(DRAFT)

A Consolidated Namespace for Network Applications, Developers, Administrators and Users

Miika Komu

August 12, 2012

Abstract

The current Internet is founded on the TCP/IP architecture that was originally designed around machines rather than humans. In the TCP/IP architecture, computers are named and communicate using numerical IP addresses rather than symbolic ones. The numerical addresses were used pervasively at all layers of the networking, until symbolic addresses were added to the architecture. Today, most the users access Internet services directly using symbolic host names using the DNS extensions or indirectly by using key words for search engines. However, IP addresses still remain pervasive as they are employed throughout the networking stack. Even network applications are bound to use them either explicitly or implicitly. This pervasiveness is a source of inflexibility as the TCP/IP architecture is reused in new contexts for which it was not designed for. On the hand, the wild success of the Internet has drawn large financial investments around it. Consequently, even conservative improvements to its ossified design can face a difficult deployment path.

In this dissertation, we examine a number of legacy-compatible solutions to three challenges in the TCP/IP architecture. The first challenge of non-persistent addressing stems from the reuse of IP addresses at network, transport and application layers. While this simplified the naming model of the original TCP/IP architecture, it disrupts TCP streams when the topological location of the mobile device changes. The same phenomena occurs when a user switches between, e.g., WLAN and cellular connectivity in a mobile handset. As other causes for non-persistent addressing, Internet transparency is lost as NAT devices are based on private address realms and site renumbering is difficult as addresses are hard coded into various configurations. As the second challenge, heterogeneous addressing as introduced by IPv6 complicates addressing of hosts and the networking logic of applications. As the third challenge, IP addresses are easy to forge and measures to secure addressing are needed.

A consolidated namespace meets all of the aspects of three high-level challenges. From the surveyed solutions, we have compared five prominent solutions to fulfill the requirements for such a namespace and chosen Host Identity Protocol (HIP) for empirical evaluation. As other work exists in this area, our work focuses on application layer issues as it has remained relatively unexplored. The concrete research problems are three fold. First, we revisit some aspects of the challenges for consolidated naming at the application layer to understand the impact of the problems. Then, we implement improvements to HIP to better meet the goals for consolidated naming for end-users, network application developers and network administrators. Thirdly, we design, develop and analyze technical improvements to HIP in order to facilitate its adoption and deployment.

Tiivistelmä

XX TODO

Preface

XX TODO

Helsinki, August 12, 2012,

Miika Komu

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Miika Komu, Samu Varjonen, Sasu Tarkoma and Andrei Gurtov. Sockets and Beyond: Assessing the Source Code of Network Applications. *Linux Symposium*, Proceedings of Ottawa Linux Symposium, Technical Paper, Ottawa, Canada, July 2012.

II Miika Komu, Sasu Tarkoma, Jaakko Kangasharju and Andrei Gurtov. Applying a Cryptographic Namespace to Applications. In *Dynamic Interconnection of Networks Workshop (DIN'05)*, Proceedings of the 1st ACM Workshop on Dynamic Interconnection of Networks (co-located with Mobicom 2005 Conference), Cologne, Germany, pp. 23-27, ISBN 1-59593-144-9, September 2005.

III Miika Komu, Sasu Tarkoma and Andrey Lukyanenko. Mitigation of Unsolicited Traffic Across Domains with Host Identities and Puzzles. In *15th Nordic Conference on Secure IT Systems (NordSec 2010)*, Springer Lecture Notes in Computer Science, Volume 7127, pp. 33-48, ISBN 978-3-642-27936-2, Espoo, Finland, October 2010.

IV Janne Lindqvist, Essi Vehmersalo, Miika Komu and Jukka Manner. Enterprise Network Packet Filtering for Mobile Cryptographic Identities. *International Journal of Handheld Computing Research (IJHCR)*, Volume 1, Issue 1, pp. 79-94, ISSN 1947-9158, January 2010.

V Miika Komu and Janne Lindqvist. Leap-of-Faith Security is Enough for IP Mobility. In *Consumer Communications and Networking Conference (CCNC'09)*, Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference, Las Vegas, pp. 830-834, ISBN 978-1-4244-2308-8, February 2009.

VI Kristiina Karvonen, Miika Komu and Andrei Gurtov. Usable Security Management with Host Identity Protocol. In *Computer Systems and Applications (AICCSA'09)*, Proceedings of the Seventh ACS/IEEE International Conference on Computer Systems and Applications, Rabat, pp. 279 - 286, ISBN 978-1-4244-3807-5, May 2009.

List of Abbreviations

6RD	IPv6 Rapid Deployment on IPv4 Infrastructures	39
ALG	Application-Level Gateway	34
API	Application Programming Interface	5
APR	Aggregation Point Router	31
ARP	Address Resolution Protocol	26
AS	Autonomous System	28
BGP	Border Gateway Protocol	28
BTNS	Better Than Nothing Security	47
BTMM	Back to My Mac	58
CA	Certificate Authority	44
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart	48
CGA	Cryptographically Generated Address	33
DoA	Delegation-oriented Architecture	36
DHT	Distributed Hash Table	26
DCCP	Datagram Congestion Control Protocol	70
DDoS	Distributed Denial of Service	48
DFZ	Default-Free Zone	22
DH	Diffie-Hellman	46
DHCP	Dynamic Host Configuration Protocol	18
DNS	Domain Name System	10
DNSSEC	Domain Name System Security Extensions	10
DoS	Denial of Service	48
DSMIPv4	Dual Stack Mobile IPv4	24
DSMIPv6	Dual Stack Mobile IPv6	24
dTLS	Datagram Transport Layer Security	45
DTN	Delay Tolerant Networking	7

1. Introduction

The Internet has grown beyond its original expectations but its design architecture has remained relatively static. Particularly, its IP-based addressing model has remained the same even despite of the evolution of network devices. For instance, modern smart phones are equipped with multiple network access technologies and portable devices (e.g. laptops and tablets) traverse between multiple networks. On the other hand, IPv4 address depletion introduced NAT devices that have created problems for end-to-end connectivity. IPv6 was designed to reintroduce end-to-end connectivity but but the protocol has been adopted slowly, possibly due to additional complexity to access control in firewalls, network application developers and even end-users.

Many of the individual challenges in the Internet addressing architecture are solved by a number different, complex and possibly incompatible "band-aid" solutions. Sometimes application developers solve redundantly some of the challenges at the application layer as the network stack or utility libraries are missing the required functionality.

In this dissertation, we propose to extend the addressing architecture of the Internet to consolidate it for application developers, network administrators and end-users. The goal is to explore the limits of the TCP/IP architecture in backward-compatible manner while designing for forward compatibility. To find a balance between a "band-aid" and "clean-slate" solution, we suggest "hip arthroplasty" to the Internet architecture to meet the present challenges in a consolidated way.

By consolidation[1] of the addressing architecture, we refer to tackling of three challenges: non-persistent, heterogeneous and insecure addressing. To mention a few examples, non-persistence means that the addresses of hosts are topologically dependent and the basic TCP/IP architecture

[1]The term was briefly used, e.g., in RFC [49, p. 4] but here we extend the coverage of the term beyond site renumbering

does not provide generic support for topologically-independent addressing for mobile or multihoming capable devices. Non-persistence also includes renumbering of a site when changing Internet Service Provider (ISP) [49] or Internet transparency [48], that is, all devices cannot successfully reach the other devices as Network Address Translation (NAT) and firewall middleboxes block some of the data flows. Heterogeneous addressing is caused by the introduction of IPv6; applications have to be written to support two address families. Insecure addressing refers to the weak security properties of IP addresses that can be forged as they provide no security per se. As a fourth challenge, we also take into count the deployment of protocol architectures from a technical perspective [215, 214].

We argue that the challenges originate from the design of the network-layer addresses. Despite of higher-level naming as supported by DNS, applications have to use addresses and, hence, inherit their limitations. We present a number of alternative solutions to IP addressing but the experimentation is based on one particular architecture. *Host Identity Protocol (HIP)* was chosen as the empirical evaluation tool. In nutshell, the standardized protocol offers a cryptographic identity for end-hosts that isolates the application and transport layers from the fluctuations of the underlying network topology in a secure way [157, 166]. The protocol facilitates IPv6 interoperability at the application and network layers [94, 245, 113], and can restore end-to-end connectivity in the presence of NAT devices [124, 228]. Thus, HIP provides a consolidated namespace to meet the three presented high-level challenges and the approach is reasonably realistic to deploy in practice as it is compatible with legacy applications. The impact of the namespace introduced by HIP is analyzed at the higher levels of the networking stack from the view point of end-users, firewall administrators and application developers.

1.1 Problem and Scope

The challenges studied in this dissertation are related to the lack of a consolidated addressing model for the Internet. In this dissertation, we survey a number of different solutions to these challenges[2]. However, we narrow the focus to a single solution HIP in the end and most of the individual articles are also related this particular architecture.

Compared to some other approaches, a distinct characteristic of HIP is

[2]A number of related surveys exist [54, 173, 97, 129]

that it provides a new namespace that is visible to the applications. For this reason, have chosen to focus on the application-layer aspects of HIP in this particular dissertation. This way, we also supplement a number of other dissertations analyzing HIP from different view points: mobility mechanisms [244, 130], power consumption on hand-held devices [120], HIP-aware middleboxes [91] and HIP applied to cellular networks [93].

It should be explicitly mentioned that Publication I is also part of another dissertation [226]. The other dissertation is based on the HIP namespace as well but contributions lean on connectivity, network hand-off mechanisms from IPv4 to IPv6 and securing name resolution. In contrast, the focus here is to analyze the effects of the new namespace on applications, developers, firewall administrators and users.

While the viability of HIP to the addressing problems will be argued later in this dissertation, the main contribution is related to analyzing of the artifacts of applying and using HIP at the application layer. Based on this, we postulate three high-level research problems for this dissertation:

Problem I. Revisit the challenges for a consolidated namespace at the application layer.

Problem II. Improve and evaluate HIP as a consolidated namespace from the view point of network application developers, network administrators and end-users.

Problem III. Understand the technical deployment issues related to HIP.

Problem I questions the challenges related for consolidated addressing and revisits different the different aspects of it at the application layer. Thus, this problem acts as a “reality check” and is mostly investigated in Publication I.

Problem II takes a step towards more concrete direction and chooses a particular consolidated naming solution from the alternatives presented in chapter 2. In this research problem, we explore and improve HIP architecture empirically for it to better meet the challenges of consolidated naming. To add further practical value, this problem is analyzed from the view point of different interest groups in chapter 3. The contributions to this research problem can be attributed to multiple publications as follows. Publication II introduces a new programmable HIP Application Programming Interface (API) for developers, Publication III presents a use case for HIP to protect end-users from unwanted traffic, Publication IV proposes a firewall to support mobile devices that should ease the burden of network administrators and Publication VI shows a usability evaluation of HIP on end-users.

Problem III acts as another reality check for the proposed solution for a consolidated namespace as offered by HIP. Namely, this problem challenges how feasible it is to deploy HIP from a technical perspective. Again, the contributions to answer this research problem originate from multiple individual publications. Publication I gives practical insight on the API deployment in general, not only HIP. Then, Publication III describes another deployment model where HIP is deployed only at the server side, thus avoiding the hurdles of the client-side deployment. Finally, Publication V proposes a transition path for HIP that reduces the infrastructural dependencies, which are often considered a deployment obstacle.

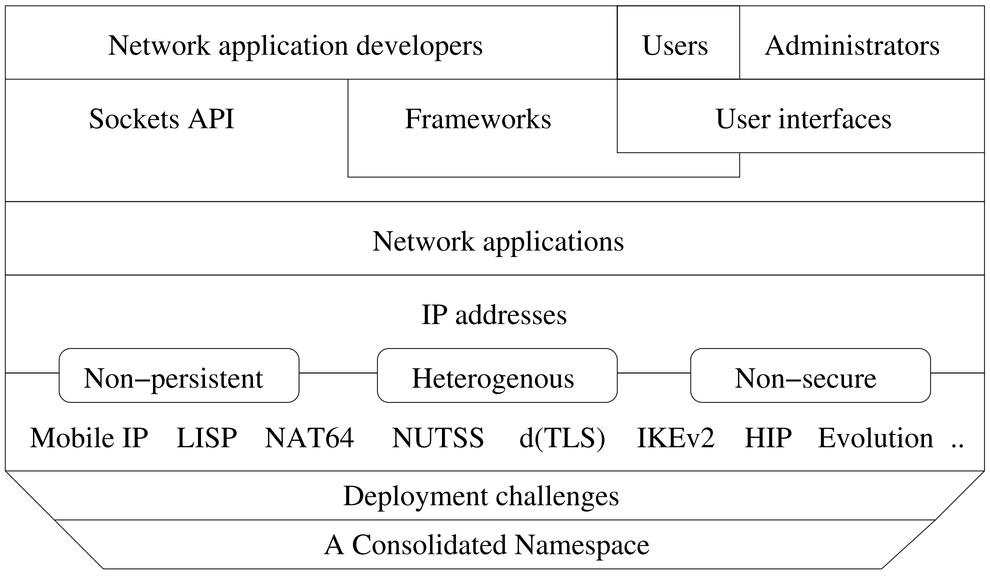


Figure 1.1. A visualization of the challenges in the TCP/IP stack and some solutions

Figure 1.1 visualizes the research challenges of this dissertation using stack diagram. The layer on the top of the figure represents the target groups of this work, that is, network application developers, users and administrators. The developers program network applications either directly using the Sockets API, network application frameworks or libraries, where as the users and administrative personnel typically utilize applications with graphical or command line user interfaces. However, both frameworks and user interfaces use the Sockets API for network communications and it exposes IP addresses directly to networks applications introducing them to non-persistent, heterogeneous, insecure and deployment related challenges. As listed in the bottom in the figure, these individual challenges to achieve consolidated naming can be met using different TCP/IP extensions operating at the various layers of the networks stack. Some of the extensions solve to multiple challenges where as others only to a few. It should be noted that the taxonomy for consolidated naming is described in detail in the next chapter.

The research problems focus on different areas of figure 1.1. In problem I, we analyze consolidated naming in the context of the Sockets API and network application frameworks. In problem II, we investigate and improve HIP to better meet the requirements for consolidated naming from the view point of end-users, network developers and administrators. Research problem III focuses on the deployment aspects of HIP.

For completeness sake, other alternative solutions to HIP will be presented and compared later. However, alternatives will be constrained to backwards compatible or incrementally deployable architectures to make a fair comparison. In other words, we focus on evolutionary architectures that try to minimize economical impact and extend the current life span of the current TCP/IP architecture instead of, e.g., so called clean-slate architectures [188, 172, 174]. Consequently, a number of research problems and related solutions are out of scope [106, 114, 10, 127, 76]. For instance, clean-slate networking based on, e.g., content/data-oriented networking paradigm can require pervasive changes in network applications, stacks and infrastructure. Delay Tolerant Networkings (DTNs) [249, 68, 107] can require a total rewrite of the network logic of the application and wireless sensor networks [28, 242] do not always implement a full TCP/IP stack. Network mobility and mobile ad-hoc networks will not be considered as the thesis makes no contributions on this field of research. Multicast addressing model is not in the scope because it is not globally deployed as a network-layer solution. Our view point is technical; a complete economic analysis of HIP is out of scope.

