chapter 1

Introduction

1.1 From the Past to the Present

In the old days, when the computers were the size of a large room, the users could not even imagine that the computers could be connected to each other or even that the computers could be easily moved to an other location. A few decades later, the ARPANET was formed. The ARPANET connected initially only a small number of large systems, mainly universities, into one network. Soon it was noticed that the success was not only due to technical merits, but also how it affected the humans using the network. People could communicate much more easily and faster than before. At that point there was no turning back, because the benefits were clearly seen: the worldwide network, the Internet, is based on the ARPANET [3].

Moving on a couple of decades to the era of the new millennium, computing platforms began to be small enough to fit into a pocket. The computers had even some level of network connectivity. User experience showed that there is a real need for the ability to move around physically and still remain connected. This is commonly called as *mobility*. The advanced hosts had even multiple network connections, which is called as *multihoming*. However, the networking protocols were developed at the time when these issues were not even visible in the horizon. The network protocols could not be used efficiently with the technology at that time.

At the end of the 20th century, several proposals were being developed to improve the situation. As usual, the best way to achieve an internationally accepted design is to participate into the work done by the standardization organizations. Internet Engineering Task Force (IETF) is one of the most notable organization related to computer networking.

1.2 Problem Statement

Implementation of mobility, multihoming, and security protocols is a challenging task, even not to mention that the protocol implementing these issues must make them work together at the same time and not only one area. When new requirements arise, things are getting much more complicated if the existing protocols can not be bent enough to fulfill the new requirements.

To get an insight what issues will be raised when this kind of a new requirement is encountered, this thesis inspects the new concept of *identifier/locator split* using Host Identity Protocol (HIP) as the example implementation. HIP is a good example because using it we can show several problematic areas and the implications of identifier/locator split at the same time. For example, what modifications network stack of an operating system requires.

We concentrate in the implementation issues of HIP when putting mobility, multihoming, and security together. This way we will notice what are the typical issues encountered when implementing a protocol supporting the identifier/locator split and how they can be dealt with.

The problem statements are discussed in more detail in chapter 3.

1.3 Scope

The scope of this thesis is on the implementation issues of the identifier/locator split. The focus is in the operating system level, mainly the network stack part and modifications it needs. Another focus is on the design of the related items that are not actually implemented but what needs to be considered anyway.

Application level issues and security analysis of the proposed design work are out of scope of this thesis.

1.4 Organization of the Thesis

Chapter 2 defines and examines the fundamental key concepts needed for introducing the research area of this thesis and for further related discussion.

The problem statements this thesis focuses on are presented in chapter 3. A short background discussion about the issues is also given.

Thesis related work, two network protocols, is presented in chapter 4.

Chapter 5 presents an overview on the Host Identity Protocol, which is the

network protocol used in this thesis.

This thesis discusses also how implementation work should be performed. The design of implementation is in chapter 6.

An analysis on the implementation is presented in chapter 7. The analysis contains the measured results and graphs, and then discussion on the results.

Future work in chapter 8 gives thoughts what areas should be concentrated on in the future if research would be continued from where this thesis left out.

Finally, conclusions are presented in chapter 9.

Chapter 2

Background

To become familiar with the topic of this thesis, the relevant terminology and concepts are discussed in detail. Some of the most important and common existing technologies are presented during the discussion.

Next we take a closer look on the two concepts which were defined previously in the introduction chapter, multihoming and mobility.

2.1 Multihoming

Many of the networked hosts have only a single network interface that connects the host to a network. The single interface is sufficient for most of the users, because they use their hosts mainly as a client which use services provided by servers. This situation is referred to as a *single-homed host*.

Nodes may also have many simultaneous connected network interfaces. The interfaces do not necessarily have to have same kind of properties. For example, a node might have an Ethernet network interface and a wireless interface. A common example of a node having multiple interfaces is a router. Additionally, a host or a node may use a mechanism where it has one network interface which is assigned multiple addresses. A host having multiple network interfaces or multiple addresses in one network interface is called a *multihomed host*.

2.1.1 Site Multihoming

Nodes belong to a larger entity, a site. A site is “an entity autonomously operating a network using IP and, in particular, determining the addressing plan and routing policy for that network” [1]. A “transit provider” “operates a

site which directly provides connectivity to the Internet to one or more external sites. The connectivity provided extends beyond the transit provider’s own site. A transit provider’s site is directly connected to the sites for which it provides transit” [1].

Using these definitions, *site-multihoming* refers to a situation where a site has more than one transit provider. Site-multihoming is depicted in figure 2.1.

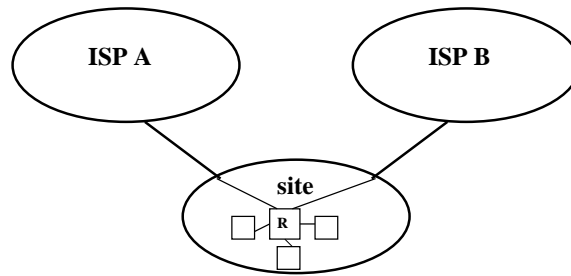


Figure 2.1: Site Multihoming

2.1.2 End-host Multihoming

End-host multihoming means that a host has more than one network interface which are connected to networks as in figure 2.2. Interfaces are either physical or virtual. The interfaces might also have multiple addresses associated with them. Modern laptop computers are good examples of end-host multihoming capable end-hosts: usually they have at least built-in LAN and modem connections, and often wireless LAN (WLAN) cards, too.

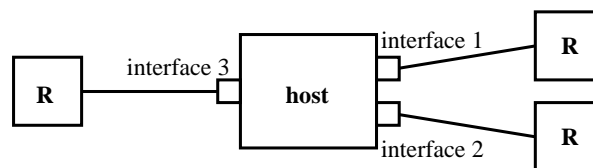


Figure 2.2: End-host Multihoming

2.1.3 Motivation for Multihoming

Multihoming has some noticeable advantages: To save data transfer costs a site can use several Internet Service Providers (ISP) and route data traffic to a certain ISP depending on where it is cheapest to forward to. Similarly, data traffic needing a special kind of a service level requirements can be directed

to the link which can provide the requested service level if all of the links do not have adequate characteristics such as sufficient bandwidth or low transfer delay. This is important e.g. for VoIP or video applications which require low latencies or “bulk data transfers” (e.g. FTP) which would benefit from high bandwidth but do not need low latencies.

One of the most important advantage gained from multihoming is increased fault-tolerance and redundancy. When one of the host’s links goes down the host can use another link to continue the communication. Similarly on larger scale, if there is a major network problem (e.g. hardware related) in one of the Internet Service Providers (ISP) a site is connected, the site can just use other ISPs it is connected to.

Multihoming provides also a way for load sharing. A site might have a policy which says that outgoing traffic to the hosts belonging to the network N is routed via ISP A and all other traffic via ISP B (maybe because network N is directly connected to the ISP A). Another example of a policy is that traffic is routed by default via slow but cheap ISP A but traffic generated by some busy hosts requiring more bandwidth is routed via fast and more expensive ISP B. By carefully designing ways for load sharing, there might be notable network performance increases or cost savings.

Having simultaneously more than one address eases renumbering transition. Nodes and hosts can be configured e.g. to use new address prefixes and routes while the transition is in progress and remove the old ones from the nodes/hosts when transition is finished.

Some mobility mechanisms might use multiple addresses at the same time, e.g. when a node is changing its location. It has received its new address while still maintaining the old address in case there might still be packets in the network which have the node’s old address as the destination.

Some networking protocol are even designed with multihoming aspects in mind. For example one of them, IPv6, is designed so that a node may have multiple unicast addresses which are used for different reachability scopes (link-local, site and global) [11].

2.1.4 Problems with Multihoming

The use of multihoming is not as straightforward as it seems to be at first look. Just by making service agreements with several ISPs and setting up routing tables is not enough for efficient multihoming. Next, we list some problems with multihoming.

Technical Issues

Technical issues related to multihoming span several levels of the OSI model [30], but most notable effects are on levels three (internetworking) and four (transport). Extensive use of multihoming increases sizes of the routing tables in the routers, thus decreasing routing performance. Other hosts than the one in the multihomed site are also affected by this because the backbone routers are slowed up.

Address selection (both for source and destination addresses) is one of the major problems with multihoming. This selection process is affected by at least the following aspects [6]:

- administrative policies
- characteristics (QoS, bandwidth, delay etc) on different interfaces
- the cost of using a certain interface
- the requirements of applications, user, or operator

If the characteristics of all of the interfaces are different, there needs to be some kind of a decision (e.g. based on the costs) on how to select the most suitable interface for the given destination address. Detection of changes in the interface states is also needed, such as when an interface goes up or down or when its characteristics change.

When choosing the source address to be used for sending data it must be noted that the selected source address can affect the address and path used for receiving traffic. The data might even not reach the destination if the selected source address has insufficient scope. Address selection issues are discussed more in more detail in [7].

A network connection in TCP is identified as a pair of interconnected sockets at the ends, which can be represented as $\{source\ IP\ address, source\ port, destination\ IP\ address, destination\ port\}$ [29]. When an application connects to a remote host, it is bound to a certain source address and port. When the connection is established, the source and destination addresses or ports do not change. This causes a few consequences:

- An application on a host A binds itself to an other interface some time later, its source address is different from what it was before. Now, when the peer host B receives data from the host A, the data packets contain

the new source address. Problems might arise if the host B acts differently based on the source address it receives the data from even if the source addresses are located at the same host. (If addresses are used as identifiers, the application can not easily tell if different addresses identify the same peer.)

- Mobility becomes harder, because mobile hosts might change their addresses frequently. Changing of either of the addresses breaks the connection.

2.2 Mobility

The term *mobility* means that an entity changes its topological point of attachment to a network while keeping its network connections active. An essential requirement in mobility is that elements using the connections (e.g. a process in a host having a TCP connection) do not notice the mobility expect possibly in changes to QoS parameters [25].

In the simplest form of mobility, the connection is re-established when the location changes. In a better scheme, existing network architecture provides continuous connectivity to mobile devices when they change their location. The most ideal scheme would be that everything is done completely dynamically and invisibly, possibly without any previous knowledge of the existing network architecture. The last ideal scheme is typically known as *mobile ad hoc networking*.

The next important question when designing mobility architecture is “where in the network protocol stack should mobility implemented?”. This decision affects greatly the features and characteristics what can be done within the mobility implementation. [14] discusses some issues what should be noted when selecting a level in the networking stack:

- **physical layer** resource reuse, interference avoidance
- **link layer** bandwidth, security, reliability, channel allocation algorithms, collision detection and avoidance, QoS resource management
- **network layer** needs new routing algorithms, handling of node movement and connectivity
- **transport layer** differences between characteristics between different networks (e.g. wireless and wired) needs developing of current algorithms

- **middleware and application layer** new requirements on middleware, opportunities to applications (context-awareness etc)

Despite the method of implementing mobility, mobility management is needed. This is discussed next.

2.2.1 Mobility Management

Mobility management refers to technologies and supportive methods which provide mobility when combined together. The purpose of mobility management is to “automatically support mobile terminals enjoying their services while simultaneously roaming freely without the disruption of communications” [14]. That is, networks need to locate devices and maintain the connections.

Mobility management contains two important aspects: *location management* and *handoff management* (handoff is also known as *handover*). These are discussed briefly later in sections 2.2.4 and 2.2.5.

2.2.2 Mobility Levels

As mentioned before, mobile devices roam within or between different sized systems. These systems have unique characteristics and requirements depending on the size of a system. Networks can be classified according to different providers and/or technologies. A network consists of domains. Domains have location areas, which in turn have access point regions. The regions have zones of access points. An access point has logical channels [14].

Every level of a system presents a specific set of requirements for the mobility management it needs. Very low level system does not need the same level of complexity as a high level system does. For example, a low level system might not necessarily have use for *location updates* (process of informing the current address), whereas that is mandatory when moving from a network to an other network. Using the definitions from [14], *Mobility levels* could be categorized having different granularity:

- mega-mobility, between networks
- macro-mobility, between domains, within one network
- micro-mobility, between location areas, within one domain
- mini-mobility, between access point regions, within one location area

- pico-mobility, between access points, within one access point region
- nano-mobility, within the zone covered by one access point (which may have many logical channels)

2.2.3 Mobility Categorization

Mobility can also be categorized depending on what kind of a mobility management scheme is needed for a mobile system. Some user requirements can be satisfied just by using one level of mobility management scheme, but a more complicated scheme might require using more than one scheme for efficient mobility usage. A rough categorization of the mobility schemes follow next.

Network mobility means that an entire network changes its point of attachment to another network (and consequently reachability in the network topology). This kind of a network is often called as a mobile network. *Personal Area Network* (PAN) is an example situation where network mobility is needed. For example, a PAN might consist of wireless interconnected devices (such as a mobile phone or a PDA) belonging to a person carrying them. All these devices together create a very small network. When the person walks around, the PAN notices the movement and perform functions needed to update its location information [20].

Host mobility (or *terminal mobility*) refers to a situation where a host changes its point of attachment to the network. During this process host's network connections remain uninterrupted. A required feature for the access networks providing host mobility for the mobile hosts is that the networks track continually hosts' current location. If continuous tracking is not possible, there should be at least some mechanism to find out the location quickly [20].

In *User mobility* (or *session mobility*) user can access services from different physical hosts, e.g. when the user moves from a terminal to another terminal. It is assumed here that the user is authorized to use these services [20].

Personal mobility tracks the location of the user and provides ways to initiate/receive sessions to/from the other users of the network by using the current location information of the user. Personal mobility involves security issues (such as what types of connections are allowed and from where they are allowed), billing and service subscription authorization between administrative domains [20].

Application mobility represents a higher level of mobility scheme. One example of application mobility could be software agents which move in the network searching for information based on some initial query parameters given by an user [20].

2.2.4 Location Management

In section 2.2.1 we mentioned that *location management* is the other of the two important things in mobility management (the other was *handoff management*). Without proper location management mobile devices can not operate decently because the peer nodes have to be informed of the address changes.

Location management procedures include addressing related operations: location registration include a way to inform the middleboxes of the new location of the mobile node. Similarly, other nodes need to have a procedure to find the mobile node's location. To speed up the location management procedures, node locations could be stored in a location database or some other cache, but then there is a problem of receiving old and possible invalid addresses. Dynamic query methods for finding the current location, for example sending a query to broadcast addresses, provides a solution to this problem. However, it might have a drawback of being more bandwidth consuming than using a database or cache based location management method. Broadcasting is also restricted within the same local network.

Location management is a complex issue, and it has many open topic areas for research. Some kind of a global addressing scheme and common address representation are needed if mobile nodes move between different type of networks with different type of addresses.

Design of location database structure affects the location query and update performance. The database could be either centralized, distributed, or a combined approach. When the database is used, overhead from increased resource use (retrieving, storing, and transferring of addresses) cannot be avoided.