1.2 Methodology

The methodology is heavily inclined towards empirical experimentation in the collection of publications. With the exception of a purely statistical analysis in Publication I, we implemented and analyzed a proof-of-concept prototype into the remaining publications. The methodology for the quantitative analysis of the prototypes includes usability testing, software performance and complexity measurements, and mathematical modeling.

Qualitative analysis is present in all of the publications. Typically, the design is scrutinized based on the practical insight learned by prototyping and present solutions for the short-comings of the design. As an example of the qualitative aspects, Publication V includes a qualitative analysis related to deployment (backward and forward compatibility of the design). Table 1.1 summarizes the different methodologies used in the publications.

Methodology	PI	PII	PIII	PIV	PV	PVI
Prototyping		✓	✓	✓	✓	✓
Performance		✓	✓	✓	✓	
Statistics	✓					✓
Modeling			✓			
Qualitative	✓	✓	✓	✓	✓	✓
Usability						✓
Complexity	✓	✓			✓	

Table 1.1. The methodology employed to analyze the networking software and concepts in the publications

Simulation was not used as a method in the publications because large-scale scalability was not in the focus of this work. However, all of the performance measurements were conducted on commodity hardware to understand the performance impact for individual hosts.

1.3 Contributions

The contributions of the individual publications can be summarized as follows:

Publication I analyzes statistically open-source software in Ubuntu to explore how applications of today resolve and utilize host names and IP addresses, and how applications employ transport protocols and security. A key finding of this publication is the recurring security problems in the initialization of OpenSSL library. Another finding related to this dissertation is a multihoming issue in all of the four investigated network application frameworks. The contributions of this publication can have real-world impact for improving the open-source software in various Linux-based distributions. The authors have also reported the UDP multihoming problem to the HIP working group at the IETF and it is mentioned in RFC5338 [95, p. 10].

Publication II argues that low-level security transparent to applications, such as HIP and Internet Protocol security (IPsec), can be transformed into something more tangible for application developers. We support our argument by extending the Sockets API to support the cryptographic namespace of HIP by implementing the extensions in the Linux kernel and embedding them successfully into an existing application. We were also among the first ones to empirically experiment and report so called referral problems in HIP and to implement user-specific identifiers for HIP. This publication was further evolved in the Internet Engineering Task Force (IETF) community and eventually was published as an experimental standard [123]. The publication inspired the author to collaborate with SHIM6 working group to design another API that was published also as an RFC [122].

Publication III proposes a cross-layer application of the computational puzzles and the cryptographic namespace in HIP to combat against email spam. In this publication, we integrated HIP puzzle control into a spam filter to introduce a computational cost for senders of spam. As a counter measure, spammers could change their identity to escape identity and puzzle tracking. We addressed this problem as an game-theoretic problem to find an optimal solution for inbound email servers. A shortcoming of the proposed solution is the lack of universal HIP adoption; however, the results are generalizable to new application scenarios without legacy burdens, such as Peer-to-peer Session Initiation Protocol (P2P-SIP) using HIP [118].

Publication IV presents the design, implementation and performance evaluation of a transparent and HIP-aware firewall. The core idea in the design is that the HIP-aware firewall tracks the identities of the client-side devices instead of their IP addresses. This is a relatively simple and secure way to authenticate mobile and multi-access clients because the IP addresses of the devices can change frequently. The proposed design resembles a Virtual Private Network (VPN) solution but is based on end-to-end architecture instead of end-to-middle and supports easy network address renumbering at the service side. The approach eases the life of network administrators because they do not need separate access control lists for IPv4 and IPv6. The publication also analyzes and proposes solutions to some challenges related to deploying the solution, including management of the identities and fine-grained access control to services.

The outcomes of this experiment are referenced by the HIP experiment report [94, pp. 25].

Publication V investigated if the cryptographic namespace of HIP could be managed without deploying the cryptographic keys in the Domain Name System (DNS). The extra records can introduce management complexity and may be subject to forging until Domain Name System Security Extensions (DNSSEC) is adopted globally. The chosen approach was based on leap-of-faith security model that was also a recipe for success for Secure Shell (SSH). We implemented the model for HIP using an interposition library that translated application traffic based IP addresses into Host Identifiers on the fly. The implemented library prototype did not require changes in the applications and could fall back to non-HIP connectivity when a peer did not support HIP. We measured the prototype for software complexity and performance. Finally, we analyzed the design for forward compatibility and for security issues inherited from the chosen model. The results indicate that HIP can be managed without any support from DNS, similarly as other substitute technologies, such as MobileIP, VPNs and Transport Layer Security (TLS). The publication is referenced by RFC5338 [95, p. 9] and the IETF experiment report [94, pp. 11] on HIP.

Publication VI evaluated how end-users perceive the cryptographic namespace of HIP. While HIP can be visible to the applications, this does not necessarily imply that the users actually observe the use of HIP. To raise the awareness of the user of HIP-based security, we implemented a graphical prompt for the user to explicitly approve all HIP-based connections, a HIP plug-in for Firefox web browser and a HIP-aware web site. The entire system was evaluated for usability with two groups of test users. In tests, we applied different combinations of security: no security, leap-of-faith HIP, normal HIP and HIP combined with Secure Sockets Layer (SSL). We employed familiar security indicators in the browser and most users perceived when the connectivity to the web site was secured, despite the prototype was rather unpolished. The findings of the publications were reported to IETF and, consequently, the end-user GUI is briefly mentioned in RFC5338 [95, p. 9] and referenced in HIP experiment report [94, pp. 11].

As the contributions of the publications to the research problems are convoluted as discussed in section 1.1, the order of the publications deviates from the logical order of the problems. Instead, the publications are organized to ease readability. PI is the most generic publication and gives an introduction to the Sockets API. Next, PII extends the Sockets API to provide a generic API for HIP-aware applications. Then, PIII extends the HIP-specific API to support the use case of mitigation of spam. PIV introduces another use case for HIP, that is, to provide infrastructure-based access control for the services of mobile clients. Finally, PV experiments with use of HIP without the dependency to DNS infrastructure which is then tested with end-users in PVI.

Table 1.2 summarizes the contributions of the individual publications from the view point of the challenges and the target user groups. The challenges related to non-persistent, heterogeneous and insecure addressing are covered by the first research challenge. The target user groups are related to the third research problem.

Challenge/target group	PI	PII	PIII	PIV	PV	PVI
Non-persistent addressing	✓	✓		✓		
Heterogeneous addressing		✓		✓		
Insecure addressing	✓	✓	✓	✓	✓	✓
End-users						✓
Network app. developers	✓	✓				
Network administrators	✓		✓	✓		

Table 1.2. Contributions of the publications for the addressing challenges and target user groups

1.4 Author's Contributions

The author of this dissertation was involved in all activities of the publications, starting from conception of the idea, spanning from design, implementation, measurement and analysis of the results, and ending in writing of the publication. Here, some of the author's contributions to the publications are highlighted.

Publication I: The author contributed many of the ideas in this publication and manually inspected half of the frameworks. A co-author handled collection of the statistics.

Publication II: The author did most of the work for this publication, including design, implementation, experimentation and writing.

Publication III: The idea and design for this publication was conceived by the author. The author integrated the puzzle mechanism into the spam filter and measured puzzle performance. The theoretical analysis originated from the co-authors but was coordinated by the author.

Publication IV: The author designed and conducted the measurements, and participated to the writing of the journal article.

Publication V: The initial prototype was designed by the author and implemented by a student under the instruction of the author. However, the author rewrote the prototype several times to optimize it before the actual measurements, which were also conducted by the author. The author also wrote a large part of the text in the publication.

Publication VI: The author chose to be the second author in this publication even though the workload was split evenly between the first and second author. The author's contribution to this publication was mainly technical even though he participated to the design of the implementation and the design of the usability tests. The author participated also to the actual usability test events in the role of a note taker. The graphical user interfaces were implemented by another developer under the instruction of the author.

During his post-graduate studies, the author published also a number of other research papers, participated in the implementation and interoperability testing of the various HIP-related standards [157, 164, 158, 112, 138, 137, 163, 163, 162] and has become a co-author of four IETF standards [95, 123, 122, 124]. The author has also contributed HIP-related kernel code that were adopted to the vanilla Linux kernel[3]. The author has been involved with HIP for Linux implementation[4] activities since 2002 which has been used by many other researchers for their own research[5].

[3]`http://lwn.net/Articles/144899/`
[4]`https://launchpad.net/hipl`
[5]Please refer to hipl-users and hip-dev mailing lists at `http://www.freelists.net/`

1.5 Structure of the Thesis

Chapter 2 describes the background and related work. It describes the challenges related to the TCP/IP architecture and organizes them into a taxonomy for consolidated addressing. Also, different solutions to meet the various challenges are presented and HIP is described in the end separately as it has an important role in the collection of articles. Chapter 3 describes and discusses the findings of articles at a high level, and also points out future directions. It should be noted the results of the performance measurements and other technical details are not repeated in this section; rather, the goal is to position the work from the view point of consolidated naming and the target user groups. Finally, the work is concluded in chapter 4.

2. Challenges and Solutions in the TCP/IP Architecture

The TCP/IP architecture was conceived at an era when end-users consisted of a trusted circle of people and network devices were too large to be portable. The short-term needs prevailed and TCP/IP architecture was designed with these two assumptions in mind. While these simplifying assumptions contributed to the success story of the Internet, they do not hold true anymore as practically every Windows desktop machine is equipped with a virus scanner and hand-held devices are a part of everyday life. Despite of the early pioneering work to improve network architectures [202, 193, 194, 52], the restrictions of the original TCP/IP architecture are still present and the remaining challenges to meet modern requirements are fulfilled by different extensions to the architecture.

This chapter presents a survey of a number of problems originating from restrictions in the TCP/IP architecture and their solutions. We focus on issues related to network addressing from the view point of the network and upper layers. These issues are further organized into taxonomy and divided into challenges related to non-persistent, heterogeneous and insecure addresses. Within the problem scope of this dissertation, we then enumerate through a number of standardized and research solutions that address the challenges within the scope of this dissertation[1]. In addition, technical challenges related to deployment barriers will also be discussed briefly. We also present the architecture of HIP in more detail because it was chosen as the vehicle for experimentation in the collection of articles. Finally, the last section presents a comparison of the different approaches and explains how the collection have improved HIP in order to prepare for next chapter that presents the contributions in detail.

[1]With the exception of Plutarch, all chosen solutions have been also implemented

2.1 Non-Persistent Addressing

The TCP/IP architecture of the Internet was designed around the contemporary restrictions of large computers that were difficult to move around. However, electronics followed Moore's law, resulting in cheaper and smaller electronics for consumers. As the electronics was shrinking in size, portable devices, such as laptops and cellular phones, became pervasive. Consequently, the original restriction for static hosts was no longer true even though is was still present in the design of TCP/IP networking stack.

The TCP/IP stack remains still constrained by its original design that was effectively a design compromise to make the addressing model more simple. Namely, TCP was designed to reuse the same namespace as the IP layer. While an obvious benefit of this short cut was to avoid managing an additional namespace, the main drawback is that this makes TCP connections more static. As TCP connections are created based on the same addresses used by the underlying network layer, the connections are rendered broken when the address changes or is removed. In contrast, UDP is more suitable than TCP for tolerating address changes by its connectionless nature. However, it can be used in a connection-oriented way, exposing it to the same constraints as TCP.

In general, the TCP/IP architecture is challenged in the temporal dimension of addressing as it was designed to assume stable addresses. This not only problematic from the view point of initial connectivity but especially with sustaining of on-going data flows.

In this section, we look at the challenges related to the transient nature of addresses in the TCP/IP architecture from the view point of the application layer. While the lifetime of addresses, and perhaps the quality of service related to the use of the address, is directly visible to the application, we describe a more fine-grained taxonomy of the related challenges. Challenges related to long-term disconnectivity as being tackled by DTN are out of scope. Then, we fit some example solutions to each category. Finally, as some solutions fit multiple categories, such as is the case with HIP, we provide a summary of the problems solved by each solution.

2.1.1 Challenges

Originally, IP address was defined to only be used at the network layer but TCP reused the addresses as its connection identifiers [214, p. 15]. Correspondingly, the Sockets API, the de-facto low-level programming interface for network applications, was designed before DNS and is therefore heavily encumbered with the use of IP addresses [49, p. 15].

While the reuse of IP addresses at multiple layers offers relief from address management issues, it is a layer violation that results in undesired dependencies between the layers. IP addresses are confined to the local network topology and effectively define "where" a host is located, where as transport-layer identifiers define "who" the connection end-point is [153, p. 6][2]. Consequently, transport layer becomes dependent on the location of the end-host and its data flows break when the end-host changes its location. The problem is further aggravated by applications that should be using application-layer identifiers (defining "what"[3].), e.g., FQDNs or its extensions, but instead employ IP addresses directly. Such use can result in connections to incorrect or even malign hosts because IP addresses are typically ephemeral by their nature and, in private address realms, overlapping. To further aggravate the problem, applications have also little means to discover when IP addresses are stale because the Sockets API does not attach any lifetime to the data structures associated with IP addresses [214, p. 11].

For the just described reasons, it could be argued that the basis of the TCP/IP architecture is founded on the assumption of stable, or persistent addresses. Paradoxically, addresses are nowadays *non-persistent* especially due to the advancements in modern, mobile end-user equipment and dynamic network environments. As TCP/IP is universally deployed and adopted, changing its fundamental nature is economically challenging and, thus, various network technologies to reintroduce the persistent addressing model have emerged. However, many of the solutions tackle only single problem emerging from non-persistent addressing. Hence, we roughly categorize the different approaches according to mobility, multihoming, renumbering and Internet transparency challenges.

In the first *mobility* challenge, a single device or an entire network of devices changes its attachment to the network, which typically occurs due to physical movement of the device(s). Network mobility [160] is out of scope of this dissertation and the focus will be instead on host mobility [149, p. 25]. In host mobility, the problem is dual fold: when a host moves to different network, it cannot no longer be reached by other hosts and its existing

[2]IP addresses are also used for routing which defines "how" to get there
[3]Due to the coupled role of addresses, Fully Qualified Domain Names (FQDNs) could be considered as the new "who" and Universal Resource Locators (URLs) as the new "where" [69, p. 21] due to the pervasiveness of the web

data flows are terminated. While both of these mobility-related issues are important for applications based on the Peer-to-peer (P2P)-networking, the initial reachability is more of a concern for servers where as sustaining of existing connections is of importance for the clients in the client-server paradigm. It should be noted that mobile end-hosts are also challenging from the view point of infrastructure as network-level firewalls typically authenticate based on access-control lists based on fixed IP addresses.

The second *multihoming* challenge results in of multiple alternative paths between two end-hosts and can be considered dual fold. Site multihoming occurs transparently within the network, without the end-host applications realizing it besides observations in latency or throughput when the path changes. In contrast, end-host multihoming is more explicit and visible for end-hosts and applications even though many developers are unaware of the end-host multihoming issue as they assume that a single network interface always implies a single address [214, p. 12].

In contrast to mobility[4], the multihoming challenge does not require physical movement of the host but rather stems from multiple available addresses, either on the same or different network interfaces. It has impact not only on the client side but also on the server side. At the client side, many hand-held devices support multiple access technologies such as Wireless LAN (WLAN), 3G and bluetooth. A multihoming problem emerges when inadvertent client chooses initiate communications from a "wrong" address and this may result in a firewall on the path or the server to block the traffic. At the server side, a misconception is that an application binding to a specific IP address to filter incoming data will also send outgoing data from the same address [214, p. 15]. Finally, besides initiation of data flows, the multihoming challenge manifests itself during communications at both client and server side. When the path between an active pair of addresses renders the data flow broken, it would be useful to automatically switch to functional pair of address. Alternatively, two data paths could be simultaneously utilized to maximize throughput. However, such functionality remains unsupported by the TCP/IP stack.