A general method for location update is hard to design, because different networks may have different mobility related requirements. The point of decision when to update can be determined statically, e.g. always when a mobile node moves to an other network, or in a more sophisticated and adaptive manner. A related problem is how a mobile node's exact location can be found within a given time frame. A method to query address query might be a simple flooding-like which consumes a lot of resources. A less resource consuming method is based on intelligent algorithms.

2.2.5 Handoff Management

The second part of the mobility management issues is handoff management. Handoff handles "the process by which an active mobile node changes its point of attachment to the network, or when such a change is attempted" [20]. Node's current connections are maintained during the handoff procedure.

Handoff Scopes and Concepts

A handoff has several scopes in which it can happen. These scopes are classified by different aspects involved in the handoff. The following description of the scopes is based on [20].

The lowest layer where handoff can practically happen is the link layer (OSI layer 2). *Layer 2 Handoff* happens e.g. when mobile node changes its radio access point. Layer 2 handoff does not necessarily involve interaction with internetworking layer. This implies a major advantage that Layer 2 handoff is transparent to the upper layers, possibly only thing there needs to be done is to reconfigure the link layer.

A handoff in which an access router's (AR) interface to the mobile node changes is called as *Intra-AR Handoff*. The access router does only internal routing related updates.

When the mobile node moves within the same access network (AN) and changes the access router, *Intra-AN Handoff* is performed. Typically this kind of a handoff is not seen outside the access network. A situation where the handoff is seen outside the access network is when the access network gateway which serves the mobile node changes, because this affects the routing.

Inter-AN Handoff happens when the mobile node moves to a new access network. Inter-AN Handoff requires host mobility mechanisms across access networks, such as getting a new address for the mobile node due to the change in the node's topological location in the network.

Intra-technology Handoff is a handoff between equipment of the same technology. Similarly *Inter-technology Handoff* is a handoff between equipment of different technologies.

In *horizontal handoff* a mobile node makes a handoff from an access network to an other access network using the same network interface. A typical horizontal handoff is an intra-technology handoff but it can also be an inter-technology handover if the MN can do a layer 2 handover between two different technologies without changing the network interface seen by the IP layer.

In *vertical handoff* mobile node's network interface to access network changes (e.g. from WLAN to LAN), network connections from a mobile node move from one interface to another. A vertical handover is typically an inter-technology handover but it may also be an intra-technology handover if the MN has several network interfaces of the same type. That is, after the handover, the IP layer communicates with the access network through a different network interface.

Functional Operations of Handoff Management

Handoff management contains several functional operations: *handoff triggering*, *connection re-establishment*, and *packet routing*.

Handoff triggering is one of the fundamental operations in handoff management. It deals with issues which lead to the decision whether to initiate and perform handoff process or not. Typically, handoff process is triggered when certain conditions are met, such as when current connection's characteristics change noticeably, there is a better connection available, or underlying network topology changes. Triggering might even happen explicitly when the node or user forces the handoff process.

Connection re-establishing is used to set up a new connection to new attachment point or link. The process contains separate sub-procedures: finding the new suitable target for the connection, and making the actual connection. The third procedure, packet routing, takes care of routing packets along the new path after the new connection is set up.

Handoff Types

Handoff procedures can also be divided into different types depending on how the handoff procedure is actually implemented. These different architectural approaches have therefore distinct properties which may significantly affect the overall performance of the whole handoff procedure. Examples of these properties are: how long it takes the host to be able to use the new address, how many packets the handoff procedure needs to complete, how many packet retransmissions might be needed to take into account the lost packets while the handoff procedure takes place, and so on.

Typically handoffs are categorized into following types:

- ***Fast handoff***: low delay, short interruption time between disconnection at the old access router and connection to the new point of attachment
- ***Smooth handoff***: no packet loss (interruption time is minor issue) / low loss
- ***Seamless handoff***: includes fast and smooth handoff The new path is established in parallel with the old one
- ***Hard handoff***: Old connection is broken before the new connection is established.

- **Soft handoff:** New connection is established before the old connection is broken.

2.3 Identifier/Locator Split

The original IP architecture was designed so that there was a major assumption that the hosts do not move. This means that the IP addresses in an IP packet tell the identity of the hosts (“who you are”), but also the location of the hosts (the topological location of a host in a network, “where you are”). The IP address is also used in the routing decisions of the packet.

The upcoming need for mobility and multihoming changes these roles of IP addresses. In multihoming, the identity must be dynamically mappable to multiple locations and routing paths. In mobility, the identity of the host remains the same even after the host moves and its address is changed. The identity (being stable in its nature) of the endpoint needs to be distinguished from the network-based location of that endpoint for multihoming and mobility to work in a scalable manner. This role split is called as the *identifier/locator split*.

There are several approaches to implement the identifier/locator split. A straightforward one is to add a completely new layer to the existing networking stack. This layer handles the conversions from identities to locations and vice versa. A good choice for the location where the new layer could be inserted is above the networking layer and below transport layer as in figure 2.3, because this approach allows indirection between identity and location. With indirection multiple locators can be associated with the identity. Also, the sessions can be bound to the stable identities.

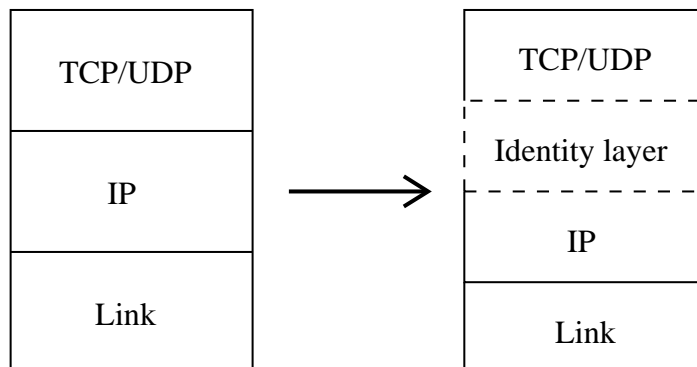


Figure 2.3: Identity Layer

The existing network or transport layers could also be modified to support

the split without adding a new layer. Network layer could support IP-in-IP structures that distinguish between current locator and persistent locator addresses. Transport layer can be altered to allow a number of locators to be associated with a session.

Additionally, modification to the behavior of the interaction between the host and the site exit router is one place to implement the identity/location conversion, so that the packets go to the correct destination no matter what the path is. All this might need that the exit router modifies the packet headers, there are source-based routing to allow host-based site exit router selection, and host and exit router exchange reachability information.

The identifier/locator split raises problematic implications. The mapping from a identifier to a locator (and vice versa) must be secure to prevent identity thefts. Mappings must also be performed transparently, otherwise the system will be too hard and complex for humans.

2.4 Security

2.4.1 IPsec

The Internet protocols were not specified with security properties in mind. At the time of the ARPANET, the whole network was so small that practically every operator of the computers connected could be traced. Misuse was non-existent or minimal. In the modern days, there are millions of hosts connected to the Internet. Security related breaches have become part of everyday life in the Internet.

One standardized security architecture is called as *IPsec (Security Architecture for the Internet Protocol)* [17]. “IPsec is designed to provide interoperable, high quality, cryptographically-based security for IPv4 and IPv6. The set of security services offered includes access control, connectionless integrity, data origin authentication, protection against replays (a form of partial sequence integrity), confidentiality (encryption), and limited traffic flow confidentiality. These services are provided at the IP layer, offering protection for IP and/or upper layer protocols.”

The IPsec specification discusses also components of the architecture which are needed for the defined functionality. The main components are Security Protocols (*Authentication Header (AH)* [16] and *Encapsulating Security Payload (ESP)* [18]), Security Associations, Key Management, and Algorithms for authentication and encryption.

The overall idea of IPsec is relatively straightforward. IPsec implementations define separate Security Policies for inbound and outbound traffic. When a packet arrives at or is about to leave the host, the IPsec implementation consults the Security Policy Database (SPD), and decides the fate of the packet based on the policy in the SPD whether the packet is allowed to pass or not. Third case is that the packet can be allowed to bypass IPsec completely.

“A *Security Association* (SA) is a simplex ”connection” that affords security services to the traffic carried by it. Security services are afforded to an SA by the use of AH, or ESP, but not both. If both AH and ESP protection is applied to a traffic stream, then two (or more) SAs are created to afford protection to the traffic stream. To secure typical, bi-directional communication between two hosts, or between two security gateways, two Security Associations (one in each direction) are required.

A security association is uniquely identified by a triple consisting of a Security Parameter Index (SPI), an IP Destination Address, and a security protocol (AH or ESP) identifier. In principle, the Destination Address may be a unicast address, an IP broadcast address, or a multicast group address.” [17].

An IPsec implementation needs also to store other related information in the A Security Association in order to encapsulate and decapsulate the IPsec packets. Other important fields include source address, algorithms used, secret keys, and IPsec mode.

2.4.2 Effects of Mobility and Multihoming to IPsec

A simplified SA could also be represented as the following triple:

$\{IP\ source\ address, IP\ destination\ address, SPI\}$

The triple describes a SA for any traffic going from the source address to the destination address, and having the given SPI in the packet header. The SPI value can then be used as an index for looking up the rest of the SA information.

The problem here is that mobility and multihoming have implications on the use of SAs. When a host moves to another location, its address changes and the previously set up SA is not usable anymore, because the SA was bound to the previous address. The connection might then go unencrypted or be blocked. The same applies also when a host has more than one address, the same SA should protect traffic sent through all the addresses.

A proposed conceptual change to the interpretation of a SA is the triple:

$\{\{list\ of\ IP\ source\ addresses\}, \{list\ of\ IP\ destination\ addresses\}, SPI\}$

The interpretation of this is that one SA covers all traffic coming from some if the listed source addresses and destined to some of the listed destination addresses. The SPI means the same as before.

An IPsec implementation must therefore handle these situations correctly, so that the right SA is selected for all address combinations and the connections work without any interruptions during the address changes.

Chapter 3

Problem Statement

3.1 Research Problems

Mobility, multihoming, and security will surely raise issues which are not clearly seen during developing a new network protocol. When trying to add several separate functional areas into a single protocol, things get complicated. Even if the design phase of a protocol would be successful, it is not enough if the purpose of the design work is to really gain real life popularity. For the protocol to be adopted widely, it must also be proven that the protocol is feasible to implement.

The outcome of the implementation work is beneficial no matter what the result is. If the implementation work can not be done in a reasonable time or the implementation depends on too many external parts, the design is shown to be too complex, and therefore needs redesigning. A well defined protocol is straightforward to implement and the development work can be divided in separate areas. A typical division could be the functional parts of the protocol, for example. Both results share the common benefit: they give insight and experience to future protocol designers and software developers what kind of a design is possible to implement.

This thesis studies the **Host Identity Protocol (HIP)**. We have selected HIP because it gives us a concrete base for the research topics and the rest of the discussion of this thesis. HIP will be presented in more detail in chapter 5.

The two main topics what this thesis focuses on are:

- *Issues when implementing a networking protocol providing*

identifier/locator split

- *Implications of identifier/locator split on mobility and multihoming*

3.2 Elaboration of the Research Problems

3.2.1 Identifier/Locator Split

The concept of the *identifier/locator split* is something that networking stacks of the operating systems do not, or even can not, support by default without any modifications. In order to support it, there must be a new kind of a logic in the network stack which handles the translations between the identifiers and locators (and vice versa). Plain translation service is not adequate for an usable implementation. Other support functionality is also required, such as handling of addressing related events.

In general, an implementation should not cause any changes to the accustomed use of the upper layer protocols. For example, because the identifiers are typically not real routable addresses, the implementation must take into account how to deal with this issue so that the application can just rely on the network stack to forward the packets to the ultimate destination. In practise, the higher protocol layers, mainly the transport layer, are affected in some level. The main actors we are focusing on in transport layer are the TCP and UDP protocols.

However, an implementation may cause changes in the non-functional properties of an upper layer protocol. For example, it is interesting how TCP, which is a connection oriented protocol, behaves when the connection endpoints are stable but some parts of the underlying connection changes without telling it to the transport layer.

Application level issues are out of scope of this thesis. A thorough example of an API design is presented in [19]. Security analysis is also out of scope of this thesis.

3.2.2 Implications of Identifier/Locator Split on Mobility and Multihoming

As seen in the background chapter, mobility and multihoming are very complex issues. If a solution providing both of these is added with security enabling features, the overall complexity of the protocol raises again to a higher level.

Another focus of this thesis is to address the implications on mobility and multihoming when IPsec is used at the same time.

At this point we give only a short description on the problem field. A more detailed discussion is in the following chapters.

The basic IPsec scenario is that there is one IPsec Security Association for both directions between the two hosts. As stated in section 2.4.2, multihoming needs changes to the handling of the IPsec. Because HIP uses the functionality provided by the IPsec, we again use HIP as the basis for the discussion related to the implications on multihoming.

The main reason for the performance drawback is the replay window of the ESP. When the ESP packets arrive out of replay window they are dropped. This happens when packets are sent simultaneously using several paths. The HIP base protocol [22] defines only one IPsec ESP Security Association by default for all HIP data traffic. If the implementation has support for multihoming and only one Security Association is used for all addresses, it will lead to bad performance due to different QoS characteristics of the paths. One aim of this thesis is to answer to the question *“If one SA is not enough, how to avoid these problems with multiple SAs ?”*.

Chapter 4

Related work

In this chapter we present briefly two schemes for managing mobility and multihoming. Both are developed at the IETF and the specifications are publically available.

Mobility is presented using the well known and standardized *Mobile IPv6 (MIPv6)*. *Hash Based Addresses (HBA)* is used as an example of (IPv6) multihoming. HBA is not yet standardized, but it shows how multihoming can be thought from a different point of view.

4.1 Mobile IPv6

[15] specifies a protocol which “allows nodes to remain reachable while moving around in the IPv6 Internet. Each mobile node is always identified by its home address, regardless of its current point of attachment to the Internet. While situated away from its home, a mobile node is also associated with a care-of address, which provides information about the mobile node’s current location. IPv6 packets addressed to a mobile node’s home address are transparently routed to its care-of address. The protocol enables IPv6 nodes to cache the binding of a mobile node’s home address with its care-of address, and to then send any packets destined for the mobile node directly to it at this care-of address. To support this operation, Mobile IPv6 defines a new IPv6 protocol and a new destination option. All IPv6 nodes, whether mobile or stationary, can communicate with mobile nodes.”

A mobile node capable of IPv6 will acquire its address automatically by either stateless or stateful Address Autoconfiguration which are standard IPv6 mechanisms. [33] specifies IPv6 Stateless Address Autoconfiguration while Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [9] is an example

of stateful autoconfiguration.

Even though the host would get its address with these methods, the transport layer connections will break when the host moves. Most probably majority of the networking applications do not handle the sudden change of addresses graciously. Connections would have to be reopened using the new address.

4.1.1 Mobile IPv6 Components

MIPv6 provides an alternative solution to this problem. MIPv6 specifies a concept of *Home Address*. Home address does not change even when the host moves to another network, so the host is always reachable through this address. Also, applications bind to this address so the transport layer binding will not break anymore. The mobile node has a home network, in which the home address resides.

Home Agent takes care of maintaining the registrations of the mobile nodes which are located outside their home network. It forwards the traffic to and from the MIPv6 node. The home agent is located in the home network. For discovering a home agent, Mobile IPv6 defines the Dynamic Home Agent Address Discovery protocol ([15]). The mobile node sends an ICMP Home Agent Address Discovery Request to a pre-defined Home-Agents anycast address. The home agent that receives the message will answer with a reply message including a list of home agents. The mobile node then registers to one of the home agents by sending a binding update.

When the mobile node moves to a foreign network, it will also be assigned a *care-of address*. *Correspondent node* is an IPv6 node communicating with the mobile node. The correspondent node can support MIPv6 or it can be a node not supporting MIPv6.

The association of a home address with a care-of address for a mobile node is known as a *binding*. MIPv6 capable correspondent nodes and home agents store the bindings in a *binding cache*. Mobile nodes maintain information about correspondent nodes in a *binding update list*.

4.1.2 Data Transfer in Mobile IPv6

When away from the home network, the mobile node has a few ways to receive packets from the corresponding node. First one is that the packets can be sent directly to the mobile node's home address if the correspondent node does not support MIPv6 or correspondent registration is not yet complete. When the home agent receives the packets destined to the mobile node, it tunnels

them over a IPv6-over-IPv6 tunnel to the care-of address of the mobile node. The second case is when the correspondent node supports MIPv6 and the correspondent registration has been completed, the correspondent node can send the packets directly to the mobile node. The packets sent include a new Type 2 Routing extension header that contains the mobile node's home address.

Very similar cases apply when the mobile node is sending packets to the correspondent node. An IPv6-over-IPv6 tunnel to the home agent is used when correspondent node does not support MIPv6 or correspondent registration has not been completed yet. The home agent forwards the packets to the correspondent node. When the correspondent node is MIPv6-capable and correspondent registration has been completed, packets are sent directly to the correspondent node. In this case the packets include Home Address option in a Destination Options header that contains the mobile node's home address.

There are two modes of routing, *route optimization* and *bidirectional tunneling*.

In *route optimization* only the first packet is tunneled through the home agent. Then a binding update is made between the mobile node and the correspondent node. The rest of the packets are delivered directly to the current care-of address. The correspondent node adds to the packets a routing header with the home address, while the mobile node uses the "Home Address" destination option for the home address. All this eliminates the transmission delays as much as possible.

Bidirectional tunneling means the tunneling of packets between the correspondent node and the mobile node via the home agent. Use of bidirectional tunneling ensures that the mobile node is always reachable, even if the correspondent node is not Mobile IPv6-capable. Tunneling is performed using IPv6 encapsulation. One downside is the additional delay bidirectional tunnel creates.

4.1.3 Route Optimization and Registration of Correspondent Node

Route optimization needs a registration process of correspondent node before the data can be sent between the mobile node and the correspondent node. A procedure for checking the reachability of a node is called *return routability* test.

To show that the mobile node is really at its home address and its care-of address, the correspondent node and the mobile node perform the return routability tests. These cryptographically protected tests also prevent some

denial-of-service and connection hijacking attacks.

The procedure starts when the mobile node sends two different test packets to the correspondent node, the first packet goes through the home agent and the other is sent directly to the correspondent node. When the correspondent node receives the test packets, it replies to them with packets containing a cryptographic token proving the authenticity of the response packets.

After completing the return routability tests, the mobile node sends a Mobile IPv6 Binding Update message to the correspondent node. This message contains authentication data based on the tokens sent by the correspondent node in the previous step. When the correspondent node receives the packets, it can validate the authentication data. On successful validation, the correspondent node can create an entry to its binding cache containing the information on the mobile node. Finally the correspondent node sends a Binding Acknowledgement message back to the mobile node. Similarly, when the mobile node receives the Binding Acknowledgement, it can add an entry related to the correspondent node to its own binding cache. At this point the nodes can send data using the route optimization.

4.2 Hash Based Addresses

[5] describes a mechanism to “provide a secure binding between the multiple addresses with different prefixes available to a host within a multihomed site. The main idea is that information about the multiple prefixes is included within the addresses themselves. This is achieved by generating the interface identifiers of the addresses of a host as hashes of the available prefixes and a random number. Then, the multiple addresses are generated by appending the different prefixes to the generated interface identifiers. The result is a set of addresses, called *Hash Based Addresses (HBAs)*, that are inherently bound. A cost efficient mechanism is available to determine if two addresses belong to the same set, since given the prefix set and the additional parameters used to generate the HBA, a single hash operation is enough to verify if an HBA belongs to a given HBA-set.”

In HBA, information about the multiple prefixes available to a multihomed host is encoded in the interface identifier part of the IPv6 address. There are also other schemes which store cryptographic information in the interface identifier part of the address. [4] describes a method for “binding a public signature key to an IPv6 address in the Secure Neighbor Discovery (SEND) protocol. *Cryptographically Generated Addresses (CGA)* are IPv6 addresses for which the interface identifier is generated by computing a cryptographic

one-way hash function from a public key and auxiliary parameters. The binding between the public key and the address can be verified by re-computing the hash value and by comparing the hash with the interface identifier. Messages sent from an IPv6 address can be protected by attaching the public key and auxiliary parameters and by signing the message with the corresponding private key. The protection works without a certification authority or any security infrastructure.” To avoid compatibility issues between HBA and CGA, an important design aim of HBA is that both can be used.

4.2.1 HBA address set

Typically the addresses of a multihomed host are known in advance, changes to available address prefixes happen very seldom. HBA assumes that this address set stability provides a secure binding between all the addresses of the node. This is why no new prefixes can be added later to the original HBA-set.

The CGA is inherently bound to a public key, while a HBA is inherently bound to a prefix set. This means that a public key is not strictly required to generate an HBA. That way the mechanism provides a cost efficient alternative to public key cryptography based schemes.

The HBA generation process is based on the CGA generation process defined in section 4 of [4]. The goal is to require the minimum amount of changes to the CGA generation process. The changes required in the CGA generation process when generating a single HBA are the following: First, the Multi-Prefix Extension has to be included in the CGA Parameters Data Structure. Second, in the case that the address being generated is an HBA-only address, a random nonce will have to be used as input instead of a valid public key [5].

The inputs to the HBA generation process are the list of 64-bit prefixes, a security parameter Sec, and also a public key if HBA/CGA address is to be generated. The output of the process is an HBA-set and their respective CGA Parameters Data Structures.

HBA verification needs an HBA-set and a CGA Parameters Data Structure. Verification starts by checking that the 64-bit HBA prefix is included in the prefix set of the Multi-Prefix Extension. If it is included, replace the prefix contained in the Subnet Prefix field of the CGA Parameter Data Structure by the 64-bit HBA prefix. Then the verification process described in section 5 of [4] is run with the HBA and the new CGA Parameters Data Structure as inputs.

Chapter 5

Host Identity Protocol

As discussed previously in the problem statement chapter, we have decided that *Host Identity Protocol (HIP)* is our target for studying the defined research topics.

In this chapter we present a general overview on what HIP is and how HIP works based on the current base HIP specifications ([23], [21]). The scope is on the base protocol functionality only, all other specifications extending the base protocol functionality are out of the scope except when they are closely related to the discussion and need to be presented at least on a high level. *End-Host Mobility and Multihoming with the Host Identity Protocol* [10] is an example of a specification extending the base protocol.

5.1 Introduction to Host Identity Protocol

The core of the Internet is the *Internet Protocol (IP)* [28]. IP uses IP addresses to transmit data from a source address to a destination address. An IP address can be said to have two roles at the same time: an address specifies an *endpoint identifier* of a node (eg. name of a network interface) and also a *locator* to reach the node (eg. the actual address of the node). This has led to architectural difficulties as discussed in more detail in the chapter 2. Several new protocols and mechanisms have been proposed to fix these shortcomings, Host Identity Protocol being one among them.

“HIP allows consenting hosts to securely establish and maintain shared IP-layer state, allowing separation of the identifier and locator roles of IP addresses, thereby enabling continuity of communications across IP address changes. HIP is based on a Sigma-compliant Diffie- Hellman key exchange, using public-key identifiers from a new Host Identity name space for mutual peer authentication.

The protocol is designed to be resistant to Denial-of-Service (DoS) and Man-in-the-middle (MitM) attacks, and when used together with another suitable security protocol, such as Encapsulated Security Payload (ESP), it provides integrity protection and optional encryption for upper layer protocols, such as TCP and UDP.

The HIP architecture proposes an alternative to the dual use of IP addresses as "locators" (routing labels) and "identifiers" (endpoint, or host, identifiers). In HIP, public cryptographic keys, of a public/private key pair, are used as Host Identifiers, to which higher layer protocols are bound instead of an IP address. By using public keys (and their representations) as host identifiers, dynamic changes to IP address sets can be directly authenticated between hosts and if desired, strong authentication between hosts at the TCP/IP stack level can be obtained." [23]

5.2 Host Identity Namespace

HIP proposes a method to separate these end-point identifier and locator roles of IP. From this it follows that HIP also implicitly proposes to change the current architecture of the TCP/IP stack.

HIP introduces a new namespace called as the *Host Identity* namespace. "A name in the Host Identity namespace, a *Host Identifier* (HI), represents a statistically globally unique name for naming any system with an IP stack" [21].

A Host Identifier is cryptographic in its nature. A HI is typically based on public-key cryptography, and the keys can be self-generated. This HI is then the public key of an asymmetric key-pair. In theory, any kind data that is statistically unique could be a HI, but in practise cryptography-based ones have the best applicability. In HIP architecture, hosts are identified with these public keys and not with IP addresses.

A HI can not be used as a routable address, because they have no hierarchy. Using the cryptographic HI namespace ensures that the HIP data transfer communications is secure. Actually, as soon will be seen, HIP integrates security, mobility, and multihoming in one clean solution package.

5.2.1 Host Layer

This new HI namespace requires a new protocol, HIP. The HIP protocol is used for exchanging namespace identities and continuity between those hosts

independent of the networking layer [21]. HIP protocol payload is transferred on a conceptual layer called as the *Host Layer*, which is located between the network (typically the IP layer) and the transport layer (eg. TCP or UDP).

5.2.2 Mobility and Multihoming with HIP

In the standard networking applications using the Internet Protocol, the higher layer protocols (such as TCP) statically bind themselves to the IP layer. The applications can directly see the IP addresses of the remote peer node when sending or receiving data. The problem here is the case when either node, or both, moves. The problems and implications of this mobility case were discussed in chapters 2 and 3.

In HIP, the transport layers used by the applications bind to this new Host Layer instead of binding to an IP address. However, the application data needs to be transferred anyway, but the HI is not routable and therefore the data can not be sent right away. The solution to this is that the HIP design specifies that a HI can be associated with multiple IP addresses at the same time. A host can decide what network interfaces, and therefore what addresses of the interface, it wants to use for binding the HIs to. These *<HI-IP address>* associations are added, updated, and deleted dynamically and transparently to the user. When a data packet is sent to a HI, a destination IP address is selected among the associated IP addresses, and the packet is sent to the selected IP address.

The HIP specifications also define that a host can store the address mappings of the peer host. A host can select what addresses it wants to tell to the other party. With these two things in mind, a host has a list of all the available addresses of the peer it can use for sending data. If an address becomes unusable, the host can run a deterministic algorithm which selects the next best reachable peer address and sets that as the active destination address.

Having these items as the main characteristics, HIP provides a mobility and multihoming solution. Details of mobility and multihoming with HIP is out of the scope of this thesis, but interested readers might read [10].

5.2.3 Representations of a Host Identifier

Host Identifiers can not be used as such in protocols. Many protocols use fixed size value in their data structures where an identifier value could be used. HI has no fixed size, and it could be almost of any length. There are also different public key algorithms that can be used with different key lengths.

Therefore other representations of a HI are needed. *Host Identity Tag (HIT)* is a 128-bit long cryptographic hash of the full HI. The hashing method used is SHA-1 ([27]). HITs are used in the HIP packets as the source and destination of the packet.

A HIT has also the same size as an IPv6 address has. Due to the same size, HITs could easily be used eg. in place of IPv6 addresses in the networking APIs and protocols. Hashing has also another advantage, it presents the identity in a consistent format independent of the cryptographic algorithms used.

There have also been defined *Local Scope Identifier (LSI)* which is a 32-bit localized representation for a Host Identity. The main purpose of an LSI is to facilitate using Host Identities in existing protocols and APIs. For example in IPv4-based protocols and APIs (IPv4 addresses are 32 bits long).

5.3 The HIP Base Exchange

A *HIP association* is set up between hosts using the *HIP base exchange*. This cryptographic exchange is used to establish an IP-layer communications context between the hosts. The context is needed to store all information related to the connection, such as the Host Identity used during the connection.

The base exchange is a four packet exchange. The party which starts the base exchange is called the *Initiator*, and consequently the other party is the *Responder*.

Base exchange is quite similar to the TCP connection establishment procedure [29]. When the Initiator wants to initiate the HIP base exchange, an *I1* packet is sent to the Responder. The packet contains only the HIT of the Initiator and possibly the HIT of the Responder, if it is known (if the Responder HIT is not known, this is called as *opportunistic mode*). I1 packet might be spoofed, so Responder does not perform any time consuming operations on this packet.

Second packet sent by the Responder, *R1*, is sent as a reply to the I1 packet. R1 contains the public value of Diffie-Hellman key [31] of the Responder, connection related information such as supported encryption algorithms of the Responder, and finally a signature covering part of the packet data to avoid packet replay attacks.

Additionally, R1 contains a puzzle which the Initiator must solve in order to make a successful base exchange. Puzzles are designed in such a way that solving them requires significantly more time than creating them. If Responder has many simultaneous connection attempts, it can simply create a harder puzzle. Responder could also set puzzle difficulty based on its level of trust of

the Initiator.

Puzzles are used in an attempt to diminish the effect of *Denial of Service* (DoS) attacks. DoS attacks can cause huge performance hit to the target, so it is very important to at least try to avoid them. Precomputed puzzles and re-using R1 packets for a short period of time provide additional security for the Responder.

Third packet, *I2*, contains Initiator's public value of Diffie-Hellman key, Initiator's selection of encryption algorithm supported by the Responder, Host Identity of the Initiator (encrypted using the selected encryption algorithm), Security Parameter Index (SPI) for Responder's ESP traffic, and the calculated answer to the puzzle. Signature of the packet is appended to the packet.

The fourth and the last packet is *R2*, which contains Initiator's SPI and signature over the packet.

The result of a successfully finished base exchange is that the hosts have authenticated themselves to each other and created bidirectional IPsec ESP Security Associations for the HIP related traffic. Right after the base exchange packets carrying user data can be sent securely over the ESP channel.