The third challenge is *site renumbering*. As many client-side networks are frequently renumbered with, e.g., Dynamic Host Configuration Protocol (DHCP), renumbering issues surface at services that rely on hard-coded

[4]As a common denominator, Ylitalo [244, p. 22] scopes multihoming as a subset of mobility

IP addresses internally. For instance, corporate mergers or change in the ISP affects the IP address prefix of sites and requires the site to be renumbered. By human error, stale addresses might still be left in various locations after the renumbering. For instance, hard-coded addresses might be discovered in firewall access-control lists of the firewall and configuration files of web servers as described in Request for Comments (RFC) 5887 [49, p. 34]. As human error can result in downtime of services and economic loss, companies tend to avoid site renumbering despite it is adequately documented in the RFC. It also attributes the renumbering issues partially to the fact that the lifetime of IP addresses are dissolved when DNS resolver functions of the Sockets API pass the addresses to the application.

The fourth challenge is *Internet transparency* [48, p. 2] that refers to two aspects of the original Internet design. Hosts were universally addressable and intermediate hosts did not essentially modify packets, which resulted in end-to-end connectivity where all hosts were reachable by others. However, the transparency is broken by the evolution of the Internet as NATs have introduced private address realms that break universal addressability and firewalls drop undesired flows of packets. While NATs slowed down the depletion of the IPv4 address space and firewalls a countermeasure against increasing malign traffic, they did not come without trade offs. As examples of negative impact, NATs complicate communications of P2P-software and firewalls are enforcing Hypertext Transfer Protocol (HTTP) as new the narrow waist for the Internet [184]. Consequently, the Internet is evolving into an end-to-middle architecture, favoring the client-server paradigm, and making the deployment of new transport and network-layer protocols challenging. It is also worth noting that end-hosts need to be able to address all middleboxes for complete Internet transparency.

Related to Internet transparency, RFC 6250 [214, p. 6] describes two misconceptions about applications and reachability. First, reachability with NATs and firewalls is not always asymmetric as clients can reach servers but the reverse may not hold true. The RFC mentions *callbacks* as one source of the problem which are present in, e.g., File Transfer Protocol (FTP). With active data transfer in FTP [185, p. 4], the client passes its IP address as a callback to the server that then creates a new TCP connection with the client. When the client is behind a NAT, the creation of the TCP connection fails, resulting in a failure of the file transfer.

A second issue described by the RFC is that reachability is not always transitive. As an example scenario, host A can reach B that can reach C but this does not guarantee that A can reach C as routes may be different between each node, with different firewalls or NATs between. This problem is also referred as a *referral* problem. For instance, HTTP [72, p. 62] avoids the referral trap with its clever design of redirection. When a web server receives a request from a client that needs to be redirected to another server, the server informs the client to connect directly to other server instead of bluntly passing the client's address as a referral to the other server for connecting back, which would be problematic with in the presence of NATs.

2.1.2 Solutions

As discussed in the previous section, the root cause for the inflexibility of the TCP/IP architecture is that the transport and network layers share the same namespace, convoluting the semantics of the layers. Thus, it is only natural to decouple the namespaces, either in a strict way or loosely to facilitate host mobility and multihoming. In general, this architectural approach is sometimes referred as the identifier-locator split [214, p. 14] and the Internet Architecture Board (IAB) has acknowledged it is a viable way to modularize the TCP/IP architecture [153, p. 7], [145, p. 4],[146, p. 7].

In practice, the identity-locator split approaches introduce one level of indirection in naming and many backwards-compatible approaches try to restore persistent addressing with "surrogate" addresses [146, p. 58] to be used in place of routable addresses. Despite the syntax of the two addresses can be convoluted, the semantic difference is nevertheless usually emphasized by calling the surrogate addresses as identifiers. In some literature [157, p. 3] identity refers to an abstract entity and identifier is a concrete realization of the identity with certain predefined presentation format. In the context, we also use the term *persistent identifier* [103, p. 20] to emphasize the invariant nature of stable surrogate addresses.

The identifier-locator split is just an architectural paradigm that has to be concretely realized with a concrete protocol and, thus, different designs alternatives have been proposed. According to RFC 4177 [103, pp. 8, 15], the split can be implemented by modifying existing protocol elements, or by adding new "shim" protocol elements between application and transport, or between transport and network elements. The RFC further describes that the split can be realized at end-hosts or middleboxes (site-exit or edge routers). According to some terminology [146, pp. 11,21], the former is called *core-edge elimination* and the latter *core-edge separation*.

Regarding to the syntax and semantics with the identifiers, several alternatives exist according to RFC 4177 [103, pp. 18-20]. For instance, the identifier and locator namespaces can be *overlapping* or *disjoint*. The trade off with completely disjoint namespaces is the extra complexity of additional mappings between identifiers and locators – the context [193, p. 2] of use may be needed in order to disambiguate between the overlapping namespaces. The identifiers can be *structured* (typically hierarchical) or *unstructured* (commonly referred as "flat").

Identifiers can further be classified into three levels according to their uniqueness according to RFC 4177 [103, pp. 20-23]. The least unique type of identifiers are *ephemeral identifiers* that are created during a communication session to merely distinguish it from others. As an example of this, two hosts using IPsec-based protection use an Security Parameter Index (SPI) [117, p.] number to indicate the symmetric key used to protect the packet. *Opportunistic identifiers* are bound either temporally or topologically, and do not always guarantee that sequential use of an identifier result in a connection to the same host. For instance, the present Internet with its private address realms can be categorized into this group as the same, possibly private, IP address can result in a communication with a different host depending on which network the connecting host is located. These two classes of identifiers are jointly labeled as *non-persistent* in the context of this dissertation.

Persistent identifiers, are global in their scope and unique across concurrent and parallel sessions according to RFC 4177 [103]. They can be used for initiating communications at any time and anywhere, and are guaranteed to always result in communications with the same host or no communications at all. For instance, MAC addresses are an example of unique identifiers at the data-link layer that are centrally assigned. In contrast, public keys generated to identify SSH hosts can be considered statistically unique identifiers at the application layer.

In the context of this work, we clarify two properties for persistent identifiers. Firstly, we specify that such identifiers must facilitate end-host mobility, end-host multihoming, site renumbering and Internet transparency. Secondly, we note that persistent identifiers are impacted by the referral issues when the identifiers are unstructured and used in the context of name directories supporting only structured name look ups. In such a scenario, the identifiers may need some additional information in order to be successfully searched through the structured directory. Here we, define that the referral issues will not render a persistent identifier into non-persistent and rather describe the referral issues separately.

Besides address agility, the identity-locator split holds also the promise to improve the *routing scalability* of Internet "core", or Default-Free Zone (DFZ) to be more exact, which is facing some addressing-related challenges. For example, many companies prefer Provider-independent (PI) addresses over Provider-allocated (PA) addresses to facilitate easier migration from an ISP to another [146, p. 5],[153, p. 5]. The trade off here is that PI addresses do not aggregate as well as PA addresses, thus PI addresses create larger routing tables and challenge routing scalability [29, p. 8]. Essentially, the identity-locator split can be used to reap the benefits of PI and PA addresses. The identity namespace offers the same topology-independent functionality as PI addresses and the locator namespace supports aggregation similarly as PA addresses. For applications, the unresolved scalability problems may cause degradation of Quality of Service (QoS) in the future. The solutions (based on identity-locator split) can limit the degradation and affect the way how applications identify other hosts.

The benefits for the routing scalability in the elimination approach are only fully realized when most of the hosts of a site are upgraded to support the elimination. In contrast, only the edge routers have to be upgraded in the separation approach to enable it for the whole site[5]. The elimination approach improves routing scalability, as routing tables grow with the number of ISPs rather than edge networks [108, p. 2]. However, the separation does not support Internet transparency because it creates a separate locator namespace to solely for the routers. Consequently, end-hosts cannot address routers anymore because they are confined to the identity namespace [108, p. 4] in the separation approach.

Protocols implementing the identity-locator split are typically based either on *tunneling* or *address rewriting* [103, pp. 15-16]. The tunneling approach is also referred as *map-and-encap*. With tunneling, the packets are encapsulated with an extra header: the inner header contains the identifiers and the outer header the locators. The header is added when the host responsible of the identity-locator split sends the packet, and

[5]Communications with legacy sites is another issue in core-edge separation

correspondingly removed at the destination by the responsible host. In address rewriting schemes, the responsible hosts translate identifiers to locators when sending and translate locators back to identifiers at the destination. The translation can involve the whole address or just portions of it, such as the prefix.

The tunneling approach requires a mapping infrastructure from where to look up the locators corresponding to the identifiers. This approach can result in lost packets especially when combined with the core-edge separation. When an edge router receives an outgoing packet with the identifiers, it has to look up the corresponding destination locator and thus may have to drop the packet until the look up is completed. As the tunneling scheme adds an extra header, it changes Maximum Transfer Unit (MTU) and fragmentation processing [153, p. 18]. Tunneling breaks also geolocation based on IP addresses [214, p. 14]. However, a benefit of the tunneling is that it is stateless as each packet contains all the necessary information to process it.

As an alternative to tunneling, address rewriting typically requires some extra state at the middleboxes. Typically, packets do not have be dropped as the translation is known beforehand. Translation does not affect MTU as no extra header is added. It should be noted that some translation schemes alter the semantics of IP addresses, e.g., by splitting a single address into identifier and locator portions. Such schemes require changes to application logic as most of them generally treat addresses as opaque tokens without additional semantics [153, p. 18].

Next, we describe some protocols that try to introduce persistent identifiers to facilitate mobility, multihoming, site renumbering or internet transparency. We survey different approaches based on the identity-locator split in addition to other application, transport and network-level solutions.

Mobility Solutions

The mobility-related terminology in this section refers to RFC 3753 [149] unless separately mentioned. Network mobility refers to relocating to an entire network without interrupting packet delivery. Network mobility [65])[6] will not be further examined here and the interested reader may refer to other sources [64, 60, 178, 62]. In *host mobility*, or *terminal mo-*

[6]In contrast, site renumbering will be discussed later. While renumbering and network mobility could be solved with same technologies, the solutions differ in practice

bility, a host retains its connectivity with other hosts despite of changing its network attachment point. For example, this can occur when a laptop moves from the range of one WLAN network to another. Session mobility refers to migration of an application-layer session between two different devices.

In the temporal dimension, the connectivity can divided into sustaining initial connectivity versus sustaining of on-going communications. Typically, the former requires the help of immovable infrastructure to relocate the moving host. As an example solution to this problem, DNS includes extensions [230] that allow hosts to update their new location in the DNS. In order to provide a stable "anchor" point, the infrastructure is also needed to sustain on-going communications of two hosts that move at the same time. When only one end-point of the communications moves at a time, support from the infrastructure is optional because the moving host can inform directly its other communications partners about its new whereabouts. This process, independently of whether it includes network intermediaries or not, is usually referred as *handoff* or *handover*.

A *horizontal handoff* occurs when a device moves between homogeneous access technologies. As few examples, this can occur when in bridged ethernet networks [218] or when a laptop moves between two WLAN access points [155, 186] or when when a cellular phone transitions between two base stations In contrast, *vertical handoff* occurs when a mobile devices switches between different network types, such as WLAN and 3G. Some mobility mechanisms function only within a single address domain and this is called *local mobility* or *micro mobility*. In contrast, *global mobility*, or *macro mobility*, works across different domains.

Mobile IP is classic example of a protocol supporting global mobility, albeit it has both a standardized [176, 175] and a research variant [41] more optimized for local mobility. Originally, Mobile IP was not *address-family agnostic* [244, p. 22] at the network layer because it had one protocol supporting IPv4 networking [176] and another IPv6 [177]. However, Dual Stack Mobile IPv4 (DSMIPv4) [221] or Dual Stack Mobile IPv6 (DSMIPv6) extensions [205] facilitates both IP versions within a single protocol[7].

In Mobile IP terminology, a moving host is denoted as *mobile node* and its communication partners are referred as *correspondent nodes*. In gen-

[7]Another issue is IPv6 interoperability at the application layer which is the topic of section 2.2

eral, Mobile IP supports host mobility by introducing a persistent, surrogate address for mobile nodes. The address identifies the node, and it is referred as *home address*. Correspondingly, the mobile node is located using its *care-of-address* that refers to its current IP address in the topological network topology. The home address and care-of-address are bound together by a network intermediary, called *home agent* that relays traffic from the correspondent to the mobile node, thus maintaining an illusion for the correspondent node that the mobile node has a persistent address [144, p. 6]. The home agent [156] relays also the reverse direction[8]. When a home agent becomes unreachable, the mobile node can switch to an alternative home agent [134] albeit with the cost of disrupting existing transport-layer sessions.

In contrast, MobileIPv6 supports direct communications from the between the mobile and correspondent node (route optimization) when the correspondent supports MobileIPv6 protocol. Another difference in MobileIPv6 is that IPsec-based security is mandatory which is typically managed with the help of Internet Key Exchange (IKE)v2 [116].

While Mobile IP could be described as standardized end-to-middle tunneling solution, a number of end-to-end solutions have been proposed by the academia. For example, Location Independent Networking for IPv6 (LIN6) [135] splits an IPv6 address into two: the first 64 bits identifies the network attachment point (locator) and the last 64 bits identify the end-host. In essence, LIN6 introduces a new logical shim layer between transport and network layers that serves two purposes. Firstly, the upper layers are isolated from network layer changes as the shim layer translates the locator portion to a LIN6-specific prefix before delivering an incoming packet from the network to the transport layer. This identifier is referred as the "generalized identifier" and the application can utilize it, e.g., for access control as it is immutable. Secondly, the shim layer can manage end-host mobility because it can translate the LIN6-specific prefix to a suitable locator before sending a packet to the network. LIN6 is dependent on extra infrastructure, called *mapping agents*, that the LIN6-capable end-hosts update regarding to their current identity-locator bindings.

As another core-edge elimination approach, a later proposal [9] called

[8]Early versions of Mobile IPv4 employed so called triangular (asymmetric) routing where the reverse direction did not involve the home agent. Also, another optimization called *foreign agents* is not usually deployed at all in practice

[9]ILNP is founded on an even earlier approach called GSE which will be described in section 2.1.2

Identifier-Locator Network Protocol (ILNP) [16] splits an IPv6 address into identifier and locator portions similarly as LIN6. In contrast to LIN6, ILNP does not provide a generalized identifier to upper layers but rather exposes the possibly outdated locator portion. To avoid changing application semantics to interpret only the identifier part, ILNP proposes instead extensions to the Sockets API that use FQDN names to hide the details of addresses entirely from the applications. As another difference, ILNP reuses DNS rather than requiring new mapping infrastructure and stores the mapping information in new DNS resource records. To secure the publishing of identity-locator bindings, the approach assumes secure dynamic DNS updates [234] for site adopting the approach. End-hosts can also inform each other directly about their changed network locations using ICMP(v6) messages but DNS is utilized as a fixed re-contact point when two end-hosts relocate simultaneously. While the approach targets primarily IPv6 with its concrete ILNPv6 proposal, a separate ILNPv4 protocol is sketched that uses IPv4 options to carry the identifiers and requires modifications to Address Resolution Protocol (ARP). ILNP can reuse IPsec to secure its data plane but requires changes to IPsec processing to ignore the locator portion of IPv6 addresses.