Timeline of the HIP base exchange can be seen from the figure 5.1.

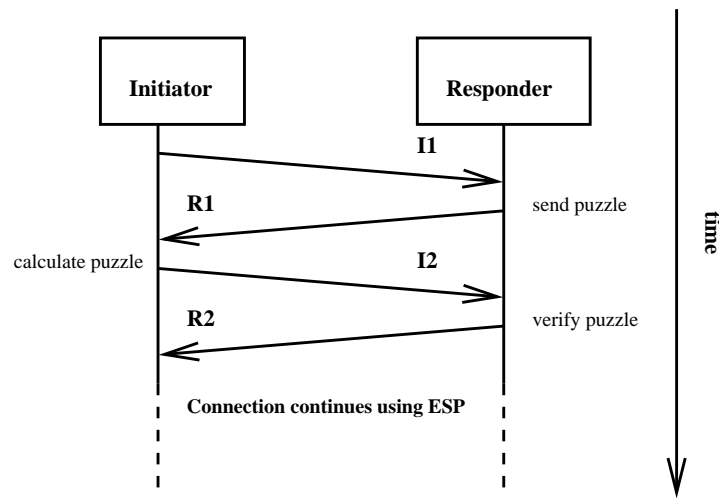


Figure 5.1: The HIP Base Exchange

Chapter 6

Design of Implementation

This chapter describes the relevant parts of the operating system kernel which a HIP implementation is mainly concerned about.

We start by presenting the architecture of the relevant parts of the operating system kernel. After that we point out the places in the kernel which must be changed in order to support the basic HIP protocol functionality and also support for multiple IPsec Security Associations (SAs) in HIP. That is followed by a short requirement analysis on what is needed for implementation of multiple IPsec SA support in HIP. Then we present possible ways for grouping the IP addresses into these multiple IPsec SAs. After that, the natural continuation in the discussion is how to basically select the outbound SA and how to improve the selection process.

6.1 Architecture of the Linux Networking Stack

A typical HIP implementation should not have any need to interfere with all the important tasks an operating system performs, such as process management and scheduling, file system operations, and memory management. Because HIP is a networking protocol, the emphasis is on the implementation of the networking stack (TCP/IP in particular), so the other areas are out of the scope and therefore are excluded from this thesis.

At the time of writing this thesis, the HIPL implementation supports only the IPv6 protocol for sending and receiving the HIP control messages and HIP related data traffic encapsulated in ESP packets. Therefore the focus of the networking subsystem is on IPv6 and IPsec related functionality. Other relevant parts are described only where necessary.

Architecture description is split into distinct parts, each being responsible for certain specific high level functionality. Networking in general can be abstracted into two major areas, sending data to the other hosts and receiving data sent from the other hosts. These two are not sufficient alone for an usable operating system, other related support functions are needed too. Examples of such support functions are network device registration, registration of protocols on the internetworking layer and transport layer, and changing the state of networking devices, protocols, or network sockets used in the applications.

The network stack is presented through an example of two simple networking applications, a server application, and a client application which uses the services provided by the server. The C language sources for both of the applications are listed in appendix A.

The example scenario is as follows. Both applications use TCP as the transport layer protocol on top of IPv6. The server starts up by creating a way for other hosts to be able to connect to the server by setting up a service which is reachable at its TCP port 2000. When it is set up, the server starts to listen to incoming connection requests. The client application tries to connect to the server. If the connection request to the server succeeds, the client sends some data to the server over TCP. When a connection is accepted in the server, the server reads the received data sent by the client, and closes the client connection. The server finishes by stopping its service. After the client has sent its data, it closes the connection.

6.1.1 Output Packet Processing

In order to communicate with the other host, a host must first create a communication endpoint called a *socket*. Sockets provide a standard protocol-independent interface between the application and the networking stack of the operating system. An application creates a socket using the *socket* system call.

When the socket is created, the host initiates a TCP connection to the peer host by calling the *connect* system call. From this point on, the operating system handles all of the lower level details needed for the connection setup, such as setting up resources for the transport layer socket and creating, sending, and receiving the packets.

Because the example code in the appendix A for the client uses IPv6, the *connect* call is handled by the *tcp_v6_connect* function.

To speed up the data transfer, the *tcp_v6_connect* function creates a *destination cache* entry (called *dst_entry* in Linux) by calling the *ip6_dst_lookup* function. This cache entry is then stored into the socket by *ip6_dst_store* function. The

entry is referenced whenever packets are sent out to the peer. The destination cache improves overall performance by several means, such as storing the network interface used for the packet transmission, routing related information (e.g. router that forwards the packet onward), hardware header inserted into the link layer header, and functions used when packets are sent or received.

Finally, at the end of the *tcp_v6_connect* function, call flow continues to function *tcp_connect* which handles sending of TCP connection requests for both IPv4 and IPv6 sockets. Again, *tcp_connect* calls function *tcp_transmit_skb*, which in turn calls an address family specific function for queuing the packet for sending. In the case of IPv6, the function is *tcp_v6_xmit*. The TCP SYN packet is forwarded to IPv6 layer by calling the function *ip6_xmit*.

ip6_xmit fills in the IPv6 header values to the packet. Most important values include the source and the destination addresses. At the end of the function, packet is passed to the Netfilter [24] hook. The hook calls *ip6_maybe_reroute* which checks if the route cache is still valid.

The last step in the output processing is handing off the packet to the network. Function *dst_output* does this at the end of the *ip6_maybe_reroute* function. After this point it is the responsibility of the network interface to deliver the packet onward.

6.1.2 Input Packet Processing

Next we take a look what happens when a packet is received from the network. First, the network interface card receives the packet physically through the wire. The network device gathers the signaling into an entity called a frame.

When a frame is successfully received by the network card, the card raises a hardware interrupt, so the operating system knows that a packet is received and available for processing. The network device driver software allocates memory for the data and other related information, such as time of the reception, network device interface which received the packet, and such. All this information is stored in a *socket buffer*, which is in Linux called as *sk_buff*.

The interrupt handler of the network driver calls the function *netif_rx*. *netif_rx* appends the received *sk_buff* into the general incoming packet receive queue. Soon after that a *softirq* NET_RX_SOFTIRQ is raised. The purpose of this *softirq* is to process the queued input packets.

When the *softirq* NET_RX_SOFTIRQ is raised, the function *net_rx_action* gets called. *net_rx_action* loops through all the packets in the receive queue, and puts them into the incoming queue of the network device which originally received the packet. After this the input packets are processed in the func-

tion *process_backlog* which provides flow control so the processing does not reserve the CPU for a too long time. Call flow goes on from *process_backlog* to *netif_receive_skb* whose responsibility includes passing the packet to the network layer based on the protocol value in the protocol field of the frame.

Because the example scenario uses IPv6 as the network protocol, *ipv6_rcv* gets called to start handling the IPv6 packet. After the initial checks are made, the packet goes through Netfilter to the function *ip6_rcv_finish*. *ip6_rcv_finish* routes the packet and sets the destination cache entry for this packet. The destination cache entry contains e.g. the function which will process the packet next.

If the destination of the route decision is local, the next called function is *ip6_input*. Practically this function calls *ip6_input_finish* through the Netfilter mechanism. *ip6_input_finish* is the real point where the packet is again passed to the higher layer, the transport layer.

In this example case the transport layer is TCP. TCP protocol handler for IPv6 packets is *tcp_v6_rcv*. After the TCP has performed the required checks and looked up the correct socket which is related to the connection, the packet is added to the receive queue of the socket. Now the application can issue the *read* system call to retrieve the data from the socket.

6.1.3 Related Support Functionality

The above described packet processing paths for inbound and outbound packets are not enough to cover all the issues required to have the understanding on the topics related to the discussion. These other areas include at least *IPsec ESP processing*, *IPv6 addressing*, and *network related event handling*. An usable HIP implementation needs these in order to work efficiently.

Not all of these are called directly during the packet processing. Some are called asynchronously, most notable example being the addressing events. Next we will see how these issues are related to a HIP implementation.

IPsec ESP processing

HIP encapsulates payload data into IPsec ESP packets. ESP processing can be divided into two distinct parts, encryption and decryption of data. Encryption happens on outbound traffic processing and decryption on inbound traffic processing.

IPv6 ESP input is handled by the function *esp6_input*. This function is really simple. First, the function checks validity of the ESP header. After the

checks the packet data is decrypted, so after the return the operating system can continue the processing by passing the clear-text data to the appropriate transport layer handling function.

Accordingly, the function *esp6_output* handles IPv6 ESP output. The correct IPsec SA used in the data encryption is fetched from the *dst_entry* cache. Before the data is encrypted, the ESP header is constructed. One of the important values of the header is the SPI field, which is taken from the cached SA. The other fields include the sequence number used for replay protection.

IPv6 Addressing

Because The HIP mobility and multihoming draft [26] deals with addressing issues, there is a clear need for any HIP implementation to know when the host is performing some level of addressing related operations.

Mobility is the typical scenario when a host receives new addresses to be used in the network the host moved into. Static hosts usually setup their addresses during the system boot, but they might still encounter a situation where the address of the host changes, e.g. when using DHCP ([8], [9]).

A host may also have multiple network interfaces or a network interface has multiple addresses. This scenario is called as multihoming. Network interfaces and the addresses associated with them may be added or deleted dynamically.

In the IPv6 networks, a host may also configure its network interface address(es) using the IPv6 Stateless Address Autoconfiguration specified in [33]. This specification defines how to create a link-local address based on locally available information (such as address prefixes advertised by the routers) and verify its uniqueness on a link using the defined *Duplicate Address Detection* (DAD) procedure. It also specifies how to configure site-local and global addresses statelessly.

“In the stateful autoconfiguration model, hosts obtain interface addresses and/or configuration information and parameters from a server. Servers maintain a database that keeps track of which addresses have been assigned to which hosts. The stateful autoconfiguration protocol allows hosts to obtain addresses, other configuration information or both from a server. Stateless and stateful autoconfiguration complement each other. For example, a host can use stateless autoconfiguration to configure its own addresses, but use stateful autoconfiguration to obtain other information.” [33]

Linux implements the DAD procedure when adding a new IPv6 address. The newly added address is not usable until it is verified, even though the address seems to exist in the interface. This is due to the delays caused by the DAD.

When an IPv6 address is added manually e.g. using the commonly available *ifconfig* tool, the function *inet6_addr_add* is called. *inet6_addr_add* checks that the network interface into which the address is to be added is up, and only after that the address is added (by calling the function *ipv6_add_addr*). After successful addition the DAD procedure is started for the address.

Manual IPv6 address deletion of an interface is handled by the function *inet6_addr_del*. It simply calls *ipv6_del_addr* which does the low-level removal of the address from the operating system data structures. *inet6_addr_del* also checks the special case when no more IPv6 addresses remain. In that situation IPv6 is disabled on that particular interface.

Events in the Operating System

Notification on change of the status in a high level kernel facility is implemented in Linux using the mechanism called *notifier call chain*. In practice, a notifier chain is simply a list of function callbacks, which all get called when an event has occurred. The called function is provided with the event type and possibly some extraneous data related to the event. The notifier chains are defined at compile time.

Notification on IPv6 address and network interface state changes are also implemented using the notifier call chains. Typically the Linux kernel defines an unique notifier chain used for IPv6 address addition and deletion events. Those parts of the implementation interested in receiving these notifications must first register to the corresponding notifier chain and unregister when it is not anymore interested in the event processing.

Similarly, there are events when a network interface is set up or down. In addition to those two network interface level events, the operating system issues an event also when the interface is being registered or unregistered. The registration is performed when the operating system notices the existence of a new device. The unregistration process is done prior to removing the interface completely from the system. The purpose of unregistration is to prepare the device for safe removal and free the resources which were allocated during the registering. The difference between a *down* and an *unregister* event is that the *down* event does not remove the information on the interface, the device still exists in the kernel data structures.

A typical Linux HIP implementation is likely to register to both chains, because (for example) a periodic polling based approach for address status changes is not feasible.

6.2 Required Changes to the Linux Operating System Kernel

Now it is time to show why the current Linux operating system code base needs changes and where the changes are located when adding support for HIP and multiple SA support.

High level classification of problem areas and other HIP implementation related areas are:

- Network Input and Output
- Storing of IPsec SAs
- Caching of Data Structures
- Issues on Lower and Higher Layers
- Event Handling

6.2.1 Network Input and Output

As the HITs are not routable as such, a conversion from HITs to routable IPv6 addresses is needed. When the kernel notices that packets having a HIT as the destination address are about to be sent out, the HIP implementation gets an usable corresponding IPv6 address of the peer. The IPv6 is selected from the address set which the peer has previously advertised to us. After that the kernel selects a source IPv6 address and sends the packet out. Potential places in the kernel to modify this behavior is in the functions *ip6_xmit* (for TCP) and *ip6_push_pending_frames* (for UDP).

As expected, on input side of the network traffic the IPv6 headers contain IPv6 addresses. For HIP data traffic, the operating system need to check the ESP packets and then decide based on the SPI value whether the packet belongs to a HIP connection or not. If it does, then the source and destination IPv6 addresses are replaced by the corresponding source and destination HITs.

6.2.2 Storing of IPsec SAs

An obvious minimal requirement for the underlying IPsec implementation is that it supports multiple and simultaneous IPsec SAs for source *source / destination / protocol* pairs. For HIP implementations which use HITs in the

address fields of an IPsec SA, this requirement is clearly seen when a host configures a new network interface and advertises that to the peer hosts using the method described in the host multihoming scenario in the HIP mobility draft.

For outbound traffic the HIP implementation must be able to tell the IPsec layer which SA it should use when there are more than one matching SA available.

For inbound traffic there are no noticeable requirements, because the right IPsec SA will be found based only on the SPI field of the ESP header even though there would be more than one matching SA for the *source / destination / protocol* pair.

6.2.3 Caching of Data Structures

The Linux kernel caches some commonly used data structures into a socket. These caches have a major impact on overall performance during the data transmission phase. The major cache structure in the socket is the destination cache, *dst_cache*. Along with all kinds of other cached information about the peer host, there is also pointer to the IPsec SA used for this connection.

This causes a major problem for implementations supporting multiple IPsec SAs. When the implementation decides to send data to an IPv6 address which is associated with a different IPsec SA than the previous preferred IPv6 address was, the cache becomes invalid. This leads into problems, for example, when the outgoing data may be encrypted using the keys from the old IPsec SA and perhaps only part of the data ESP needs for encryption is valid. At the responder side, the ESP packet will be dropped due to failed decryption.

Connected sockets, such as TCP, are assumed to be affected by this issue more than not connected sockets (UDP, for example). The reason for this is that connected sockets create this cache only once upon creation, and not connected sockets create it for each packet. When making a not connected socket to a connected socket (e.g. calling the system call *bind* for an UDP socket), the problem is assumed to exist also in that case.

This issue could be fixed by resetting the cached IPsec SA from the destination cache of the socket in question and doing a IPsec SA cache relookup when the HIP layer notices that it is about to change the preferred destination address.