Internet Indirection Infrastructure (i3) [211] is another research-oriented architecture facilitating end-host mobility based on core-edge separation and identity-locator split. As the name suggests, i3 achieves mobility by introducing indirection infrastructure that hides the location of the end-hosts. The infrastructure consists of an application-based overlay based on Distributed Hash Table (DHT). For two applications to communicate over the overlay, the server-side chooses an arbitrary identifier for itself and publishes a hash of its identifier along with its IP address in the DHT. Then the client-side application can deliver data to the server through the application-layer overlay using the published identifier. This way, the infrastructure decouples senders from receivers and allows applications to update their current location to the overlay. Based on the decoupling, i3 can further support anycast and multicast.

Forwarding directive, Association and Rendezvous Architecture (FARA) [57] decouples also identity from its location by abstracting the functionality of the different layers of the networking stack. The architecture does not require any new namespace to be introduced but it relies on overlay-based infrastructure to assist in mobility. As a concrete realization of the architecture, M-FARA [181] is based on IP namespace but implements its own transport protocols and network intermediaries to support mobility.

It should be noted that many of the network-layer solutions for mobility remain marginally adopted and deployed. Thus, many application layer solutions have emerged to tolerate mobility at some level. For instance, many web browsers, including Mozilla Firefox, support pausing and resuming of downloads. Email client software, such as Mozilla Thunderbird, can tolerate disconnectivity and automatically reconnect to the email server. Internet telephony as supported by Session Initiation Protocol (SIP) includes session mobility [199]. Most web services identify HTTP sessions[10] with browser cookies and, thus, can tolerate IP address changes for non-streaming applications. In web browsers, the use of persistent HTTP, i.e., the reuse of the same TCP connection has been reducing [44, p. 3], perhaps to tolerate mobility better. Finally, more generic application layer solutions based on, e.g., libraries [246], by introducing a new session layer [203] and overlays [5] have also emerged from the academia but have not been yet embraced by application developers.

End-host Multihoming Solutions

In mobility scenarios, the host may be disconnected for a short period of time (*break-before-make*) during a handoff. However, multihoming scenarios can involve simultaneous availability of multiple network paths that may result in a smoother handoff because the host may be able to prepare an alternative path while sustaining communications using the active path (*make-before-break*). Similarly as mobility, multihoming can be realized at multiple layers of the networking stack. For instance, the multihoming is present with multiple email accounts; sending an email using an unsubscribed email address will bounce off an mailing list. While the challenge persists at different levels, we focus on transport and network level solutions in this section.

As with mobility, multihoming solutions can be realized at end-host or intermediate hosts. When multihoming is implemented at network intermediaries such as routers, it is usually referred as *site multihoming*. The purpose of such multihoming is to make use of network redundancy in order to dynamically switch from one network provider to another when a network fault occurs, but it can also be utilized for load sharing and other traffic engineering purposes [1, pp. 2-3]. For IPv4, RFC 4116 [2, pp. 4-7] documents a number of methods to facilitate site multihoming.

[10]According to some measurements [148, p. 1], HTTP traffic can amount for nearly 60% of Internet traffic

The use multiple Autonomous System (AS) numbers is another option but this approach is problematic as the numbers are a finite resource. A more scalable way is to make use PA addresses and advertise new routes in the Internet with Border Gateway Protocol (BGP) when switching between ISPs. However, the drawback of this is that the site needs to renumber its network when changing its primary service provider. Site renumbering is the topic of the next section and we focus on approaches dedicated solely on end-host multihoming in the remainder of this section.

Scalable multihoming is an important goal for the Internet architecture [146, p. 7]. When the goal is merely to sustain on-going communications for the sake of multihoming, non-persistent identifiers are sufficient enough [103, pp. 21-22]. At best, Stream Control Transmission Protocol (SCTP) [170] is an example of the use opportunistic identifiers[11]. SCTP makes no attempt to uniquely identify hosts but merely facilitates end-host multihoming, and also end-host mobility with later extensions [210], with routable IP addresses. SCTP introduces a new transport protocol [208] that applications can utilize with its separate Sockets API extensions [209]. In contrast to TCP, SCTP can also support multiple streams within a single session and offers a messaging oriented API that avoids application-level framing of messages.

Multipath TCP (MPTCP) [74] extends TCP to support end-host multihoming for both failure tolerance and load balancing purposes. Similarly as SCTP, it makes no attempt to introduce a new namespace. As it is based on IP addresses, it can be characterized to be based on opportunistic identifiers. However, a crucial difference to SCTP is that can work with unmodified legacy applications, despite its optional extensions for the Sockets API [195].

As another approach, identity-locator split has been proposed to decouple the existing transport and network layers from each other to facilitate more generic form of multihoming [103, p. 13-14]. As an example, Site Multihoming for IPv6 (SHIM6) [168] is an IPv6-specific end-to-end solution that introduces a shim layer to the end-host stack. The shim layer does not introduce a new namespace but rather reuses routable IPv6 addresses as both identifiers and locators. This design choice allows a SHIM6-capable host to communicate with a SHIM6-incapable host. Consequently, SHIM6 does not require any changes to IPv6 applications even

[11]Individual streams inside SCTP sessions have stream IDs that are effectively ephemeral identifiers

though it has an optional API for SHIM6-aware applications [122].

Site Renumbering Solutions

Site renumbering occurs typically when site changes its network provider, e.g, to obtain more competitive prices[12]. The new provider will offer a different IP address range and the site needs to be renumbered to correspond the new prefix. Ideally, only DNS-based names would have been used internally by site and by external sites to reference the various services of the local site. Then, the transition could be accomplished by merely updating DNS records and waiting for the cached entries to expire. However, this is not often the case and IP addresses are embedded in various application, service and infrastructure configurations [49, p. 34], and downtime is avoided at any cost.

In a nutshell, site renumbering can interrupt existing transport-layer connections and it can cause certain hosts to be completely unreachable due to address misconfigurations. The former problem can be solved using some of the end-host mobility or multihoming protocols described in the previous two sections and will not further discussed in this section. The latter problem is more severe and this section focuses on solutions adhering to it. Instead of ephemeral or opportunistic identifiers, many of these solutions require more heavy-weight means in the form of persistent identifiers to ensure that services remain reachable.

Another characteristic of the protocols is that many of them support at least site multihoming in addition to the site renumbering. Many of the protocols also advertise themselves as a solutions routing scalability. In this work, we treat problems with routing scalability as a symptom or side effect of missing renumbering support rather than the root of the problem; the actual source of the problem is either a non-scalable choice to support site multihoming or site renumbering, such as is the case with PI addresses. Thus, "routing scalability" will not be discussed in its own section but rather as a benign property of individual solutions to site renumbering.

The scalability of the Internet is challenged by increasing amount of prefixes that aggregate poorly. According to RFC 4984 [153], there are multiple sources that attribute to the problem. For example, some companies avoid renumbering costs by using PI addresses instead of PA addresses. Historical, non-aggregatable address allocations, well as mul-

[12]At least in Finland, it is possible to change your cellular operator without changing your phone number. Unfortunately, this does work at all with ISPs

tihoming and traffic-engineering tricks further aggravate the problem. Moore's law do not help to curb the costs for high-end routers because they employ different type of memory than what is used in commodity hardware such as cellular phones. To recap, the RFC states that routers are becoming more expensive, routes are becoming more "flat" despite routing is still based aggregatable algorithms[13] and the pursuit for local benefits has resulted in global scalability related costs.

RFC 4192 [25] describes procedures on how IPv6 renumbering can be realized manually without introducing service breaks. The renumbering includes prefix-related modifications to switches, routers, firewalls, DNS, DHCP, end-hosts and application configurations. However, RFC 5887 [49] states that renumbering is still hard in practice due to very nature of IP addresses. While dynamic service discovery, IPv6 support for multiple addresses during transition period and unique local IPv6 addresses [100] for intranet communications relieve some of the problems, manual renumbering procedures are still far from a seamless. For example, Time To Live (TTL) values in DNS should be manipulated smaller in the beginning of the transition but this causes caching-related tensions. The TTL values are not percolated to the applications because Sockets API does not support such concept as it was designed before DNS. Until browsers are restarted, many of them employ so called "DNS pinning" that caches the addresses of the server to avoid security issues. In addition, routers may have to be restarted as they cache addresses and server-side applications have to bind to multiple addresses for the duration of the transition period. Finally, the RFC also mentions that protocol level dependencies as 34 out of 257 RFCs had explicit address dependencies.

Independently of whether PI or PA addresses have been used for multihoming or site renumbering purposes in an edge network, injecting non-aggregatable addresses has introduced scalability challenges for the global routing in DFZ [119, p. 1]. As a compromise, readdressing can be avoided in IPv4 by employing private addresses within the site while sustaining scalability with PA addresses [2, p. 7]. In IPv6, the same can be accomplished with Unique Local Addresss (ULAs) [102]. While these approaches support renumbering within an intranet, they need additional support to facilitate external communications. Without additional support (e.g. VPN tunneling), ULAs cannot be used for external communi-

[13]RFC 4984 points out that only few feasible approaches to non-topological routing have been proposed [42, 4, 132]

cations at all. By default in IPv4, NAT devices drop incoming data flows unless some special arrangements are in place. For both incoming and outgoing data flows, NAT devices typically do not support support survivability of transport layer connections [49, p. 17][14].

Evolution [146, p. 52-56] is an incentive-based strategy that combines a number of short-term traffic engineering schemes requiring no coordination between sites into a long-term plan that requires coordination. Instead of promoting a sudden “revolution” in routing architecture, the approach is phased so that each step gradually improves routing scalability and brings immediate benefits for its early adopters [119]. The approach starts from the intra-AS changes and ends with inter-AS changes to ultimately accomplish core-edge separation. The intra-AS changes involve algorithmic improvements to the software in local routers to achieve better Forwarding Information Base (FIB) aggregation locally.

In the second iterative step of the evolution approach, new infrastructural indirection elements are introduced. The local AS deploys new routers, called Aggregation Point Routers (APRs), to facilitate *virtual aggregation* [27, p. 1] within its own AS. The idea is that the APRs split the entire IPv4 namespace artificially into large, virtual fragments and each APR advertises its responsible block to legacy routers inside the AS. This way, legacy routers of the AS survive with fewer route prefixes and the complexity of finding more accurate routes is shifted to new APRs.

As a trade off, the APRs naturally introduce path stretch in the sense of an extra hop. Another drawback is that the approach requires tunneling to avoid routing loops [27, p. 3]. The tunnel starts from the APR and is terminated at the egress site-exit router that further uses extra information of the tunnel to make the precise forwarding decision.

While the intra-AS tunnels allow a single AS to evolve independently of others, it is very suboptimal when multiple ASes have adopted the approach a packets have to be encapsulated and decapsulated redundantly. Fortunately, the overhead involved with this map-and-encap approach can be mitigated in the final phase of the evolution where pairwise agreements are expected to emerge between different ASes that employ virtual tunneling. The incentive is that two ASes can start advertise and peer virtual routes directly for each other so that the tunnels do not have to be terminated in the middle but rather are established between the two

[14]NATs also conflict with Internet transparency but this will be discussed in more detail in section 2.1.2

ASes. With time, the number of agreements grow and eventually the Internet gradually evolves towards a separate virtual routing name space.

In general, virtual aggregation introduces a new parallel routing namespace and this mapping between two namespace need to be managed somehow. The evolution approach suggest reuse and extend BGP for backwards compatibility, and to minimize costs related to the mapping infrastructure. In contrast, Location-Identifier Separation Protocol (LISP) working group in the IETF proposes a more revolutionary approach and directly focuses on last phase of the evolution approach. Cisco-driven LISP is a core-edge separation protocol and it implements identity-locator split in border routers. LISP introduces a new protocol for the map-and-encap scheme [70]. For distributing mappings, multiple alternatives have been proposed. For instance, NERD [143] proposed a full mapping table for each mapping host and the approach officially adopted by the working group, ALT [81], is based on partial tables organized into a hierarchical overlay. To reduce initial latency, ALT can be used for piggybacking data packets.

Besides improving renumbering and routing scalability, LISP-capable sites support also site multihoming without modifications to end-hosts[15]. As trade offs, LISP requires new mapping infrastructure, i.e., proxies to interoperate with legacy networks and the overhead for testing aliveness with other connected LISP routers as described in RFC 6115 [146, pp. 9]. The RFC lists also other protocols with different technical details based on core-edge separation and map-and-encap scheme (such as Six/one [231], IVIP [235] and IRON/RANGER [213]), but LISP suffices here as a prime example of protocols in this category.

Global, Site, and End-system address elements (GSE) [169] serves as an early example of locator-rewriting approach based on core-edge separation. The approach splits an IPv6 address into three pieces. The prefix of the address is denoted as *routing goop* which can be changed by routers and it essentially identifies a network. In the middle, *site-topology partition* can locally be used defining different subnets for a site. The remaining part, *end-system designator*, is a globally unique end-host identifier generated from the Media Access Control (MAC) address of the host, for instance. GSE supports site renumbering because the end-hosts are identified with the end-system designator and GSE-capable routers adjust the

[15]LISP extensions to support end-host mobility are being pursued at the IETF [71] that require similar modifications at the end-host as in the various approaches for core-edge separation

routing goop according to the local topology. The approach proposes also new two DNS records, one for the routing goop, and another for the site-topology partition and end-system designator.

GSE does not fully conform to Internet transparency because the routing goop inside the site may be different from the outside, effectively requiring split DNS. At the transport layer, the routing goop and site-topology partition should be excluded from the pseudo-checksum calculations. RFC 4984 [153, pp. 18-20] notes also that the end-system designators change semantics of some applications that compare addresses for equality because they have to compare only the designator part. The RFC further criticizes GSE for undefined locator-failure discovery and raises the question of compatibility with Cryptographically Generated Address (CGA) [18] addresses. The issues with GSE are further analyzed by others [248].

Internet Transparency Solutions

While waiting for IPv6 to become ubiquitous, the IPv4-based Internet is not transparent anymore as different kinds of middleboxes have broken the original end-to-end nature [194] of the Internet. Firewalls are examples of such boxes and some protocols, such as Skype for Internet telephony, have been known to work their way around the firewalls. NAT devices also involve the functionality of a firewall to filter out traffic flows originating from the Internet, but NATs also support address aggregation with private address realms. The aggregation requires the NAT middleboxes to couple transport and network layers tightly[16] as the translation information is stored into transport-layer ports. As NATs constrain connectivity to the client-server model and private address realms make unique identification of hosts based on IPv4 addresses impossible, development of peer-to-peer applications has become more complicated. Consequently, several solution to work around the issues with NATs have emerged.

RFC 2775 [48] describes two practical solutions to achieve better interoperability with private address realms. In the first approach, *split (horizon) DNS* can be used to give a more concise view of a site from the view point of FQDNs. The idea is that the DNS returns private addresses for host name queries originating from the private address realm of the site and public addresses from outside. Thus, the host names are con-

[16]As mentioned also by others [244, pp. 53-54], transport and network layers are not only coupled at end-hosts but also at intermediary hosts as a result of NATs

sistent but map to different addresses depending on the location of the querying host. Nevertheless, this approach does not comply to Internet transparency from the view point of addressing and presents issues for applications caching addresses. The second approach is not problem free either. Application-Level Gateways (ALGs) can modify application-layer protocols to retrofit them with NATs. For example, an ALG is needed when FTP is used in active mode because the FTP server creates a new TCP connection back to the client located behind a NAT device. ALGs can be problematic especially when they are not properly implemented [50, p. 12]. The complexity of ALGs have been investigated and alternative type of NAT architectures have been proposed [37] but never adopted.