6.2.4 Issues on Lower And Higher Layers

The HIP documents do not define any requirements on lower layers below the network (IP) layer. Lower layers, using the classical OSI layering model [13], are the link layer and the physical layer. This means that in theory and also in practice, HIP related packets can be transmitted over any networking technology over which IP packets can be sent.

Higher layer handling needs changes, because the transport layer binds to the HITs instead of IP addresses. The implementation must take into account the requirements for the additional functionality provided by HIP compared to the plain TCP. For example, the connection setup takes more time than the TCP connection setup, if there is no previous HIP association between the hosts. The HIP association must complete first before any other data transfer to the peer host can occur. This will add some delays, which are also seen in the application. To prevent the possibility of failing when the application tries to send data, the HIP implementation should freeze the transport layer when the HIP layer needs to setup the association. For example, in the case of a TCP connection, the socket sending the SYN packet should be put to sleep when the base exchange is needed. When the base exchange is finished, the socket is woken up, and the SYN packet is sent off. However, this forced sleeping for an undefined period of time will cause problems with some protocols, most notable example being TCP and its retransmission timers, which might fire off during the base exchange, possibly causing multiple SYN packets to be sent.

Layers above the transport layer are not a concern of HIP, because the transport layer handles them indirectly.

6.2.5 Event Handling

On IPv6 address addition, the kernel issues immediately an IPv6 addressing event when the function *ipv6_add_addr* is about to return. It is possible that the DAD procedure does not complete successfully. If an HIP implementation assumes that all added addresses will be valid, other peer hosts may receive non-reachable addresses and then try to validate them with no success.

To avoid this resource consumption, a better idea is to start readdressing only when the DAD procedure completes successfully. This needs modifications to the function *addrconf_dad_completed*.

On IPv6 address deletion, the event is also sent immediately. On deletion, there is no need to wait for anything, because the address is assumed to be unusable at the time of the reception of the event.

Theoretically, the network device events are rather simple. When a device comes up, there is not much to do because the device does not have any addresses yet, so it is better to wait for the IPv6 addressing events.

A network interface going down may have multiple configured addresses. One part of the tasks that the kernel does when a device goes down is to delete all the addresses of the interface. If there are multiple addresses, there will be multiple address deletion events. On those implementations that advertise the other peers the current address list whenever an address is deleted, the amount of readdressing UPDATE packets will become large. The receiver sees the packets almost simultaneously, and might even drop them. An other problem is that the packets might be handled in different order than the packets were sent, which will cause inconsistent information on the receiver side. These are good examples of the major reasons why any implementation must know how to make a difference on IPv6 address deletion event and network interface down event.

6.3 Grouping of IP Addresses into IPsec Security Associations

When implementing the HIP mobility draft, the first major problem that the implementor faces is the grouping of the IP addresses of the host into related IPsec Security Associations as described in mobility draft section 4.1. The draft does not give explicit instructions on how to perform this address grouping in such a way that it will be feasible to an average mobility use case and its performance characteristics are adequate.

6.3.1 Requirements

Even though it is quite impossible to define the requirements for every imaginable user requirement, some high level requirements can be defined as follows:

- **Ease of implementation work** Complex design usually causes complex implementations. Complex implementations have a high rate of possible hard to find bugs. Complexity may even ruin the performance on rare cases. Simpler implementation is also much easier to maintain and expand when necessary. These issues are however related to general software design, and they are out of scope of this thesis.
- **Performance** No matter what kind of a design is defined, it should have predictable performance characteristics when used in a constant

environment. Even though if the design would be a really inefficient one, the users know what to expect. If a well performing design in theory is practically implemented improperly, users might notice often significant fluctuations in the data transfer and then assume that something is not working correctly.

The operating system should not need any complex IP address selection algorithms or data structures when creating or updating a group.

The implementation should avoid sending of updates to the peers on changes of IPsec SA states when it is not necessary. For example, if local policy instructs that when a virtual network device is brought up, no new IPsec SAs are created for it and other current IPsec SAs are not changed in any way. From the viewpoint of the peers the situation is the same as before, so no HIP UPDATE packets are sent.

When the implementation decides that an update on the IPsec SA state must be informed to the peers, the notification should be constructed in an efficient way. Efficiency means here the required resource usage when selecting the IPsec SAs which are notified, selection of addresses into the SAs, creating the UPDATE packet, and preparing for using the new SAs. On the receiving side, resources are needed for processing the received UPDATE packet, possibly updating its own SAs, preparing for using the newly received SAs, and sending an acknowledgement UPDATE packet.

- **Predictability** This is related to the previous performance requirement. Previous user experience on an usage situation can be used as a reasonable base when a similar usage case is planned and its requirements need to be specified beforehand.
- **Conform to the specifications** Hosts implementing some custom address grouping algorithm must still be able to interoperate with the hosts that do not implement the same algorithms.

Hosts should not assume some extra functionality not specified in the draft unless the peer explicitly tells them that they really support the requested functionality. In any other case a default fallback scheme supported by both hosts must be used instead.

Given these high level requirements, next we try to propose some alternatives to the address grouping methods.

Address Grouping Methods

- **1 SA per 1 IP address** This is the simplest case. Every IP address is assigned its own SA. This should be straightforward to implement, but the downside is that each SA needs separately allocated resources when a new address is added. Processing takes also longer because the operating system has to do a lot of work to fetch the correct SA from the list of all known SAs. This design approach might not scale well on busy hosts having several addresses.
- **1 SA per 1 network interface** This is the only advice on the address grouping issue that the draft suggests. Quoting the draft: “A basic property of HIP SAs is that the inbound IP address is not used as a selector for the SA. Therefore, in Figure 2, unnecessary for address31, for example, to be associated only with SPI3 – in practice, a packet may arrive to SPI1 via destination address address31 as well. However, the use of different source and destination addresses typically leads to different paths, with different latencies in the network, and if packets were to arrive via an arbitrary destination IP address (or path) for a given SPI, the reordering due to different latencies may cause some packets to fall outside of the IPsec ESP anti-replay window. For this reason, HIP provides a mechanism to affiliate destination addresses with inbound SPIs, if there is a concern that replay windows might be violated otherwise. In this sense, we can say that a given inbound SPI has an “affinity” for certain inbound IP addresses, and this affinity is communicated to the peer host. Each physical interface SHOULD have a separate SA, unless the ESP reordering window is loose.”

The advantage of this approach is that the data traffic has quite well predictable characteristics if the peer uses fixed source address and any of the addresses within the SA.

This design should also be quite simple to implement. The only notable change to the previous method is the need to determine to which network interface an address belongs to when an address related event happens, for example when an address is added or deleted. The implementation also needs a way to retrieve all the addresses of the interface when it groups the current address set into a REA parameter sent to the peers. To support these things the implementation has a simple function which maps network interfaces to SAs (and vice versa if needed).

- **1 SA per address family** Include all addresses of a given address family under the same SA. For example, group AF_INET and AF_INET6 type of addresses into separate SAs.

This design adds some complexity to the implementation. The major requirement is of course the one that the implementation has to have support for several address families when sending HIP packets or data over ESP to the peers. The peers need also support for the used address families in order to receive the packets successfully. The current draft specifies only IPv6 addresses or IPv4-in-IPv6 format IPv4 address [11] to be used in a REA parameter.

Additionally, the implementation needs many external support functions. Examples of such functions are the ones that retrieve all addresses, filter certain types of address families from the list of retrieved addresses, and final selection of the addresses to be included into the SA. Also, if a host has several addresses, the REA parameter size may become large due to the high amount of addresses listed in it. On the receiving side long address lists require longer processing time, causing delays during the readdressing.

Hosts moving rapidly from a network to an other encounter frequent address events. Whenever an address event is triggered, the SA grouping process must be restarted. If this address grouping design is used, a host has to perform several heavy operations frequently. Because typical mobile hosts have a limited processing capacity and power supply, this design is probably not the most efficient one.

A good thing in this design is that traffic can be filtered and processed differently depending on the address family. A clever implementation may have separate optimized versions of packet handling functions for each address family, therefore avoiding complex huge functions which try to handle all possible address families within a single packet processing point.

- **1 SA per upper layer protocol** SAs could also be grouped according to the upper layer protocol, the upper layer being here the transport layer. The most common examples of transport layer protocols are the TCP and UDP protocols. In this scheme each transport layer protocol has its own SA. Because HIP does not restrict what transport layers are supported in ESP packets, this effectively causes the creation of as many SAs as there are different transport layers used. Additional note is that all IP addresses of a host appear in every SA simultaneously, simply because these transport layers are sent on top of IP, and we do not want to restrict the selection of available addresses to be used.

The problem in this scheme is that it interferes with several layers. Implementations should by all means avoid modifying the other layers than the internetworking layer unless absolutely necessary.

To implement this scheme, the implementation needs a way to know what kind of data traffic is transferred. To find out the transport layer protocol used, every packet has to be inspected before it is encapsulated in ESP packets. This will most probably result in major performance drawback. Because major part of the applications today use TCP or UDP, it is expected that when using this scheme there would be only two SAs in use in majority of the time.

- **1 SA per address scope** Group SAs based on the address scope. For example, IPv6 addresses have different scopes, such as global, site-local, and link-local (see [11])

Grouping based on scopes may help e.g. designing administrative policies. Because link-local addresses are not routed outside the local network, the properties of the SA can be more relaxed (for example, use shorter keys for faster encryption/decryption of ESP packets) than the one which contains global addresses.

6.4 Selection of Outbound ESP SA

If a host is allowed to setup multiple IPsec ESP SAs which all are related to the same HIP association, then the next question which follows naturally is *“how to select the outbound SA of the peer host which has the best characteristics in the current moment ?”* This question raises also more questions, such as *“how to select the peer address from the ones that belong to the selected SA ?”*.

The selection of the SA has hidden implications. In general, it is extremely hard to know in advance the long-time characteristics of the path that the packets traverse between two hosts, thus rendering a trivial fixed value SA selection impractical.

The two most fundamental things affecting the path are the source and the destination address of the packet. If both hosts are multihomed and have multiple addresses, the number of possible source/destination combinations become large and therefore the number of different paths grows also.

Usually when a host has multiple addresses, it is connected to many different networks. The connection may start flowing through one network and later be switched to a network having completely different characteristics.

The destination address is selected from the address set of the outbound SA. This implies that the destination address selection algorithm has a significant effect on the path. By default, a reasonable assumption is to use the address marked as the preferred address sent in a UPDATE readdressing packet. This

assumption is based on the presumption that to peer host knows better which address will have the most suitable long-time properties (e.g. reachability, good throughput). Nothing prevents an implementation from selecting any other destination address than the preferred one, but then the host takes knowingly a risk of non-working or low-grade connection.

The destination address affects also the source address selection. Typically operating systems try to select the most applicable source address having similar properties, such as using link-local address for link-local destination address even if the host has global addresses. To prevent the possibility of selecting a “bad” address, a HIP implementation should instruct the operating system on what source address to use. A host may decide not to include all of its available addresses into the REA parameter of the UPDATE packet. That way the host can try to limit the possibility of unpredictable path behavior because the peer host will never send data back to any previously unadvertised address. This requires that the sending host remembers what addresses it has advertised previously and select the address among them.

Oscillation and repetitive fast switching between outbound SAs should also be avoided for the same reasons defined above. If some packets arrive faster when going through a different path than the packets sent later going slower path, the latest packets sent by the host may be dropped at the peer due to the ESP replay window.

Several other documents have more thorough discussion on address selection issues ([2], [32]), so deeper analysis is out of scope of this thesis.

6.5 Dynamic Outbound Path Probing

What if the data transfer performance is not adequate when using the current outbound ESP SA, and we assume that there might be a path having better characteristics which is achieved using another available ESP SA ? The problem here is that we do not know the path characteristics until we change the ESP SA. Changing the ESP SA will consume resources uselessly, and changing it back to the previous ESP SA doubles the effort if the new ESP SA was even worse choice.

This implies a need for a mechanism which does not change the active outbound ESP SA when it is not absolutely necessary. What could be better is that the implementation sends periodically probes through all other outbound ESP SAs.

If the current outbound ESP SA goes through a slow WLAN interface, and the probe is sent through a fast Ethernet LAN, the acknowledgement to the

probe will come much faster than just using the WLAN. The implementation can then decide if the results indicate the the switch would be beneficial.

The requirement for the implementation of the probing is that it should match a typical HIP related data transfer. Empty ESP packets having the SPI field belonging to an active HIP established ESP SA are suitable for this. The empty ESP packet contains only the ESP headers and not any user payload. The implementation only needs to handle these kind of special ESP packets and inspect whether the packet is a probe or response to our own probe packet.

However, the idea of probing is not that simple. First, the implementation of the probing mechanism must be reliable. It must handle lost probe packets and not perform changes of the active ESP SA based on the information in the probe response packets which are related to old probes (e.g. due to queuing delays in the path). The second note is that the probe packets might estimate only current round trip time (RTT), but not current throughput or overall long time performance in general.

Chapter 7

Analysis

To complete the theoretical discussion in chapter 2 and 3, and design decisions in chapter 6, we generate concrete results used for the analysis purposes. In this chapter we define how an actual HIP implementation is used to measure the effect of using HIP, and then provide an analysis of the measured results.

The focus of the measurements is on the other main topic of the defined research problems in chapter 3: *Issues when implementing a networking protocol providing identifier/locator split*. The purpose of this chapter is to provide an overall analysis on the focus area. Also, another important aim is to gain insight on what are the effects when adding a new layer, the Host Layer, into an existing networking stack. Both can be achieved by using an actual Host Identity Protocol implementation as the basis for the measurements.

This thesis is based on a publically available HIP implementation, **Host Identity Protocol for Linux, HIPL**, from the HIPL development group [12]. The HIPL distribution consists of the standard Linux operating system kernel with HIP support, user space support libraries and tools, and documentation. The Linux operating system kernel used is from the 2.6 series (the actual version is 2.6.11).

The network stack of the HIPL distribution also follows the implementation design guidelines presented in chapter 6, making HIPL a natural choice to be used in this thesis.

7.1 Measurement Goals

We measure the effects of the HIPL implementation on the networking stack. The focus is on the internetworking and transport layers, because HIP is re-

lated to both of them.

The detail level should not be too high, so we provide a high level measurement in order to cover the overall effect of HIP. That is, the focus is not on the individual operating system functions presented in the chapter 6, but on the issues what an average user or networking software developer will face when using HIP.

The following targets cover most of these cases:

- bulk data transfers
- common network system calls used in networking applications

Bulk data transfer test actually covers many individual areas. A data transfer can be thought of a combined item of connection setup, sending and receiving data, and closing the connection. A typical user is interested only in the time the transfer takes in total, not how much time is spent on some specific issue.

On the other hand (from a software developer point of view), individual programming language functions added with logic create the applications. This is why also the networking related programming language functions should be tested to see the effect on a particular functionality. We have selected a set of functions among the well known BSD based C language networking API for a closer look. The functions are defined below in section 7.3.3.

7.2 Measurement Non-Goals

Not every topic discussed in this thesis can, or is not even feasible to be measured. Therefore some areas have to be moved out of the focus of the measurements.

Because the Host Layer issues are considered to be the most important part of the measurement targets, mobility and multihoming related functionality are not included due to time constraints. This includes also the grouping methods of the IPsec Security Associations. Similarly no other HIP extensions are included in this thesis.

No comparison to other mobility protocols are performed.

As of writing this thesis, the implementation supports only IPv6, therefore no IPv4 will be used in the tests. Only IPv6 is used in the tests.

7.3 Measurement Methods

7.3.1 Hardware

To minimize the effect of other networking components (hardware and software), such as routers and network stack software running on other nodes between the two hosts, we use a direct connection between the two hosts.

The underlying network technology is Ethernet. All network interfaces have 100 Mbps link speed and duplex is set to full.

Two standard PCs are used, both are based on the 32-bit Intel processor architecture family. The first computer has *Pentium M* CPU running at 800 MHz and 1 GB of RAM (later referred to as **Host A**), and the other computer has *Pentium III* CPU running at 700 MHz and 512 MB of RAM (later referred to as **Host B**).

7.3.2 Software

Operating System

The measurements are done using the HIPL provided Linux operating system. The version used in the tests contains all the HIP related packet handling in the operating system kernel, so the user space applications used in the tests should not have a noticeable effect to the results.

The tests are run with HIP enabled and without HIP to see how much HIP adds overhead.

The HIPL distribution supports DSA and RSA as the host key algorithm type. Both key types are used in the tests to see if there are any noticeable differences when using different host key types.

During all the tests, the length of a DSA host key is 1024 bits and the length of a RSA host key is 1024 bits.

Test Applications

The client and server type of applications implemented in the appendix A are used as the base in these tests.

Tests are run on both of the computers while each one acts as a server and a client, then separate results for both situations are given for both of the computers.

The effect of other running processes have been minimized as much as possible, but there might still be some unexpected issues left that can have an effect to the measurements. This has to be taken in consideration when interpreting the results. For example, typically when other than the test processes are scheduled for running by the operating system while the tests are run, there will be sudden stray results in an otherwise constant and somewhat predictable graph.

Most of the data in the Internet is sent over TCP, so it will also be the main transport protocol in the tests.

7.3.3 Definitions of the Measured Targets

After all, we are interested more on the *relative* results than absolute. Better-looking absolute values of the results can always be achieved simply by using a faster processor or network elements, therefore making an evaluation of the implementation hard. By calculating the relative values between the non-HIP and the HIP case, we can spot the areas easier which are affected most by addition of the new networking layer.

In the tests which test common network system calls used in the applications, the application used should contain the parts of a typical networking application. A common typical use case is the client/server model, and it is also used in these tests. Another host acts as the server to which the other host, the client, connects to, exchanges some data, and finally closes the connection. The client and server applications in the appendix A are slightly modified in order to get the test results. The time each system call takes is measured using the standard *gettimeofday* C language function. On the hardware used in the tests, *gettimeofday* claims to have about 1 μ s resolution.

As discussed above, the data transfer can be divided into distinct measurable parts: connection setup, sending and receiving of data, and closing the connection. The C language system call functions which perform these functionalities are defined next.

Client side only:

- *connect*, connect to a remote host
- *send*, send data to the remote host

Server side only:

- *bind*, bind a name to a socket
- *listen*, listen for connections on a socket
- *recv*, receive data from the remote host

Common for both client and server:

- *close*, close a file descriptor (socket in this case)
- *socket*, create an endpoint for communication

7.3.4 Definitions of the Measured Values

The *total transfer time* of one test round of a test application is defined as the sum of all of the individual tests run in the application. For client this means the sum of system calls *socket*, *connect*, *send*, and *close*. For server this means the sum of system calls *socket*, *bind*, *listen*, *recv*, and *close*. *Data transfer rate* of a test round is measured by dividing the number of bytes written or read in total by the transfer time.

An application is not purely a sequence of system calls, it needs also control logic and data structures. These are left out from the calculations of the total transfer times because their use is minimal enough to justify that the total additional effect caused by them would be irrelevant.

In order to get feasible values for the results, the individual tests are repeated 100 times and then the average is calculated from all of those values. Each test round consists of sending 100 MB ($100 \cdot 1024 \cdot 1024$) of data from the client to the server. Server sends only TCP acknowledgement packets to the client, no other high level data is sent back. Between each round there is enough time for test setup and test cleanup tasks to make sure the previous round does not have any effect on the next round.

7.4 Measurement Results

To get the reference points to which the results are compared against, the tests are run first without HIP.

Then HIP is used in a way that there are no Host Associations setup between the two hosts prior to connecting. This tests all the procedures needed to establish the Host Layer and the TCP level connection.

The third test is similar to the second test with HIP, but now there are already setup Host Associations prior to connecting. This way we can roughly see how much the HIP Base Exchange adds overhead.

Both of the HIP tests are run using DSA and RSA keys as the host keys. All results for data transfer times are measured in seconds and data transfer rates are measured in megabytes per second. System call measurements are given in microseconds due to the resolution of the timing method used.

7.4.1 Bulk Data Transfers

The following tables 7.1, 7.2 show the combined results for both non-HIP and HIP tests, and for both hosts when they had both client and server roles.

test item	no HIP	HIP, DSA	HIP, RSA
Host A, server	9.34	20.11	20.47
Host B, client	9.32	20.45	20.81
Host B, server	9.81	17.56	17.55
Host A, client	9.81	17.98	17.91

Table 7.1: Data Transfer Times (in seconds)

test item	HIP, HA, DSA	HIP, HA, RSA
Host A, server	20.38	20.35
Host B, client	20.38	20.35
Host B, server	18.02	17.71
Host A, client	18.02	17.71

Table 7.2: Data Transfer Times (in seconds) (continued)

The following tables 7.3, 7.4 show the corresponding data transfer rates in megabytes (1024*1024 bytes) per second. The value is calculated based on the

actual payload data sent seen by the applications. Actual amount of data sent between the hosts was larger because the calculations can not include IP or TCP headers within the packets. This approach is still usable because it will anyway reveal the relative effect of using HIP compared to the non-HIP case.

test item	no HIP	HIP, DSA	HIP, RSA
Host A, server	10.7	4.9	4.9
Host B, client	10.7	4.9	4.8
Host B, server	10.2	5.7	5.7
Host A, client	10.2	5.6	5.6

Table 7.3: Data Transfer Rates (in MBps)

test item	HIP, HA, DSA	HIP, HA, RSA
Host A, server	4.9	4.9
Host B, client	4.9	4.9
Host B, server	5.5	5.6
Host A, client	5.5	5.6

Table 7.4: Data Transfer Rates (in MBps) (continued)

7.4.2 Common Network System Calls Used in Networking Applications

Next we present the measured results for all the tests that measure the individual system calls. Where applicable, for both of the hosts the results are shown separately.

Without HIP

Here are all the absolute values in tables 7.5 and 7.6 for each system call measured when HIP is not used.

system call	time (in μs)
bind	20
close (client side)	45
close (server side)	17
connect	383
listen	31
recv	9338982
send	9808608
socket (client side)	31
socket (server side)	23

Table 7.5: Host A, without HIP

system call	time (in μs)
bind	19
close (client side)	101
close (server side)	33
connect	670
listen	65
recv	9809659
send	9323927
socket (client side)	112
socket (server side)	46

Table 7.6: Host B, without HIP

With HIP

Again, the conventions for the tables used previously in the non-HIP case are repeated here.

But now, when HIP is used, two separate cases can be measured: there is no existing Host Association (HA) between the two hosts prior to connecting the to the server, or there is an existing Host Association.

Because the HIPL implementation supports both DSA and RSA algorithm as the host keys, both key types are measured separately to see if they differ in practise.

With HIP and no existing Host Association:

system call	DSA host key (in μs)	RSA host key (in μs)
bind	20	20
close (client side)	75	72
close (server side)	16	17
connect	374015	367190
listen	23	23
recv	20108273	20471863
send	17552949	17545922
socket (client side)	26	27
socket (server side)	23	23

Table 7.7: Host A, with HIP, no existing Host Association

system call	DSA host key (in μs)	RSA host key (in μs)
bind	17	17
close (client side)	185	203
close (server side)	26	25
connect	344353	340503
listen	34	35
recv	17561180	17546938
send	20108229	20470993
socket (client side)	65	65
socket (server side)	40	40

Table 7.8: Host B, with HIP, no existing Host Association

With HIP and with existing Host Association:

system call	DSA host key (in μs)	RSA host key (in μs)
bind	20	20
close (client side)	73	83
close (server side)	16	16
connect	861	1041
listen	31	31
recv	20376764	20347450
send	18022274	17710218
socket (client side)	31	30
socket (server side)	23	23

Table 7.9: Host A, with HIP, Host Association exists

system call	DSA host key (in μs)	RSA host key (in μs)
bind	17	26
close (client side)	197	180
close (server side)	27	25
connect	1102	956
listen	55	47
recv	18020254	17708120
send	20375498	20346281
socket (client side)	120	102
socket (server side)	46	56

Table 7.10: Host B, with HIP, Host Association exists

The following tables summarize how HIP affects the system calls used. For each result the absolute (in microseconds) and relative (percentage) change is calculated. If the change value is positive, it means that when HIP was used it took that much longer time compared to the non-HIP case.

system call	no HA (DSA)	no HA (RSA)
bind	0 / 0 %	0 / 0 %
close (client side)	30 / 67 %	27 / 60 %
close (server side)	-1 / -6 %	0 / 0 %
connect	373632/97554 %	366807/95772 %
listen	-8 / -26 %	-8 / -26 %
recv	10769291/115%	11132881/119%
send	10299621/105%	10662385/108%
socket (client side)	-5 / -16 %	-4 / -12 %
socket (server side)	0 / 0 %	0 / 0 %

Table 7.11: Absolute and relative changes on host A, no HIP vs. HIP

system call	with HA (DSA)	with HA (RSA)
bind	0 / 0 %	0 / 0 %
close (client side)	28 / 62 %	38 / 84 %
close (server side)	-1 / -6 %	-1 / -6 %
connect	478 / 124 %	658 / 171%
listen	0 / 0 %	0 / 0 %
recv	11037782/118%	11008468/118%
send	8213666/83%	7901610/80%
socket (client side)	0 / 0%	-1 / -3%
socket (server side)	0 / 0 %	0 / 0 %

Table 7.12: Absolute and relative changes on host A, no HIP vs. HIP (continued)

system call	no HA (DSA)	no HA (RSA)
bind	-2/-12 %	-2/-12 %
close (client side)	84/83 %	102/101 %
close (server side)	-7/-21 %	-8/-24 %
connect	343683/51296%	339833/50721 %
listen	-31/-48 %	-30/-46 %
recv	7751521/79 %	7737279/79 %
send	10784302/116 %	11147066/120 %
socket (client side)	-47/-42 %	-47/-42 %
socket (server side)	-6/-13 %	-6/-13 %

Table 7.13: Absolute and relative changes on host B, no HIP vs. HIP

system call	with HA (DSA)	with HA (RSA)
bind	-2/-12 %	7/37%
close (client side)	96/95 %	79/78%
close (server side)	-6/-18 %	-8/-24%
connect	432/64 %	286/43%
listen	-10/-15 %	-18/-28%
recv	8210595/84 %	7898461/81%
send	11051571/119 %	11022354/118%
socket (client side)	8/7 %	-10/-9%
socket (server side)	0/0 %	10/22%

Table 7.14: Absolute and relative changes on host B, no HIP vs. HIP (continued)

We have also created graphs for some of the most interesting system calls. A single graph shows the measured values for all individual runs made within the total of 100 consecutive test runs. The purpose of the graphs is to visualize what kind of fluctuations there can be when system call times were measured. The graphs are shown only for the host A. For the host B the graphs would have been similar to the corresponding graphs. The results for both of the key types are plotted into the same graph for easier visual comparison.