By default, NATs block transport-layer connections origination from the Internet unless the operator of the NAT manually open certain ports. However, this requires some expertise from the user and the port can be forwarded only a single host inside the private address space due to multiplexing reasons. For instance, only single host can serve the standard HTTP port for the outside.

To avoid the manual tweaking of the NAT device, two approaches have emerged. First, applications can employ a Universal Plug and Play (uPnP) library to request opening of certain ports in NATs [78]. The protocol is widely supported by different NAT device vendors, albeit is can sometimes be disabled by default. It is also supported by various manufactures of home multimedia devices and smart phones because it supports broadcast-based discovery of local-area services[17]. As a second approach, IETF is standardizing Port Control Protocol (PCP) [238]. Among other things, it improves upon uPnP as it can infiltrate through multiple cascading NAT devices.

As vendor-specific solution, "Back to my Mac" (BTMM) service [56] supports NAT traversal for OS X with the help of uPnP [78] and other protocols. BTMM introduces topologically-independent, ULA-based addresses [100] that can be used for addressing hosts behind NATs. The approach employs DNS to support end-host mobility and easier naming of hosts, and a combination of Kerberos, IKE and IPsec to facilitate security. BTMM inherits its restriction from uPnP; NAT traversal fails when uPnP is disabled from the NAT or in the presence of cascading NATs. Also, BTMM supports only IPv6-capable applications.

[17]Another service discovery protocol from the IETF is Service Location Protocol (SLP) [89]

In uPnP and PCP, the idea is that the application requests the NAT to open and forward a port to it. As the success of this depends on the protocol support in the NAT(s), alternative approaches have emerged. The protocol family of Session Traversal Utilities for NAT (STUN) [191], Traversal Using Relay NAT (TURN) [147] and Interactive Connectivity Establishment (ICE) [190] works around the issue and penetrates through the NAT box(es). The trick is that two communicating applications simultaneously send transport-layer datagrams to each other to create state in their NAT middleboxes, thus bypassing ingress filtering in NATs [207]. As deployed legacy NATs do not operate in a uniform way [17, 85], the success NAT traversal is not always guaranteed and especially TCP [58] has somewhat slimmer chances than UDP in practice. However, the UDP-based transport has a higher chance of succeeding than TCP [86, 75] in practice.

As a brief summary of the protocol family, an end-host uses STUN to learn its address-port mappings from STUN infrastructure deployed in the Internet. ICE protocol uses and extends the STUN protocol format to support penetration of on-path NATs. The end-host may also utilize a TURN relay to guarantee successful NAT traversal in the event the penetration fails. While ICE restores end-to-end connectivity, it can be argued that is does not support Internet transparency in an architecturally clean way as it does not untangle addresses from ports. ICE is pervasive in the sense that it requires the application to use its APIs and it changes the semantics of the application-layer protocol as the application exchanges the address-port mappings, triggers the ICE penetration procedure and has to be able to demultiplex STUN traffic from its own application-layer traffic. Nevertheless, the protocol family has been adopted to SIP [38], P2P-SIP [110] and Real Time Communication on the Web (RTCWEB) [8], to mention few examples.

Teredo [101] is less pervasive than ICE because application do not need modifications for it[18]. In Teredo, the application uses a special surrogate IPv6 address when NAT traversal is desired. The local Teredo software tunnels transport-layer traffic sent to the virtual address over UDP and tries penetrate NAT devices transparently from the application. The penetration procedures are similar as with ICE albeit Teredo has somewhat lower probabilities for establishing direct end-to-end connectivity.

[18]On older Windows versions, the application had to enable Teredo explicitly using a socket option

Teredo supports Internet transparency in an architecturally cleaner way as Teredo introduces new virtual namespace for end-hosts without dependencies to transport-layer ports. However, Teredo addresses are not persistent because address is formed in a topological-dependent manner. As further limitations, the Teredo requires an IPv6-capable application at both ends and the tunneling involves a small penalty to the MTU.

To restore Internet transparency, an early research-oriented approach called 4+4 [222] extends the NAT-based architecture of the Internet. In the approach, a host is uniquely identified both using its private IPv4 address and public IPv4 of its NAT. At the network layer, 4+4 employs stateless IPv4-over-IPv4 tunneling to store both of the address types and 4+4-upgraded NAT devices translate the addresses as illustrated in figure XX.

XX draw figure

The described implementation intercepts DNS requests at the client-side host, injects new requests for 4+4-specific Service Record (SRV) records to detect 4+4-capable servers. For 4+4-capable servers, the implementation caches the mappings between a private and public address but it modifies the DNS responses so that the originating legacy application receives the private address. This way, a site can choose its internal addressing convention and changing of the ISP can become easier. Also, private address spaces aggregate better and thus improve routing scalability.

As another research solution to Internet transparency, mobility and multihoming, Delegation-oriented Architecture (DoA) [232] proposes an architecture based on core-edge elimination. The idea is to introduce a new shim layer at the end-host between existing transport and network layers that translates upper-layer identifiers to routable locators. The approach is based on tunneling; the identifiers are stored in an additional header inserted between the transport and network layers. The architecture revives Internet transparency with its persistent identifiers and the tunneling approach shields application port numbers from changes of NATs. Further, DoA-capable NATs are required to rewrite only IP addresses in packet headers and use the additional identifier information from DoA headers as additional demultiplexing tokens.

DoA requires the identifiers to be globally unique but not necessarily cryptographically derived. The identifiers are flat and such non-hierarchical identifiers are difficult to look up from the DNS, so the architecture relies on a DHT service for storing them. DoA introduces extensions to the Sockets API to accommodate the 160-bit identifiers as they are cannot fit existing structures.

However, what really makes the DoA architecture distinct from many other proposals is the support for secure, loose source-address routing. To accomplish this, an end-host announces its own identifier as well a chain of identifiers of its middleboxes in DHT. In order to the reach the end-host, other hosts have to follow the chain in the specified order. Each of the middleboxes include cryptographic information in the traversed packets so that the end-host can verify the chain. This facilitates both on-path and off-path intermediaries that can support, for instance, virus scanning and firewall services. Off-path intermediaries are useful especially for DoS prevention. However, the trade off of such delegation is a fairly complex DHT registration procedure and extra interaction between the intermediary hosts.

NUTSS [87] architecture is also based on core-edge elimination but unlike DoA it argues that applications should be using more aggregatable Universal Resource Identifier (URI)-based names instead of flat names. A fully-fledged NUTSS introduces a new API for network applications but the implementation supports also a legacy-compatible mode where a shim layer translates routable IP addresses from the application layer into NUTSS-based identifiers. Similarly as DoA, also this architecture achieves mobility, multihoming and Internet transparency by providing applications with persistent identifiers. However, it also holds a promise for supporting also anycast and multicast.

In NUTSS, transport-layer connections are ephemeral and recreated on demand. In fact, NUTSS acts as network-application framework that abstracts away the lower-layer details and paves way for extensibility as a NUTSS-capable end-host can negotiate the support for different protocols (IPsec, TLS, IPv6) using the control plane of NUTSS. The control plane supports steering of middleboxes; similarly as in DoA, the end-host registers explicitly to middleboxes that can reside on or off path[19]. In contrast to DoA, the "boxes" handle either control and data plane. Control plane boxes are handled by off-path "p-boxes" that are organized into an overlay and can be used to negotiate, for instance, access-control policies. M-boxes are typically on the path and forward data plane when is has been

[19]The explicit negotiation between all intermediaries in both DoA and NUTSS appears somewhat reminiscent to circuit-switched networks

allowed by p-boxes. As the p-boxes and m-boxes communicate with each other, they can also support distributed firewalls for the problematic cases where routes between two end-hosts use asymmetric routes. NUTSS also suggests the use of STUN to establish direct end-to-end communications in face of legacy NAT devices.

The crux of the NUTSS architecture is the deployment of the new infrastructure. To mitigate this, NUTSS proposes a three phase deployment plan. In the first stage, public p-boxes are deployed and few end-host applications employ NUTSS with the NAT traversal capability as the driving “killer application”. Then individual networks deploy their own p-boxes and end-hosts learn about their existence via DHCP extensions. Finally, legacy middleboxes are replaced with m-boxes that can also proxy communications for remaining legacy end-hosts.

2.2 Heterogeneous Addressing

One of the fallacies in distributed computing is to assume that the network is homogeneous [82]. This applies especially to addressing, now that IPv4 and IPv6 will have to co-exist during the undetermined transition period to IPv6. Global adoption IPv6 has not been the only suggested way to deal with heterogeneous addressing as some approaches advocate replacing of IP addresses with DNS-based names in applications. Few renegades have embraced heterogeneity instead of homogenization of application layer addressing.

2.2.1 Challenges

Heterogeneous addressing as introduced, for instance, by the separation of IPv4 and IPv6 is problematic for a few reasons. For instance, access control rules double both at the end-hosts and middleboxes, leaving more room for human error. IPv4 address literals are easily forgotten in software configurations of a site and will be discovered only when the site transitions completely to IPv6 (or renumbers itself otherwise). Writing of networking software is more complicated as developers have to deal with the dual versions for DNS and for the actual data transfers. The development is especially hard because the Internet is still in transition phase: a client may discover IPv6 addresses for a server but it is not guaranteed that the client-side network is capable of support IPv6 or that the individual service at server supports IPv6. The client can try each compatible address pair sequentially until it finds a working combination but the user may have aborted the connection by then due to the additional timeouts. Thus, developers may be tempted to avoid IPv6 altogether or they have to employ parallelization for network connections, which further creates unnecessary traffic to the Internet.

It could be stated that heterogeneous addressing involves a challenge similar to the multihoming with its multiple addresses. However, heterogeneous addressing is even more pervasive at the application layer as developers as the Sockets API does not insulate applications from different address formats. Naturally, network application frameworks and other middleware can be used for insulation but a host of software has already been written without them.

Finally, it is worth mentioning that the heterogeneity does not only stem from the introduction of IPv6. For instance, wireless sensor networks comprise of constrained devices with limited memory, processor and battery lifetime. In such environments, IP-based stack has been considered too inefficient and sensor-networking protocols in many of the solutions[20]. However, some attempts to assimilate sensor networks with IPv6 have already emerged [136, 200].

2.2.2 Solutions

While it is possible to implement software agile to accommodate both IP versions, it appears that a substantial portion of networking software is still oblivious to IPv6 and address literals are hard coded into software configurations. Therefore, it can be argued based on existing practices that agility for different address families is difficult to achieve in the present TCP/IP architecture. Thus, mono-cultural attempts to unify address formats have emerged, for instance with IPv6-mapped [99, p. 10] addresses as a replacement to IPv4 addresses but have failed [152].

The IETF community has invested a lot of energy for global IPv6 adoption. As a result, a number of different transitioning solutions have been introduced. To mention a few of such approaches that would facilitate IPv6 at the application layer, for instance, Teredo has been described earlier and other tunneling mechanisms, including manually configured IPv6-over-IPv6 tunnels, 6-to-4 Tunneling Protocol and its extension IPv6

[20]For instance, the proprietary Z-wave protocol for sensor networks is not based on TCP/IP

Rapid Deployment on IPv4 Infrastructures (6RD), Multiprotocol Label Switching (MPLS) (MPLS)-based tunneling, Network Address Translation/Protocol Translation (NAT-PT) and its enhancement NAT64. Instead of iterating through all of these standards, we point the interested reader to a overview [104, p. 22-47] and here we focus instead on NAT64 as it has lately received a lot of attention in the IETF.

In NAT64 [23], the client-side host (or its network) supports only IPv6 and server supports only IPv4. To facilitate communications between these to incompatible address families, the client is located behind a NAT device supporting the proposed extensions and is using a DNS server (or proxy) supporting DNS64 [24] extensions. When the client looks up the address of the server over IPv6 from the DNS, it notes that the server has only a A record configured and decides to synthesize an AAAA response based on a special IPv6 prefix and the address of the server. This avoids the address incompatibility issue and the NAT64 device can translate IPv6 packets from the client to IPv4 packets towards the server. As other benefits, this approach introduces IPv6 to the client-side networks, where NAT deployment typically prevents it, and homogenizes, e.g., client-side access control lists to IPv6. As a drawback, the approach fails with software that employs IPv4 address literals [13, p. 14].

As an alternative to assimilate applications to use IPv6 addresses, NUTSS architecture as described in the previous section proposes URIs as identifiers in its Sockets API extensions. Others have also proposed DNS-based identification [79, 55]. One of most recent ones, Name-based Sockets (NBS) [224] is still somewhat immature, but being standardized in the IETF. Unlike NUTSS, takes a bit more conservative approach and uses only the FQDN portion as persistent host identifiers. In a nutshell, NBS suggest new Sockets API structures that can hold names instead of addresses and the name resolution occurs inside NBS module, not in the application. Thus, NBS offers a solution for homogeneous naming by reusing the existing DNS namespace.

NBS supports also mobility and multihoming. It defines its own control plane for mobility management that tries minimize round trips by piggybacking it into data-plane packets. This design choice limits it to IPv6 because IPv4 options appear to be dropped by a number of existing firewalls [150, p. 339]. Other design constraints exist as well, for example, modifying of all application to use NBS extensions is not trivial and some hosts do not have names in the DNS [214, p. 16]. RFC4177 [103, pp.

20, 31] mentions few other constrains with FQDN-based names. Firstly, services typically employ load balancing by attaching a single FQDN to multiple addresses that belong to different hosts. This means that the FQDN does not uniquely identify a single host. Secondly, FQDN-based identifiers may not survive business mergers or acquisitions as the domain name may change[21].

In contrast to the attempts to unify applications to use either FQDNs or IPv6 addresses, the abstract Plutarch [59] architecture embraces heterogeneous addressing. Plutarch does not try to modify existing Internet architecture but rather nests network end-hosts and intermediaries into two abstract entities. The smallest entity is a *context* that refers to a set of hosts with homogeneous networking capabilities, such as addresses, packet formats, transport protocols or a common service for name look up and storage. As two concrete examples, the context can be a private address realm or an autonomous system. Then, an *interstitial function* chains two heterogeneous contexts together to enable inter-context communications. As examples of interstitial functions, uPnP is its realization for private address realms and BGP routing procedures for autonomous systems. The interstitial function translates addresses and names, transport-layer protocols or even acts as an ALG to modify data for more suitable formats for, e.g. hand-held devices, according to needs of the underlying context. While this is nothing new, the novelty of Plutarch is that proposes a universal programmable API, for end-hosts to chain contexts with interstitial functions with the ambiguous goal of achieving communications across heterogeneous networks. The authors present a sketch of the API for Plutarch but admit it is still a straw-man approach (as it remains unimplemented).

2.3 Insecure Addressing

Insecure addressing is present at all layers of the network stack. In this section, we briefly introduce the challenges and number of solutions. The solutions are categorized to client-side and server-side authentication, communication privacy and availability. It is worth noting that we merely scratch the surface; many interesting topics, such as anonymity, object/data-centric security, intrusion detection and quantum cryptogra-

[21]While the reuse of FQDN-based identifiers avoid new infrastructure, it could be further argued FQDNs are domain specific and a host should not preserve its FQDN it moves to another domain

phy are worth of another dissertation, hence will be out of scope.

2.3.1 Challenges

Insecure addressing is present at the network and link layer in the TCP/IP architecture. IP addresses, as well as MAC addresses, can be characterized that they are forge because typically they typically do not include a secure verification of the ownership. When the attacker is the same network as the victim, the attacker can change its address to correspond to the MAC or IP address of the victim host. As an alternative, the look-up procedures could be intervened. With MAC addresses, the attacker can also employ ARP spoofing. With IP addresses, the attacker could impersonate as the DHCP server or try to send forged DNS responses. Alternatively, applications resorting solely to IP addresses as their only access control method can be tricked to receive traffic from other IP addresses on multihoming hosts [214, p. 15, 19] when the underlying host employs so called *weak end system model* [39, p. 63].

2.3.2 Solutions

While IPv4 is prone to ARP spoofing, the issue is resolved in IPv6 using SEcure Neighbor Discovery (SEND) extensions [12]. However, neither IPv6 or the extensions are largely deployed. As MAC and IPv6 addresses still remain trivial to forge, they are unideal as authentication tokens and the problem is pushed up in the networking stack in many scenarios. As transport-layer security are not really deployed in the Internet, the authentication is usually implemented at the application layer. The layer where security is implemented defines also the granularity, that is, application-layer solutions typically enjoy fine-grained granularity than solutions operating on lower layer.

Instead of modifying individual applications to support security, low-level solutions can be used to protect entire legacy networks more efficiently, albeit not without trade offs. For instance, IP-based firewalls or Virtual LAN (VLAN) tagging in routers can be used to isolate networks from each other. However, they offer little protection against “insider attacks” [31, p. 12-13], i.e, a user can be tricked to install malware on the underlying computer that bypasses the firewall and further infects the entire network. In other words, such firewalls offer only topological protection. This can be challenging with mobile hosts without permanent IP addresses and, in practice, it is common to use a VPNs or web proxies to access firewall protected intranets. Also, another challenge with firewalls are multihoming sites that can result in asymmetric paths.

Contrary to low-layer security, a benefit of application-layer security is that the application is aware of security and can convey this information to the user. Typically, it is also easier for the user to carry the security credentials, such as passwords, However,, application-layer security inherits all of the weaknesses of the lower layers. Implementing security redundantly at multiple layers offers more protection but can have a negative impact on performance.

In practice, a number of factors limit the impact of at attacks against addresses, at least when the attacker is off the communications path. For instance, the NAT devices drop new traffic flows arriving to the private address realm by default. Then, as the identity and location of a host are coupled, a malign host cannot claim to be the victim host, at least when it resides in another network than the victim. Source address spoofing can be difficult achieve as many routers and firewalls drop packets originating from the incorrect network. However, a compromised router or a WLAN access point allows man-in-the-middle attacks and the mentioned measures cannot protect against such on-path attacks.

Client-side Authentication

A client-side host can be authenticated with the server or any of the on-path middleboxes, such as local wireless access point, router or firewall, or to a server. Section 2.1.2 also mentioned two research-oriented approaches, DoA and NUTSS, that supported off-path intermediaries, so they will not further discussed here.

As MAC-based and IP-address based access control easy to circumvent, they are fortified with other means. For instance, a host can generate a private-public key pair and hash the public part of the key along with some additional parameters to generate a self-certifying IPv6 address. This technique is called CGA [18] and is employed in IPv6 by SEND [12]. While SEND protects the ownership of an address, it does not prevent a host from claiming an unreserved address, possibly even belonging to entirely different topology. For instance, such exploitation can be prevented with Source Address Validation Architecture (SAVA) [241] by verifying source addresses of egress packets at the varying levels of routers or by requiring the end-hosts to cryptographically verify the packets they send.

At home, it is common to employ password-based authentication for the wireless access point using Wi-Fi Protected Access version 2 (WPA2)-based security [20]. For organizational and corporate use, a myriad of protocols exist [21, 159, 189, 43, 3, 14] both for wired and wireless authentication. In such environments, the user is usually directed to web portal to input user credentials. Further, the authentication may be valid across valid multiple web-based services if the sites collaborate with Single Sign-On (SSO). Again, various different schemes to implement SSO exist [206, 90, 216].

Few other techniques could be mentioned as well. In Identity-based Cryptography (IBC), any publicly-known string, such as email or even IP address, can be used to represent the user's public key for signing or encryption [22]. Most of the complexity of public key management and certification is hidden into a Trusted Third Party (TTP) service. In contrast, purpose-built keys [40] require no infrastructure but are merely ephemeral identifiers based on public keys that merely assure two communicating end-points remain the same throughout communication session [144, p. 9]. Widely deployed Trusted Platform Module (TPM) [121] is designed for storing of sensitive information, such as private keys, on tamper-proof hardware residing on end-host. This way, applications can request the hardware for signatures but a compromised software cannot steal the private key.

Server-side Authentication

TLS [61], and its predecessor, SSL [80], are the de facto way for securing and authenticating TCP transactions especially in the web. While TLS supports also client-side authentication, only the server is typically authenticated by the client and some of the methods described in previous section are used for authenticating clients. TLS authentication is based on certificates signed by Certificate Authoritys (CAs) in the Public Key Infrastructure (PKI) hierarchy. Consequently, TLS meets the challenge to avoid server-side impersonation attacks in an address-independent way, albeit not offering any remedy to non-persistent or heterogeneous addressing. Koponen et al [128] have implemented client-side mobility extensions for OpenSSL, an open-source implementation of SSL/TLS, but the extensions were not officially adopted to the implementation nor standardized.

TLS runs on top of TCP. Followed by the TCP handshake, TLS handshake requires two round trips in the basic case or one round trip when a connection is resumed, i.e., when the client has cached information on a previous session[22]. TLS requires a different port when a service supports both insecure and TLS-secured communications to avoid man-in-the middle attacks [61, p. 34]. The application to be modified to use the TLS-specific APIs instead of Sockets API and in this way, the application is always aware when the TLS-based security is being utilized. The APIs are implementation specific and number of implementations exist, including open-source libraries OpenSSL and GnuTLS.

dTLS [187] offers protection for UDP-based communications. In essence, it is an adaptation of TLS to the limitations of UDP that does not guarantee packet delivery nor ordering. Therefore, it's security and addressing characteristics are the same as with TLS. While the datagram-oriented nature of UDP does not prevent applications from sustaining address independent data flows, security is still a concern. Thus, secure extensions to support mobility in dTLS have been defined [198].

FQDN-based names can be considered as means to authenticate services but basic DNS offers little protection particularly against man-in-the-middle attacks. To fill in this gap, DNSSEC [11] cryptographically authenticates DNS responses, albeit it does not really assure anything about the "identity" of the service (which is usually accomplished with TLS). With support for certificates, DNSSEC has the potential to be used as a PKI when it is more widely deployed. It is also worth noting that hosts may also dynamically update their own records in the DNS [230] in addition to look up and extensions for securing the updates exist [234].

Protection of Communication Content

A malign host can compromise communication by different means. It may be able to breach the confidentiality of the data by reading the content of datagrams, modify individual datagrams, forge the originator of the data or replay recorded datagrams. TLS and Datagram Transport Layer Security (dTLS) as already described in the previous section offer support for such security at the application layer, as well as SSH. Finally, IPsec [117] is network-layer security solution.

In SSH, a server create a private-public key pair for itself and is authenticated using its public key. A client authenticates itself to the server us-

[22]As a curiosity, Google has incorporated a TLS extension to its Chrome web browser [142] to permanently fix the round trips to one for Google services and has promoted also zero round trip extensions [141]

ing a username-password combination or with a user-specific public key. Instead of showing entire public keys, SSH prompting employs as fingerprint, i.e., a hash calculated over the public key, for the convenience of the user. SSH secures communications between the client and server using symmetric keys that are created using a Diffie-Hellman (DH) key exchange. Implementations of support at least terminal sessions but typically also VPN-like tunneling of any kind traffic that has to be set up manually by the user.

SSH requires a client-side and server-side application to be installed but does not require any additional infrastructure, which has largely attributed to its wild success. Albeit is has optional DNS records [196] for storing fingerprints, they are not commonly used and therefore SSH offers weaker security based on Leap of Faith (LoF) [15, p. 5]. The weakness is that is prone to man-in-the-middle during the first connection attempt, during which the client learns and caches the public key corresponding to the host name and IP address of the server. When no prior key exist or they key has changed for the server, SSH client prompts and warns the user. This is a crucial aspect because the user can verify the key of the server but most importantly protects against further attacks: the middleman has to be on the path every time or otherwise will be exposed, thus inflicting *asymmetric cost* [15, p. 5] to the attacker.

Besides asymmetric cost, LoF is relies on *temporal* and *spatial separation* [15, p. 3,5]. Temporal separation guarantees only the server remains invariant but does not guarantee that it was the correct server the first time. Spatial separation assures that the host is on a specific communication path. In the case of SSH, this means that public key of the host is coupled with its host name (or IP address). If this binding changes, the SSH connection fails. Therefore, it can asserted that security based on spatial separation as employed by SSH does not really support persistent addressing. Koponen et al [128] have implemented client-side mobility extensions for OpenSSH, an open-source implementation of SSH, to recreate new TCP sessions with the server. Unfortunately, the extensions have not been adopted to the implementation nor standardized. As an alternative approach, Winstein et al [240] have redesigned SSH from scratch to sustain mobility for terminal sessions and to support better interactivity with the user on top of UDP.

IPsec offers network-level protection for data flows. Compared to SSH, applications are not typically aware when the communication is secured.

IPsec is based on symmetric key cryptography based on unidirectional keys that are denoted as Security Associations (SAs). As setting up manual keys can be cumbersome for users, this task is usually automatized using a key-management protocol that negotiates they keys dynamically when an application sends traffic matching to a certain Security Policy (SP).

IKEv2 [67] is typically used as gateway based VPN that does not require any changes to the server side. The protocol has two phases where the first phase is DH key exchange that set ups symmetric keys to secure a control plane. The second phase uses the control plane to set up symmetric keys for the data plane which is a IPsec-based tunnel between the client and the gateway. Basic IKEv2 does not support end-host mobility or multihoming but Mobile Internet Key Exchange (MobIKE) [66] extensions fill in this gap for the client side.

The basic version of IKEv2 based on "strong" authentication that essentially requires a separately deployed PKI. To avoid the management overhead and scalability concerns involved with PKI, so called Better Than Nothing Security (BTNS) [217, 237] have been standardized. BTNS offers two methods of operation with their own trade offs. Stand-Alone BTNS offers a lowest level of protection that can be subverted by a man-in-the-middle attacker. This method merely guarantees that the other entity does not change during communications and is based on anonymous encryption [15, p. 4]. As such it is mostly suitable to be used with public services in the absence of a stronger authentication at the network layer. Channel-bound BTNS avoids middleman attacks but assumes that the application layer supports strong authentication, thus making strong authentication unnecessary at the network layer. To communicate the success or failure of this authentication to IPsec, a separate API between the application and IPsec is needed [217, pp. 6-8] to "bind" the security mechanisms together which is also referred as *channel binding*. Further, the same API can be used for fortifying BTNS-based security to introduce transport-layer specific SAs. This process of *connection latching* [236] ensures *fate sharing* so that IPsec associations are purged when the corresponding transport-layer connection terminates in order to avoid certain abuse [179, pp. 342-343]. As the application can prompt the user and cache information on unauthenticated credentials on behalf of BTNS, this method can achieve LoF security similar to SSH [217, p. 23],[179, p. 347].

Prevention of Unwanted Traffic

Unsolicited traffic is a nuisance at many levels of the networking stack. At the application layer, email spam[23] is nuisance that involve directly the end-users. Traffic flooding using Denial of Service (DoS) or Distributed Denial of Service (DDoS) [154] does not directly involve the end-users but is visible to them a degradation of QoS. In this section, we take a brief glance at few solutions to email spam and DoS prevention.

A recent survey [51] describes a number of spam prevention techniques. To mention few examples, black listing assumes that email relays are benign by default and email relays end up to the blocking list when they are reported of sending spam. White listing works exactly in the opposite way and assumes everyone is untrusted by default. It is common to employ machine learning techniques, such Bayesian email filtering, either at email clients or by the email service provider in the case of web-based email. In greylisting [133], a server receiving email will request the originator to retry after a while and this is effective as spam relays do not usually retry.

The spam problem has been analyzed also from an economical perspective by others. Goodman et al [83] model costs for spam prevention with human-interactive proofs[24] and computational puzzles. The authors prefer the latter method and show that it is not necessary to persist a proof or a puzzle forever for each email message sent but it merely suffices to apply it only in the beginning for every Nth message. Finally, Levchenko et al [115] show that the payment infrastructure is the bottleneck of the spam value chain and argue that spam could be tackled most efficiently by political means, i.e, enforcing a payment tier.

DoS attacks can be migrated, for instance, with Packet Level Authentication (PLA) [53, 140]. PLA introduces a new shim layer between transport and network layers at end-hosts. When a host sends a packet, the shim layer signs the packet and attaches a certificate[25]. This way, PLA-capable routers can authenticate and authorize the packet based on TTP that certifies the public keys of the end-hosts. For instance, this facilitates source address verification and DoS attacks can be prevented by revoking of the certificate. As such, PLA does not change the addressing model of the Internet in any way but remains compatible with other approaches that, for

[23]Countermeasures for VoIP spam [51] are out of scope

[24]Also known as Completely Automated Public Turing test to tell Computers and Humans Aparts (CAPTCHAs) or Reverse Turing Tests

[25]Alternatively, the certificate can be omitted from subsequent packets if the routers cache it

instance, implement end-host mobility [139, p. 29].

2.4 Deployment Considerations

RFC 5218 [215] characterizes a number of properties of successful protocols. One critical factor is that the early adopters should get the benefits of using the protocol. As in Mobile IPv4 or IPv6 (MobileIP) and IKEv2, one way to meet this goal is to employ gateways or proxies that terminate the client-side connections so that users there is no dependency on server-side deployment. On the other hand, avoidance of mandatory, additional infrastructure altogether can also be a recipe for success as it has been the case for SSH.

It is should also be noted that IETF always mandates a security considerations from the standardized protocols. This is required even when the target scenario would be trusted – wildly successful protocols can easily become reused beyond their original purpose [31, p.5].

While full economic analysis is out of scope, it is difficult to escape it entirely in deployment considerations. Regarding to routing scalability, Jen et al [108, p. 4] argue that cost of deployment is better aligned with the benefits with core-edge separation because it is the transit networks that are facing the scalability problem. However, a deployment at the edges of at least two sites is required even with a core-edge separation to gain some benefits of the adoption. Then, the only practical difference to core-edge elimination is that it has to be deployed for all end-hosts of the two sites. Thus, adoption of a elimination approach can be considered slower and, as suspected by Jen et al, may arrive too late as routing tables may exceed a critical threshold. In the context of this dissertation, elimination approaches will nevertheless be accepted as a viable technical solution for site renumbering despite of this claim.

Another protocol design consideration is incremental deployment without a flag day. This usually requires backward compatibility on end-hosts, even with IPv6 stacks which are already considered legacy [153, p. 28].