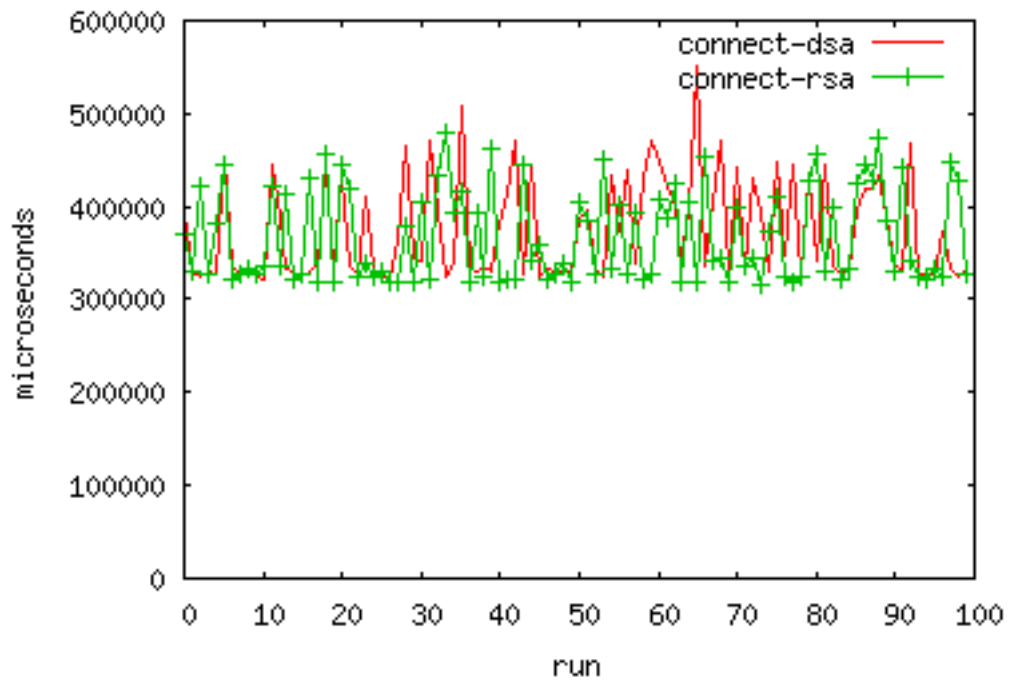


Figure 7.1: Host A client, no Host Association, connect

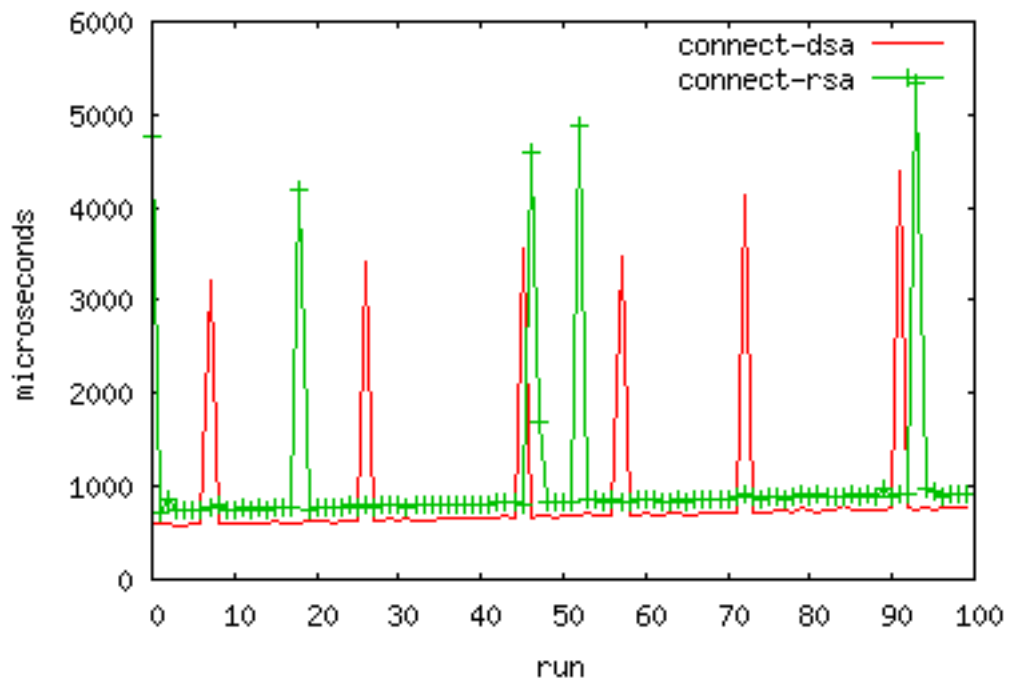


Figure 7.2: Host A client, Host Association exists, connect

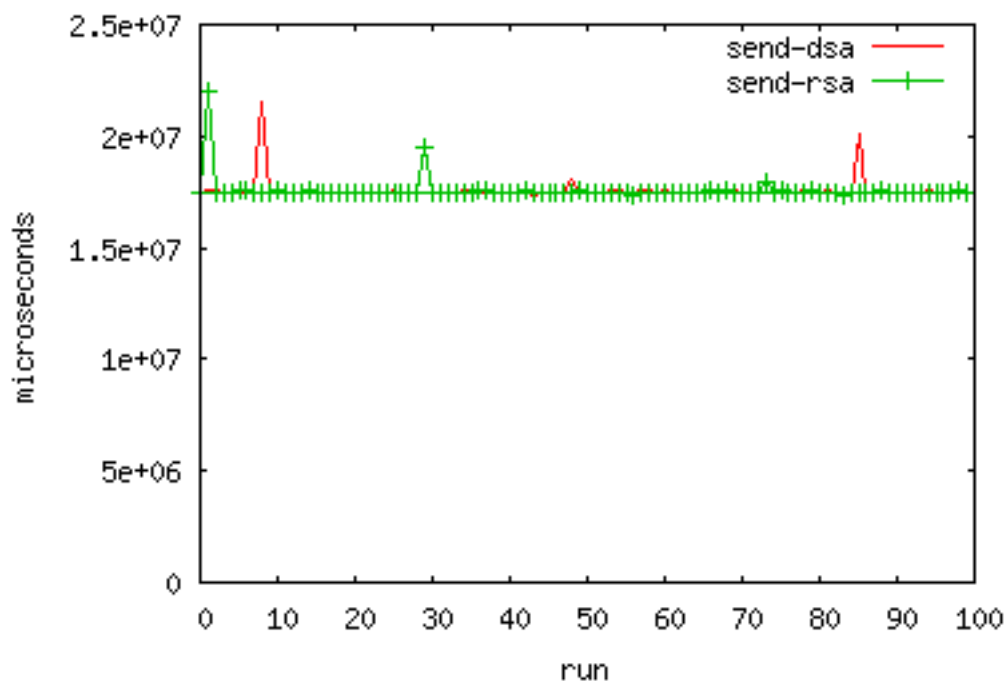


Figure 7.3: Host A client, no Host Association, send

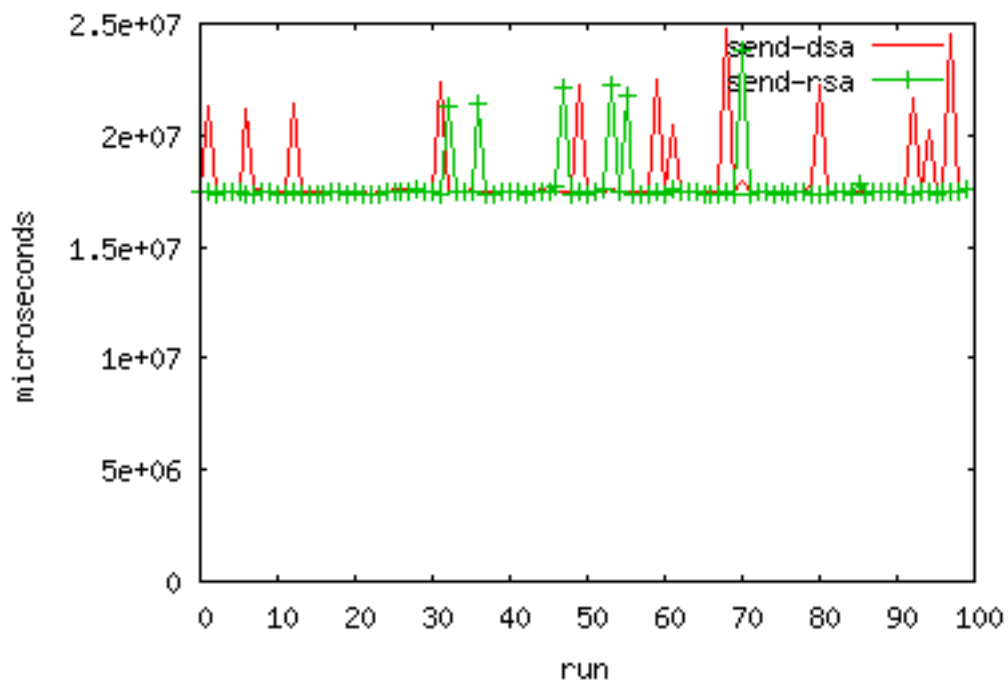


Figure 7.4: Host A client, Host Association exists, send

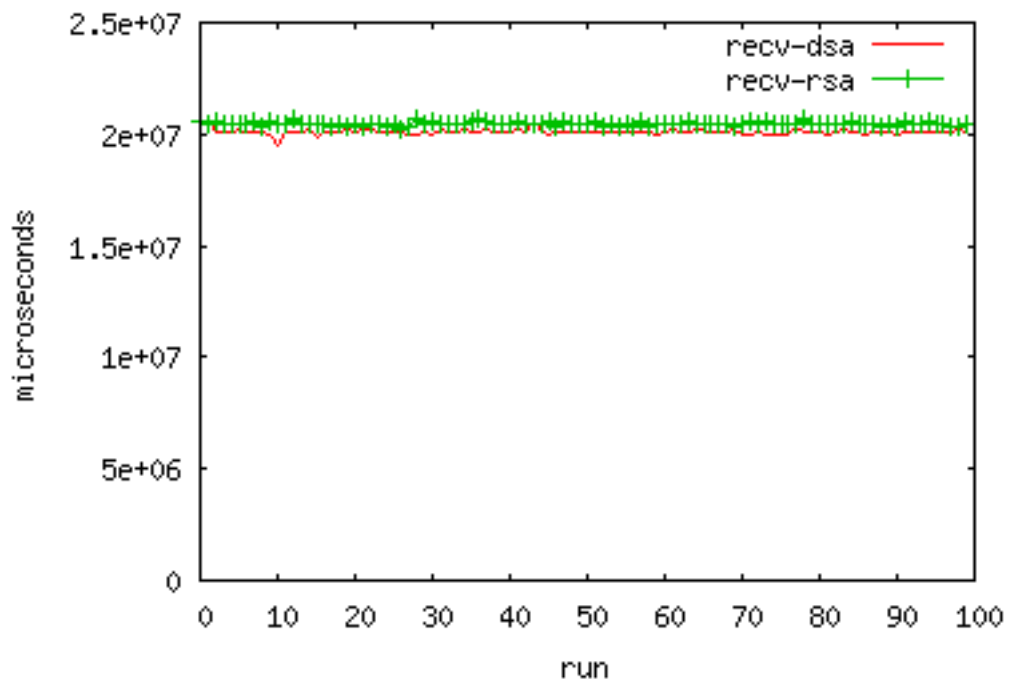


Figure 7.5: Host A server, no Host Association, recv

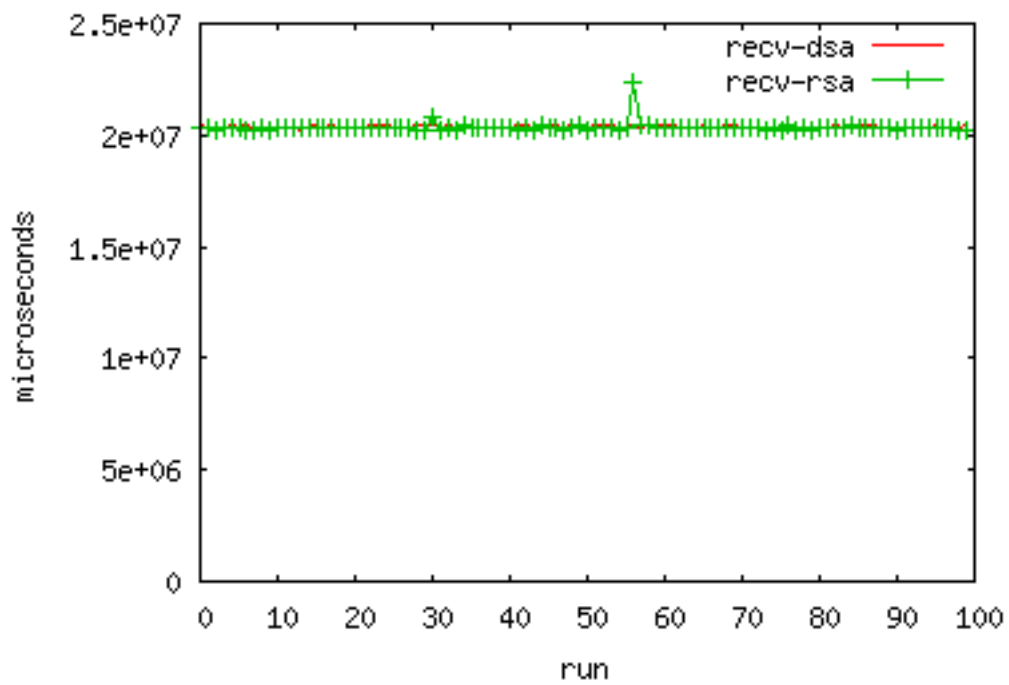


Figure 7.6: Host A server, Host Association exists, recv

7.5 Analysis of the Results

Now is the time to analyze the results. The analysis is based on the measurements presented previously in this chapter.

7.5.1 About the Measurement Timings

Most of the measured times and their differences are presented in microseconds. The function, *gettimeofday* claims to have a microsecond resolution, but in practise it can not be achieved reliably due to the characteristics of the underlying operating system. For example, the process scheduler can interrupt the test process, or there can be access to the disk drives (eg. disk caches, swap) requiring the operating system code to be run. It should also be noted that we measure the actual *wall clock time*, not the CPU time used by the test application.

All these seemingly random events will have an effect to the measurements, even though every possible issue that could cause this to happen have been removed as much as was possible.

Therefore by taking all these into account, we expect that the results within a test run (of 100 individual tests as defined in section 7.3.4) will have some fluctuations. Also, mostly due to the timing resolution issue, if the measured values are low the calculated relative value will seem to be too high.

7.5.2 Bulk Data Transfers

As is seen from the results in tables 7.1, 7.2, 7.3, and 7.4, the results are almost the same even though the other host is slightly faster than the other. When acting as a server, the faster host A is able to process the data sent by the slower client immediately, making the host A to be almost as slow as the host B. Similarly, when host A is acting as a client it will have to wait for host B to first process the packets and then send the TCP acknowledgements back.

When HIP is used but there are no ready Host Associations setup, and host A is the server, the effect of HIP is about **46 %** when compared to the case when plain TCP without HIP is used. For host B the same measurement gives about **51-52 %**. This shows that HIP benefits much from data processing power needed by the continuous data encryption and decryption. HIP will anyway have a considerable effect on the maximum throughput no matter how fast processing power is available.

7.5.3 System Calls

bind

The result tables 7.11, 7.12, 7.13, and 7.14 clearly show that HIP does not have any effect on *bind* in practise. The results are almost identical compared to the non-HIP case no matter what type of host keys were used or what host was used. This is expected because the HIPL implementation does not need to change the system call at all.

close

Even when run on the slower host, the maximum difference for client side *close* was around 100 μ s which can not be considered to be caused by the use of HIP. For server side *close* the differences were even smaller, so the conclusion is that *close* is not affected by HIP.

connect

connect is one of the system call functions which is affected by HIP. Typically, when a host is trying to connect to a peer host using HIP, the *connect* system call will block the running process during the HIP Base Exchange. This is because the first packet, a TCP SYN packet in this case, needs to be sent in order to establish the TCP connection. If a packet is destined to a HIT and there are no Host Associations between the two hosts, the SYN packet will be put into a wait queue in the network stack, the Base Exchange is performed, the SYN packet is taken from the queue, and the SYN packet can now be sent. This will take a little more time than what the actual Base Exchange takes. Optimization of the Base Exchange will therefore improve the time *connect* would take.

When the Host Associations are set up, the difference is very low. The relative change seems to be huge, but human users can really not see any real difference, which was less than a millisecond in this case. This difference is explained by the lookups to the current Host Association tables that the HIPL implementation consults to see if there is a need to perform the Base Exchange or not.

listen

listen is like *bind* or *close*: the absolute differences are too low to make any conclusions that HIP has anything to do with *listen*. In practise, *listen* only sets the socket to prepare for accepting incoming connections and set up a maximum queue limit for the accepted connections. The socket can accept connections from a HIT or just from an IP address, no difference is made at that point.

recv

The decryption of the received IPsec ESP packets sent by HIP is heavy for the processor. The effective processing speed of the received packets drops to a half due to HIP. Of course this test scenario has to be taken into consideration when interpreting the results, because the result depends on how fast the other host can send the data. However, when the results for *recv* for both of the hosts are compared, the result remains the same: HIP cuts the speed in half. Again, the faster the sender is, the less time is needed to wait for the packets to be processed.

send

Also as with *recv*, *send* is affected by the encryption of the outgoing IPsec ESP packets, cutting the effective rate into around half of the maximum value. All the previous packets have to be encrypted before any new packets can be sent out, so only using a faster equipment is an obvious choice if more speed is needed.

socket

For both client and server side *socket*, the results are as expected: HIP does not affect the *socket* system call. This is explained by that when a new socket is created, no HIP related data is calculated or stored into the socket data structures.