For realistic deployment of a protocol, the constraints as posed by middleboxes should be considered in the protocol design. For instance, the Internet is still ossified by IPv4 [7, p. 206] even though IPv6 is slowly making some progress[26]. NATs and firewalls pass only TCP and UDP

[26]Around the globe, the world IPv6 day was organized for the second time on 6th of June 2012 to permanently enable IPv6 in the products and services of major ISPs, home networking equipment manufacturers and web companies

traffic by default [214, p. 16], and in some cases only HTTP traffic on top of TCP [184]. They might also allow only one side of the communications to initiate which further requires NAT penetration procedures in the case of P2P applications. Some firewalls drop also ICMP messages and majority of firewalls drop IPv4 options but TCP options are not usually filtered. Designing protocols that pass IP address literals by default is a condemned idea due to ubiquitous NATs.

A common misnomer about DNS is that deploying new records to DNS is difficult because it requires modifications to DNS software. On the contrary, modern DNS software is actually quite flexible. For instance, the most popular DNS service implementation, *Bind*, supports non-native DNS records types specified in a (hexadecimal) binary format.

2.5 Host Identity Protocol

As most of the collection of articles for this dissertation are based on HIP, it will be introduced in more detail than the other protocols in this section. HIP [167] is an approach based on the paradigm of identity-locator split and the conventional use of HIP classifies it to core-edge elimination category. It introduces a cryptographic namespace to identify end-hosts that is managed by a new shim layer between transport and network layers.

HIP working group has been standardizing the protocol in the IETF and is in the process of moving the experimental RFCs [157, 158, 112, 138, 137, 163, 162, 124] to the standards track[27] during the time of this writing. In a nutshell, the most important updates were related to improved security to facilitate dynamic negotiation of the employed cryptographic algorithms and to introduce Elliptic Curve Cryptography (ECC) extensions [183].

2.5.1 Persistent Identifiers

HIP achieves persistent identifiers by introducing a new namespace for the transport and application layers that is decoupled from the network layer addresses. The identities are managed by a new logical layer between transport and network layers[28] that manages the bindings between the identifiers and locators as illustrated in figure XX.

[27]`http://datatracker.ietf.org/wg/hip/charter/`

[28]In RFC 5533 [168, p. 9] terms, HIP is located between IP endpoint sub-layer and IP routing sub-layer

XX FIX: stack figure

The new namespace is based on public-key cryptography. According to HIP terminology [157, p. 5], an abstract identity is referred as a Host Identity, where as a Host Identifier (HI) refers to concrete representation format of the corresponding identity, that is, the public key of a host. The end-host is responsible of creating the public key and corresponding private key for itself. This way, an HI is self-certifying and statistically unique.

A HIP-capable host creates two other compressed representations of the HI as public keys are of variable length and, thus, unsuitable to be used in fixed-length headers at the HIP control plane and incompatible with legacy IPv4 or IPv6 applications. The format for the control plane and IPv6 applications is the same: the host calculates a hash over the HI to fit it into an IPv6 address and sets a special 28-bit ORCHID prefix [164] for the generated IPv6 address called Host Identity Tag (HIT). For IPv4 applications, the host assigns locally an IPv4 address, called an Local-Scope Identifier (LSI), that acts as an alias for the HI[29].

A HI and also the corresponding locator can be stored in the DNS [163] or in any other suitable directory, such as DHT [6, 227]. However, a practical limitation with the hierarchical DNS is that flat identifiers as employed by HIP cannot be reverse look upped from the DNS unless some organization takes responsibility of the entire HIT prefix in the future [182]. As reverse look up is not guaranteed, a legacy application that has cached a HIT may not be able to connect to it especially when the HIT belongs to a host located in a different domain. The caching issue can raise from the use of address literals in configuration files or when the HIT is passed from host to another in an application-layer protocol as a referral.

Applications assume stable addresses [214, p. 11] as the Sockets API does not expose the TTL values from DNS to applications. In the absence of a deployed solution for this, the identifiers as introduced by HIP can be used to better meet this expectation even in unmodified legacy applications.

In HIP terminology, the host that first contacts the other (by delivering a datagram) is called the *initiator* and the contacted host is called the *responder*. In other words, typically the initiator is the client-side host and the responder is the server-side host. The roles are used by the state

[29]The LSIs are assigned from the private address blocks but implementers have been also experimenting with the unassigned 1.0.0.0/8 prefix. See also [179] for security advice on LSI implementation

machine of a HIP implementation during the set up phase of the control plane that is called the *base exchange*. It is a key exchange procedure that authenticates the initiator and responder to each other using their public keys. The exchange consists of four messages during which the hosts also create symmetric keys to protect the control plane with Hash-based message authentication codes (HMACs) [223]. The keys can be used to protect also the data plane and IPsec [112] is typically used as the data-plane protocol, albeit HIP can also accommodate also others [46, 220].

The base exchange includes also computational puzzle [19], that the initiator must solve. The responder chooses the difficulty of the puzzle which allows a responder to delay new incoming initiators according to local policies. For example, the policy could be to increase puzzle size when the responder is under heavy load.

Figure XX shows in more detail how HIP works in practice when IPsec is employed. In this example, the functionality of the HIP layer has been divided into a DNS proxy and a daemon to manage the HIP control plane. A client-side legacy application first looks up IPv4 (A) and/or IPv6 (AAAA) records of a server at the initiator. The DNS request is processed by locally installed DNS proxy that intercepts the request. In addition to the records requested by the application, the DNS proxy requests also HI records. To remain compatible with legacy servers, the proxy returns the records as they were returned from DNS when no HI records were found. However, the DNS proxy masks the original requests when the DNS response included HI records. To be more precise, the DNS proxy translates a HI and returns either an LSI or a HIT depending on whether the application requested A or AAAA records. Then the application delivers data to the identifiers but this will be intercepted by the IPsec module that blocks the data and requests HIP daemon has completed the base exchange. Upon completion, the daemon set ups the symmetric keys for IPsec as negotiated during the exchange. Finally, IPsec unblocks the application data flow and encapsulates and protects the data flow between the applications residing at the client and server.

XX FIX: draw bex+dns figure

End-Host Mobility and Multihoming

As the identifiers in HIP are not routable, HIP layer translates them into routable addresses, or locators, as they are called in the HIP literature. The translation occurs within a new shim layer located between the transport and network layers. As the application and transport layers are bound to the persistent identifiers, the HIP can dynamically manage the mappings to the network-layer locators as shown in figure XX. Thus, HIP layer can manage both end-host mobility and multihoming [162] in a seamless way. Multihoming is typically employed for fault-tolerance purposes albeit load balancing [180, 88] has been researched as well.

XX FIX: draw a bindings figure (reference: NDSS jukka)

At the application layer, legacy applications assume one address per interface [214, p. 12] despite multihomed devices are commonplace. For such applications, HIP can be utilized to mask the multiple local addresses behind a single, surrogate identifier[30].

At the lower layers, HIP-based handovers require three messages and are all protected using an HMAC and public-key signature of the originating node. First, the mobile node announces its corresponding node of its new set of locators. Then, the each of the corresponding nodes reply by sending a message with a nonce (i.e, a random number) to the mobile node. Finally, the mobile node completes the handover by echoing the nonce back to the corresponding node. This way, a corresponding nodes can avoid replay attacks by verifying that the mobile node has the address it claims to have. This verification procedure is commonly called *return routability* test or check which is also implicitly present already in the base exchange as well.

HIP based handovers have some limitations that are solved with extensions. Unless certain TCP extensions [63, 197] are used to optimize it, TCP is problematic with disconnectivity periods longer than a few minutes because its time-out mechanisms are independent of HIP. In scenarios where supporting double jump is mandatory, two communicating hosts can lose contact with each other and a *rendezvous server* [137] may be used as a contact point as it has always a fixed IP address. The rendezvous server relays only the first control message and the communicating end-hosts communicate directly with each other after this.

Site Renumbering

During a site renumbering, the multihoming capabilities of HIP can facilitate a more seamless transition. HIP can be suitable solution for site renumbering because it provides persistent identifiers for the hosts within a site. During business mergers, the identifiers can be hard-coded literals

[30]Naturally, this assumption holds only if the underlying host offers only a single identifier for the applications

forgotten in various software configurations even when the domain name of a site changes during the merger. As another site renumber use case, the site can merely change its ISP. With a publicly addressable site, the identifiers persist change while the underlying locators change. A site employing a private address realm for its locators in addition to HIP can have the benefit of a aggregatable address space without limiting its connectivity because HIP provides NAT traversal capabilities with its persistent identifiers.

When a site changes its ISP, the HIs can have a long TTL in the DNS records. For publicly reachable locators, DNS caching issues can be avoided by two alternative means. Either the TTL should be short or the locators of rendezvous servers can be employed as they can have a long TTL value in the DNS. To avoid readdressing of the rendezvous server itself during the change of the ISP, such servers should have locators external to the site.

Internet Transparency

As an HIs and HITs are statistically unique, they can be used to distinguish and identify end-hosts in overlapping private address realms. Thus, HIP can be used to restore end-to-end connectivity in the Internet. However, IPv4-based NATs introduce two additional challenges. Firstly, they typically block all other protocols than TCP, UDP and ICMP. This issue is trivially resolved by encapsulating HIP and IPsec traffic to UDP. Secondly, NATs have introduced asymmetric reachability as they can block all new incoming data flows, including the HIP base exchange even though it would be UDP encapsulated. As rendezvous servers have public addresses, they could be used to penetrate NATs but unfortunately this fails especially when both of the communicating end-host are located in different private address realms. For such scenarios, either Teredo can be combined with HIP [228, p. 114] or the native extensions for HIP [124] can be utilized. Besides NAT penetration procedures, the latter alternative includes also relay extensions that guarantee relaying of the control, data plane or both, in scenarios involving NATs resistant to peer-to-peer communications [207, 87].

2.5.2 Heterogenous Addressing

The LSIs and HITs facilitate IPv6 interoperability at the application layer [245, p. 3],[94, p. 15] as one end-point of a communication can use an LSI and the other end-point a HIT. IPv6 interoperability is also supported at the network layer because the mobility and multihoming mechanism support cross-family handovers [113, 229]. Thus, it can stated that HIP supports heterogeneous addressing especially for end-hosts.

Due to the introduction of IPv6, access control lists double in network equipment such as in firewalls. This can be avoided in HIP-aware middleboxes, including firewalls and also rendezvous and relay servers, by enforcing homogenized identifier types. For instance, a HIT can be used to identify a host independently of whether the network-level connectivity is based on IPv4 or IPv6. Thus, the management of these middleboxes may be simplified when resorting to single type of identifier such as HITs, again accommodating heterogeneous addressing.

2.5.3 Secure Addressing

The base exchange authenticates two hosts to each other with their public keys and implicitly tests for return routability. The public keys of the two hosts are present also in the application layer in a compressed format as HITs. This way, application layer security can be bound to lower layer security to support implicit channel bindings [95, p. 12]. In other words, the data flow of the application shares faith with underlying public key based authentication: the data will either be delivered to the destination host possessing the private key or the data delivery fails.

As HIP does not necessitate modifications to legacy applications, it improves connection security in general for them as data plane is typically protected by IPsec. Some applications or services implement access control based on IP addresses. For this class of applications, the user or application can gain more confidence on the security level by populating the access control list with HITs (or LSIs) instead of routable IP addresses. However, other legacy applications may not improved in this implicit way and the user may not be aware when HIP-based security takes place [179, p. 13], thus the application may have to modified to become aware of HIP[31]. This way, the application can involve the user in the decision to approve communications[32] who can verify the other party out of band [179, p. 16].

A native API for HIP [123] makes the channel bindings more explicit to modified applications that can explicitly request the DNS resolver to re-

[31]Publication II implements an API for HIP

[32]Publication VI fulfills this albeit without modifications to legacy applications

turn HITs and for the application to manually configure HIT-to-IP mappings. The specification also defines an explicit way to use the HIP *opportunistic mode* that facilitates LoF-based security for HIP where the initiator triggers the base exchange without prior knowledge of the identifier of the responder and the initiator learns the identifier during the base exchange. As the use of the opportunistic mode implies weaker security level, the API requires explicit consent from the application to use it.

The opportunistic mode does not meet the requirements for the persistent identifiers as the base exchange is triggered solely based on the topologically-dependent address of the responder. Thus, this fails to achieve the goal of Internet transparency for persistent addressing and is problematic when the responder is mobile – a rendezvous server could be utilized to remedy the situation but this would effectively render the opportunistic mode into a HIP-level anycast[33]. Nevertheless, the mode can be useful, for instance, with publicly-reachable services with stable IP addresses when the extra interaction with DNS is to be avoided[34].

When HIP records are stored in DNS, the DNS responses can be forged by a man-in-the-middle attacker in the absence of DNSSEC. Thus, the DNS can be the weakest link for HIP-based security even when opportunistic mode is not employed. As another weakness, advances in cryptanalysis may also discover problems with certain algorithms used by HIP. This means that HIs generated with the compromised algorithms need to be regenerated and replaced with new HIs, meaning that even persistent identifiers as employed by HIP have a limited life span[35].

Deployment Considerations

In the context of site renumbering, a benefit of protocols based on core-edge separation is that the number of nodes have to be upgraded is constrained as the protocol needs to be deployed to only edge routers. In contrast, core-edge elimination approaches require more upgrades as the number of end-hosts wildly surpasses the number of routers. Thus, the researchers have proposed a number of proxy based deployment models for HIP [192, 105, 247, 96, 151, 165]. However, the various proposals will not be detailed here as the focus in this dissertation is on the standardized,

[33]HIP-based multicast has been studied by others [201, 250]

[34]Depending on the implementation model, e.g., installation of the DNS proxy at the client side or configure the DNS records for servers

[35]This is the reason why HIPv2 will be more agile in negotiation the used algorithms

elimination-based approach for HIP.

To operate HIP in the elimination approach, software has to deployed both at the client and server-side hosts [94]. The most essential component is the management software for the control plane[36]. Assuming the deployment scenario involves protection of application traffic, the data plane needs to be managed somehow – typically, HIP implementations employ an optimized mode of IPsec, called the BEET mode [111], which is supported natively only by the Linux networking stack for the time being[37]. To support HI resolution for legacy applications via DNS, either the libraries supporting DNS requests can be modified to support HIP or a local DNS proxy can be installed on the end-host as described earlier in the example of section 2.5.1. Optionally, a relay server can be deployed both at the client and server side unless, e.g., Teredo is used for NAT traversal. Also, both relay and rendezvous extensions can be deployed in scenarios involving double jump. It is also worth mentioning that routers, switches, NATs, existing firewalls, VPNs and the most popular DNS server software do not require any changes to accommodate HIP.

HIP itself has been researched extensively and explored in the context of specific deployment scenarios, including in environments involving constrained devices [120, 161, 225], SIP [131, 45] and cellular [93] networks, and also in cloud networking [125]. Its principles have been reused in a number of other network research architectures [79, 76, 26]. For commercial purposes, HIP has been adopted in few known cases. At the Boeing airplane factory in Seattle, HIP secures connectivity for mobile robots [171]. HIP has also been used to implement a layer-two VPN [96] in a product called Tofino.

2.6 Summary and Comparison

The earlier sections in this chapter described individual protocols under certain category despite the protocols might fit multiple categories. This section gives an overview of the challenges and the solutions for consolidated naming for easier comparison.