7.5.4 Using Different Host Key Types

When the different host key types used in the tests, RSA and DSA, were compared against each other, we observe the same looking pattern as when

comparing the system calls. The absolute differences of *bind*, *close*, *listen*, and *socket* are practically not significant.

With *connect* the change was greater, but it is still below any value which a human user could ever even notice. Even when the communication is happening without human interaction, the maximum measured change of less than 0.01 second can also be considered as insignificant.

recv and *send* had the greatest difference. The maximum differences for both were about 0.3 seconds. DSA was faster than RSA, but the difference between those two is insignificant. If there are more data (at least gigabytes or more) than were used in the tests, the difference can be somewhat noticeable in the long run. However for a normal interactive use, even this value makes no difference.

Based on these observations, we conclude that *for a human user the choice of between RSA or DSA as the host key is insignificant from a performance point of view*. For other use, a few seconds could be gained when using DSA keys but this requires huge amounts of data to be transferred, thus benefiting only a very few applications.

Chapter 8

Future work

Obviously, the research topics of this thesis and the related discussion covered many different areas of interest. But as expected, time has always been a limited resource, as it also have been the case when writing this thesis. If every planned functionality and item would have been implemented, tested and analyzed, the time needed for all that would have been at least double compared to what it have taken for the current thesis. This leaves something for future work.

Most of the things that should be investigated are related to the HIPL implementation. An actual implementation work gives a deeper understanding on the protocol and this way the specifications can be improved later when issues that are hard to understand or implement are encountered.

8.1 Multiple IPsec Security Associations

The largest area to be implemented would be adding the support for multiple IPsec Security Associations and the grouping of IP addresses into an IPsec Security Association as defined in chapter 6.3. More than one grouping method is needed to implement in order to get a real life experience on what kind of a address grouping method would be the best for a typical use and for an other kind of specific use. As the base implementation of the protocol already exists, no major difficulties are assumed to expect during the implementation phase.

Related to the address grouping, the selection algorithm of a IPsec SA to use among multiple Security Associations is needed (as defined in section 6.4). Again, implementing several algorithms benefit at least the HIP mobility support specifications and possibly improve them according to the ideas seen when tests have been run. Also, multiple SA support has performance related issues

which are needed for an usable HIP implementation, such as the caching of SA data structures (section 6.2.3) and dynamic output path probing as discussed in section 6.5.

8.2 Other Interesting Areas to Research

Adding IPv4 support for carrying the HIP and IPsec ESP packets would ease the distribution of the HIPL software distribution into wider audience. Currently, IPv6-only implementation is targeted only for developers and early adopters. Also, comparing the performance between IPv4 and IPv6 would be interesting and it would also show if there are any protocol specific bottlenecks and implementation issues when adding support for a new network protocol to carry HIP and ESP packets.

RSA and DSA are good choices for the HIP host key algorithms because they are well known and widely used protocols. However, the tests were performed using only one key length. Different key sizes should be tested and compared to the measured results. This would also raise kernel side effects, because key length affects the time the kernel needs for encrypting and decrypting the ESP packets. Larger calculations increase the amount of context switches needed, decreasing the application performance.

The actual effect of the HIP implementation could be measured so that all the cryptography for ESP packets would be changed to use the NULL encryption scheme, which does not transform the contents of the packet at all. If then the data transfer rates are measured, the effect of HIP can be calculated by subtracting the time when no HIP was used from the measured result.

Using the NULL encryption would be the easiest approach to improve the data transfer rates. The expected gain is significant, probably even quite close to the non-HIP case. This will however be a bad idea from the security point of view.

Mobility tests were out of the scope of this thesis, but it was noted during the tests that support for link layer events is needed, not only support for IP address events (see section 6.2.5). Addressing events do not receive any link layer events, making the notification of interrupted communication impossible. If network link goes down, an another network interface can be used instead of the one which just went down.

Chapter 9

Conclusions

The focus of this thesis was to research the fundamental issues when a completely new network protocol is added to the network stack of an operating system. The research discussed what is needed as the base for an implementation, what problematic areas are encountered, and what kind of solutions were proposed to the faced problems.

In chapter 3 we defined the specific research problems to be addressed later in the design (chapter 6) and analysis phase (chapter 7). Host Identity Protocol (HIP) was selected as the reference network protocol used in the design of the implementation. HIP was presented in chapter 5. An actual HIP implementation, **HIP for Linux (HIPL)**, was selected as the reference HIP implementation to which all the discussion on design, measurements, and analysis later refers to.

Selecting HIP made the research for the rather new concept of the *identifier/locator split* possible. Identifier/locator split was one of the topics of this thesis. The design clearly showed that it is impossible to achieve the split without touching the protocol stack in the operating system. The positive side is that the required changes are rather small and straightforward.

Most of the items from the design chapter were implemented to the HIPL distribution. The selected design proved to be correct: the Host Associations were established, other related and implemented support functionality was working, and measurements were performed successfully using trivial network applications. Several important requirements were also fulfilled: the HIP implementation did not interfere normal network traffic not using HIP, HIP does not change the normal use of layers above the transport layer, and adding a support for HIP into a typical networking application is easy.

This thesis also provided new and useful information for the future developers

who are going to implement their own HIP implementation or continue developing the current HIPL distribution. Even though this thesis concentrated only on the HIPL, other HIP implementations are also assumed to benefit from the discussion because most of the implementation work is common to all implementations. Therefore we fulfilled the target of gaining insight and experience on the development side. The findings were commented to the authors of the HIP specifications, hopefully for the benefit of the next updated versions of the specifications.

Unfortunately the second research topic, *implications of identifier/locator split on mobility and multihoming*, was not implemented completely. This area would have been too large to complete in a reasonable time, so only the design part for it was proposed. It is assumed that the whole topic could even cover an another thesis.

However, the first research topic was successful. This proposes that the design of the second topic is feasible to implement, too, because both share the same implementation and many of the problems have already been proposed solutions.

The measurements drew guidelines on how much HIP changes the existing ways of networking. The benefit of increased data security when using HIP brings the drawback of slowing down the communications significantly, even up to half of the original value. This was shown to be the major noticeable issue for normal users.

Appendix A

Example applications

A.1 Client

```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3 #include <arpa/inet.h>
4 #include <stdio.h>
5 #include <netinet/in.h>
6 #include <unistd.h>
7
8
9 int main(int argc, char *argv[]) {
10     int sock;
11     struct sockaddr_in6 serveraddr;
12     char *data = "abcdefgh";
13     int datalen, sent;
14
15     if (argc != 2) {
16         printf("Usage: %s hostname\n", argv[0]);
17         return 1;
18     }
19
20     serveraddr.sin6_family = AF_INET6;
21     serveraddr.sin6_port = htons(2000);
22     serveraddr.sin6_flowinfo = 0;
23     if(inet_pton(AF_INET6, argv[1],
24                 (struct in6_addr *) &serveraddr.sin6_addr) < 0)
25         return 1;
```

```

26
27 sock = socket(AF_INET6, SOCK_STREAM, 0);
28 if (sock < 0)
29     return 1;
30
31 if (connect(sock, (struct sockaddr *) &serveraddr,
32     sizeof(struct sockaddr_in6)) < 0) {
33     close(sock);
34     return 1;
35 }
36
37 datalen = strlen(data)+1;
38 sent = send(sock, data, datalen, 0);
39 if (sent > 0) {
40     printf("Data sent\n");
41 }
42
43 close(sock);
44 return 0;
45 }

```

A.2 Server

```

1 #include <netinet/in.h>
2 #include <unistd.h>
3
4 int main(int argc, char *argv[]) {
5     int serversock, clientsock;
6     struct sockaddr_in6 serveraddr, clientaddr;
7     unsigned int clientaddr_len = sizeof(struct sockaddr_in6);
8     char data[100];
9     int received;
10
11     serversock = socket(AF_INET6, SOCK_STREAM, 0);
12     if (serversock < 0)
13         return 1;
14
15     serveraddr.sin6_family = AF_INET6;
16     serveraddr.sin6_port = htons(2000);
17     serveraddr.sin6_addr = in6addr_any;

```

```

18  serveraddr.sin6_flowinfo = 0;
19
20  if (bind(serversock, (struct sockaddr *) &serveraddr,
21      sizeof(struct sockaddr_in6)) < 0) {
22      close(serversock);
23      return 1;
24  }
25
26  if (listen(serversock, 3) < 0) {
27      close(serversock);
28      return 1;
29  }
30
31  clientsock = accept(serversock,
32      (struct sockaddr *) &clientaddr,
33      &clientaddr_len);
34  if (clientsock > 0) {
35      received = recv(clientsock, data, sizeof(data), 0);
36      /* process received data .. */
37  }
38
39  close(clientsock);
40  close(serversock);
41  return 0;
42 }

```

Bibliography

- [1] J. Abley, B. Black, and V. Gill. *Goals for IPv6 Site-Multihoming Architectures*. IETF, April 2003. [Internet Draft] <http://www.ietf.org/internet-drafts/draft-ietf-multi6-multihoming-requirements-05.txt>.
- [2] Jari Arkko. *Failure Detection and Locator Selection in Multi6*. IETF, October 2003. [Internet Draft] <http://www.ietf.org/internet-drafts/draft-arkko-multi6dt-failure-detection-00.txt>.
- [3] History of arpanet. <http://www.dei.isep.ipp.pt/docs/arpa.html>.
- [4] T. Aura. Cryptographically generated addresses (cga), March 2005. <http://www.ietf.org/rfc/rfc3972.txt>.
- [5] M. Bagnulo. Hash based addresses (hba), October 2004. [Internet Draft] <http://www.ietf.org/internet-drafts/draft-bagnulo-multi6dt-hba-00.txt>.
- [6] S. Deering. Overview of ip(v6) multihoming issues. Presented at IPng Working Group Meeting, Tokyo, September 29, 1999 - October 1, 1999.
- [7] R. Draves. Default address selection for internet protocol version 6 (ipv6). Technical report, Internet Engineering Task Force, February 2003.
- [8] R. Droms. *RFC 2131: Dynamic Host Configuration Protocol*. IETF, March 1997. <http://www.ietf.org/rfc/rfc2131.txt>.
- [9] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. *RFC 3315: Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*. IETF, July 2003. <http://www.ietf.org/rfc/rfc3315.txt>.
- [10] Thomas Henderson. *End-Host Mobility and Multi-Homing with Host Identity Protocol*. IETF, February 2006. [Internet Draft] <http://www.ietf.org/internet-drafts/draft-ietf-hip-mm-03.txt>.

- [11] R. Hinden and S. Deering. *RFC 3513: Internet Protocol Version 6 (IPv6) Addressing Architecture*. IETF, April 2003. <http://www.ietf.org/rfc/rfc3513.txt>.
- [12] *Host Identity Protocol for Linux*. <http://infrahip.hiit.fi/hipl>.
- [13] International Standards Organization. *The Open Systems Interconnection Basic Reference Model, ISO/IEC 7498-1*, 1994.
- [14] Sauvola J. and Sun J. Mobility and mobility management: a conceptual framework. In *Proc. 10th IEEE International Conference on Networks*, pages 205–210, 2002.
- [15] D. Johnson, C. Perkins, and J. Arkko. *RFC 3775: Mobility Support in IPv6*. IETF, June 2004. <http://www.ietf.org/rfc/rfc3775.txt>.
- [16] S. Kent and R. Atkinson. *RFC 2402: IP Authentication Header*. IETF, November 1998. <http://www.ietf.org/rfc/rfc2402.txt>.
- [17] Stephen Kent and Randall Atkinson. *RFC 2401: Security Architecture for the Internet Protocol*. IETF, November 1998. <http://www.ietf.org/rfc/rfc2401.txt>.
- [18] Stephen Kent and Randall Atkinson. *RFC 2406: IP Encapsulating Security Payload (ESP)*. IETF, November 1998. <http://www.ietf.org/rfc/rfc2406.txt>.
- [19] Miika Komu. Application programming interfaces for the host identity protocol. Master’s thesis, HUT, TML, 2004.
- [20] Jukka Manner and Markku Kojo. Mobility related terminology, June 2004. <http://www.ietf.org/rfc/rfc3753.txt>.
- [21] Robert Moskowitz and Pekka Nikander. *Host Identity Protocol Architecture*. IETF, August 2005. [Internet Draft] <http://www.ietf.org/internet-drafts/draft-ietf-hip-arch-03.txt>.
- [22] Robert Moskowitz, Pekka Nikander, Petri Jokela, and Tom Henderson. *Host Identity Protocol*. IETF, October 2004. [Internet Draft] <http://www.ietf.org/internet-drafts/draft-ietf-hip-base-01.txt>.
- [23] Robert Moskowitz, Pekka Nikander, Petri Jokela, and Tom Henderson. *Host Identity Protocol*. IETF, March 2006. [Internet Draft] <http://www.ietf.org/internet-drafts/draft-ietf-hip-base-05.txt>.
- [24] *Netfilter project*. <http://www.netfilter.org/>.

- [25] P. Nikander, J. Wall, and J. Ylitalo. Integrating security, mobility, and multi-homing in a hip way,. In *Proceedings of Network and Distributed Systems Security Symposium (NDSS'03)*, pages 87–99. Internet Society, San Diego, CA, February 2003.
- [26] Pekka Nikander, Jari Arkko, and Thomas Henderson. *End-Host Mobility and Multi-Homing with Host Identity Protocol*. IETF, October 2004. [Internet Draft] <http://www.ietf.org/internet-drafts/draft-ietf-hip-mm-00.txt>.
- [27] National Institute of Standards and Technology. *FIPS PUB 180-1: Secure Hash Standard*. National Institute for Standards and Technology, Gaithersburg, MD, USA, April 1995.
- [28] J. Postel. *RFC 791: Internet Protocol*. IETF, September 1981. <http://www.ietf.org/rfc/rfc791.txt>.
- [29] J. Postel. *RFC 793: Transmission Control Protocol*. IETF, September 1981. <http://www.ietf.org/rfc/rfc793.txt>.
- [30] Stevens W. R. *UNIX Network programming*, volume 1. Prentice-Hall, 1997.
- [31] E. Rescorla. *RFC 2631: Diffie-Hellman Key Agreement Method*. IETF, June 1999. <http://www.ietf.org/rfc/rfc2631.txt>.
- [32] M. Riegel and M. Tuexen. *Mobile SCTP*. IETF, October 2004. [Internet Draft] <http://www.ietf.org/internet-drafts/draft-riegel-tuexen-mobile-sctp-04.txt>.
- [33] S. Thomson and T. Narten. *RFC 2462: IPv6 Stateless Address Autoconfiguration*. IETF, December 1998. <http://www.ietf.org/rfc/rfc2462.txt>.