In general, the results are presented in tables where a tick mark labels

[36]Regarding code complexity, the size of a HIP implementation can be half of the size of the corresponding MobileIP implementation with IKE-based security [244, p. 162]

[37]Userspace IPsec implementations for Windows and separate support for FreeBSD do exist, though

a protocol fulfills the property without any doubt, where as parentheses are used when the property is fulfilled conditionally or when the property is optional. The qualities of the protocols are obtained from the literature references but their arrangement as a taxonomy as described in this dissertation is novel.

To keep this summary short, we omit the protocols that do not support all the four properties for persistent identifiers: end-host mobility, multihoming, site renumbering and Internet transparency. Here we exemplify some of the missing properties albeit the list missing properties is not complete. Mobile IP, i3, LISP, Evolution and GSE are excluded because they not improve transparency. This is also missing from M-FARA, SCTP, MPTCP, TLS, dTLS, SSH and IKEv2 in addition to renumbering. DoA does not specify mobility and multihoming support as it focuses on interactions with middleboxes. Similarly, PCP, uPnP, ICE[38] and Teredo are not tailored for mobility, multihoming and site renumbering purposes. In addition, SHIM6, NBS, 4+4 and Plutarch do not completely survive site renumbering as they do not provide static identifiers for applications that cache or hard code them into their configurations. Finally, NAT64, DNSSEC and PLA are essentially tools for IPv6 interoperability or security purposes[39] and do not possess any of the four properties.

The exclusion leaves us with five protocols with persistent identifiers, ILNPv6, LIN6, BTMM, NUTSS and HIP. Their other properties will be compared in the remainder of this section.

Table 2.1 shows how the five protocols accommodate heterogeneous addressing. Legacy application support is divided into three capabilities. The first two consist of rudimentary support for IPv4 or IPv6 addresses. The third one refers to more advanced IPv4-IPv6 interoperability, that is, whether the protocol supports an IPv4 application to communicate with an IPv6 application and vice versa. Next, the protocol may be associated with an API for protocol-aware, or "native", applications that offer relief for the heterogeneous addressing simply by introducing a new homogeneous identifier. Finally, network-layer compatibility in IPv4 or IPv6 networks is displayed in the last two columns[40].

Table 2.2 summarizes benign mechanisms to support security. The first

[38]At the time of this writing, mobility for ICE [239] has been proposed but not yet officially adopted in the IETF

[39]Of course, it is possible to combine different protocols to combine their benefits as is the case with Back to My Mac (BTMM)

[40]Compatibility of BTMM in IPv6-only networks was not documented [56]. Hence, we tried it in July 2012 and it failed to work for no apparent reason

Networking	Legacy application support			Native	Network layer	
protocol	IPv4	IPv6	v4-v6 interop.	new id	IPv4	IPv6
ILNPv6	(✓)	✓		✓	(✓)	✓
LIN6		✓				✓
BTMM		(✓)			✓	
NUTSS	✓	✓		✓	✓	✓
HIP	✓	✓	✓	(✓)	✓	✓

Table 2.1. Capabilities to facilitate heterogeneous addressing in the protocols

column names the protocol in question. The following column refers to a protocol where the client authenticates itself to the server and correspondingly the next column refers to the server authenticating to the server. Here, authentication refers to authentication based on pre-shared secrets (including passwords), public keys or certificates. The final column labeled with confidentiality refers to data-plane encryption and availability to the support prevention unwanted traffic (at least not non-distributed). Again, as a baseline, it should be noted that unmodified TCP/IP does have any of the properties enlisted in the first row.

Protocol	Client auth	Server auth	Confidentiality	Availability
ILNPv6	(✓)	(✓)	(✓)	
LIN6			✓	
BTMM	✓		✓	
NUTSS	(✓)	(✓)	(✓)	(✓)
HIP	✓	✓	✓	✓

Table 2.2. Secure-related properties of the five protocols

Table 2.3 summarizes the different namespace properties of the five protocols when they are used in the context of existing IP-based Internet. After the protocol column, the next column describes whether the protocol is semantically based on a separate, disjoint namespace from the IPv4 or IPv6 address spaces or it is overlapping from the view point of the application layer. The following column describes whether the address space is structured, i.e, based on aggregatable or hierarchical identifiers, or if the address space is unstructured. The last column signifies whether the identifiers are assigned centrally or in a distributed fashion; modified Extended Unique Identifier (EUI)-64 [99, p. 8] addresses are based on centrally assigned MAC addresses and also URIs [33] require central assignment, where as ORCHID [164] and ULA [100] type of identifiers are self assigned and, hence, statistically unique. As a base line, unmodified IPv4/IPv6 addresses are typically structured and centrally assigned, and they are also overlapping because a routable IP address couples the roles of an identifier and locator. In general, it should be noted that unstructured namespaces are subject to referral issues when the underlying name resolution infrastructure supports only structured look ups.

Protocol	Disjoint	Structured	Assignment
ILNPv6	✓	✓	EUI-64
LIN6	✓	✓	EUI-64
BTMM	✓	✓	ULA
NUTSS	✓	✓	URI
HIP	✓		ORCHID

Table 2.3. Technical characteristics of the identifiers in the different protocols

Table 2.4 summarizes the design of the five protocols in general. After the protocol column, the position of the logical layer of the protocol in the TCP/IP stack is identified: in practice, layer 4.5 implies typically a sockets interposition library (or an application overlay) located between the application and network layers (to support legacy applications), layer 3.5 a new logical (and address translating) layer between transport and network layers, and layer 3 solutions are typically involved with ARP or IPv6 duplicate address detection. The next column describes the deployment model: symmetric middlebox deployment at both ends (core-edge elimination) or symmetric end-host deployment (core-edge separation). The following column specifies whether the protocol stores state information on the identity-locator bindings in datagrams (tunneling) or as an extra state at hosts (translation). After this, the next column denotes a protocol that requires changes only at either the client or server side, but not both, thus typically implying an intermediate proxy or gateway. Finally, the last column indicates if the protocol depends on new infrastructure, such as protocol-specific proxies or name look up servers.

Protocol	Layer	Design type	Data plane	1-side	Infra
LIN6	3.5	Elimination	Translation		✓
ILNPv6	3	Elimination	Tunneling		(✓)
BTMM	3.5	Elimination	Tunneling		✓
NUTSS	4.5	Elimination	Translation		✓
HIP	3.5	Elimination	Tunneling	(✓)	(✓)

Table 2.4. Summary of some of the technical design choices related to deployment

It is worth noting few dependencies in table 2.4. The last two columns depend on each other, i.e., a proxy-based deployment requires infrastructure, albeit the reverse is not necessarily true. Also, core-edge separation results in incomplete Internet transparency as the end-hosts cannot address routers directly[41].

[41]Thus, all protocols are based on elimination as non-transparent protocols were excluded from this summary

The five protocols have their own trade offs when deployment is considered. For example, NUTSS is heavily reliant on infrastructure albeit it can support off-the-path services by introducing this dependency. Similarly, LIN6 introduces its own infrastructure for identity-locator mappings even though its generalized identifiers survive well site renumbering events. In contrast, ILNPv6 reuses DNS to store the mappings and also to deal with the simultaneous host movement, i.e. *double jump*, albeit this requires DNS servers to be upgraded to support secure dynamic DNS updates. As a drawback, ILNPv6[42] splits an IPv6 address into identifier and locator formats in a way that exposes the possibly stale locator portion to the application which is problematic with legacy applications caching addresses during site renumbering. Hence, legacy applications should be ported to use the FQDN-based API for ILNPv6. BTMM is a very complete protocol albeit it is still vendor-specific technology. It inherits a weakness from its use of uPnP as it does not work with multiple cascading NATs. Finally, HIP is similar to BTMM except that it is based on a single, unified protocol rather than a collection. In contrast to BTMM, it introduces secure identifiers (HITs) that can be used for authentication in access-control lists of applications and middleboxes. The trade off here is that its flat, self-assigned HITs introduce referral issues that are problematic for reverse resolution, i.e, when mapping a HIT back to FQDN or routable IP address. While the existing IPv4 is also tainted by referral issues due to private address realms and missing records to achieve reverse resolution, HIP can remain architecturally clean if an organization takes the responsibility of managing the entire IPv6 prefix assigned for HIP.

To conclude, this chapter has introduced a taxonomy for consolidated namespace and described a number of solutions that fit one or more of the categories. Based on the fulfilled challenges for persistent addressing, capabilities to facilitate heterogeneous addressing in table 2.2 and the improved security characteristics in table 2.2, HIP appears as a decent match for a consolidated namespace. While the contributions of the individual publications to improve HIP to better meet the challenges have been prematurely mentioned in this section, the following chapter explains the contributions in more detail.

[42]ILNPv4 employs IPv4 options and ICMPv4 that are blocked by many firewalls

3. A Consolidated Namespace for Network Applications, Developers, Administrators and Users

In this chapter, we summarize contributions of the individual publications related to consolidated name spaces at a high level. First, we make a reality check to understand the real state of network applications and verify some of the challenges in section 3.1. Next, we analyze how well HIP meets the challenges of a consolidated namespace in section 3.2 – especially from the view point of a developer – based on the contributions of the publications. Then, we view the impact of HIP to end-users in section 3.3 and continue with a deployment perspective that concerns especially network administrators in section 3.4. Finally, we summarize the contributions in section 3.5 and suggest future work items in section 3.6.

3.1 Revisiting the Challenges for Network Applications

Publication I characterizes the network applications and frameworks in Ubuntu Linux. The goal of this investigation was to understand how open-source network applications utilize the low-level Sockets and POSIX APIs directly or indirectly, and how applications employ security. As the number of C-based software packages using the low-level APIs was relative high (710), they were analyzed only statistically, and four example application frameworks were inspected manually. We investigated challenges related to IPv6, the use of UDP, TLS/SSL-based security and the use of a number of extensions for the Sockets API. In this section, we highlight some key issues related persistent, heterogeneous and secure addressing in the low-level networking APIs based our findings. It is worth noting that the revisiting all aspects for consolidated addressing is impossible within a single publication and, thus, we admit to have merely touched the tip of an iceberg.

As IPv4 address space has been nearly exhausted, IPv6 has become more important. This means that network applications have to be modified to support IPv6 addresses. In the investigation, the number of applications supporting both IPv4 and IPv6 was 26.9%. To support IPv6, both client and server-based applications have to support the two different address families for both name look up from DNS and network I/O. Such use of heterogeneous addressing increases complexity in all applications. IPv6-mapped IPv4 addresses do not really solve the issue of heterogeneity because they do not work with all sockets API functions and are actually considered harmful when they leak on the wire [152, pp. 1-4].

To avoid complexity with the Sockets API, certain applications employ network application frameworks to hide the low-level networking details. Therefore, we also investigated the use of the low-level networking in four frameworks: java.net for java-based applications, Twisted for Python-based software and, Boost and ACE for C++ applications. We discovered an issue with non-persistent use of addresses in all of the frameworks, a problem tainting many of the non-framework applications as well. To be more exact, the problem occurs only in the context of UDP with hosts equipped with multiple addresses. The UDP multihoming problem occurs when a server-based application receives a UDP-based message from a client and responds back without specifying the source address explicitly. Ignoring the source address may result in the underlying networking stack choosing a wrong source address at the server and the client dropping the message as it appears to originate from an entirely different server. As many of the commodity devices of today ranging from handheld devices to rack servers are multihoming capable, the impact of the problem should not be ignored. It is worth mentioning that similar problems are also being addressed at the Multiple Interfaces (MIF) working group at the IETF at a broader scope [36, 233].

While the UDP multihoming issue can be directly solved by fixing the issue in the application, this problem could also be worked around with multihoming extensions that introduce only a single identifier to the application[1]. Some of these extensions support also TCP and allow address changes throughout the lifetime of transport-layer sessions – a vanilla TCP connection cannot survive an address change during communications because it is tightly bound to the IP addresses. UDP is more tolerant against such failures by its disconnected nature but would still require additional application-specific logic to facilitate decouple the data flows from

[1]This applies also to weak end system model mentioned in the previous section

the addresses in a secure way.

To support persistent connectivity for multihoming, e.g., SCTP, MPTCP, SHIM6, HIP, MobileIP or MobIKE can be used. However, SCTP requires some changes in the application and has been adopted only by few applications in Ubuntu Linux. With the exception of MobIKE[2], the remaining three protocols were not yet adopted in vanilla Ubuntu and all four would have been difficult to trace due to their transparency at the application layer.

The most popular TLS/SSL implementation was OpenSSL that was used in 10.9% of the applications. Here, we highlight two security-related aspects with its use. Firstly, the initialization procedures were neglected in many of the applications. For example, the Pseudo-Random Number Generator (PRNG) was seeded properly only in 58.4% of the C-based applications utilizing OpenSSL and in two out of four frameworks. Secondly, setting of the security-related options was popular (53.3%). For instance, 20.1% of the applications explicitly allowed downgrade from TLSv1 to an older version of the protocol, SSLv3. While supporting such backward compatibility during transition periods is beneficial for the end-users, this opens a question why the applications have to deal with such problems in general. In the ideal case, such complexity should be automatically and transparently handled inside the library implementing the security. This lesson applies also to other security protocols with explicit APIs, including HIP [122].

3.2 HIP as a Consolidated Namespace for Network Applications

TLS/SSL had been embraced by the developers, perhaps partly due to its visibility to the applications, and it was only natural to try to repeat its success with HIP. In Publication II, we designed and implemented native APIs for HIP-aware applications. The API implementation required changes both in the system resolver and in the Linux kernel. We integrated the APIs with an example proof-of-concept application and extended the host-specific security model of HIP to allow user or application-specific identities; similar trends appeared later in the Unmanaged Internet Architecture (UIA) [77, 76] architecture.

The native API for HIP meets the criteria for consolidated addressing

[2]Strongswan-ikev2 package had 3582 installations (with rank of 16804) in Ubuntu popularity contest in January 2012

quite well. For persistence, the API uses the location-independent identities of HIP. From the view point of security, HIs are secure by their nature and the API extends the HI concept to application or user-specific identities. While the legacy APIs for HIP accommodate heterogeneous addressing with LSIs and HITs, the native API relies on homogenized identifiers called *end-point identifiers*. Such an identifier hides the different forms of HIs – LSIs, HITs and also application-specified HIs. The end-point identifier resembles a file or socket descriptor, and thus serves as an indirect, local reference to the corresponding HI. In the API, applications manage mappings from the descriptors to HIs either directly or indirectly using the DNS resolver.

While the descriptors certainly introduced some amount of complexity, they were a useful utility in organizing variable-sized HIs into Sockets API structures with a fixed maximum length. Unfortunately, the somewhat "unorthodox" concept of the end-point identifier did not make it to the final RFC [123] due to lack of consensus in the IETF. Instead of the syntactically homogeneous identifiers, the RFC version specifies different tokens for early binding with HITs and for late binding. The former refers to HIs discovered from, e.g., from DNS where as the latter refers to special "wildcards" or, to be more precise, client-side macros to handle outgoing data flows based on the opportunistic mode and server-side macros to accept incoming data flows from unspecified clients.

The native API for HIP provides a cleaner way for applications to control the use of LoF than the transparent way employed in Publication V and, in fact, later research verifies that a separate monitoring API should be available when LoF is implemented independently of the application [179, p. 353].

While applications ported to use the native HIP API can directly interact with the user to inform about the security being employed, this leaves a vast number of legacy applications without such support. In Publication VI, we implemented a graphical firewall to manage and approve HIP-based data flows directly from the end-user. While prototype was far from complete, the experiment was a success in the sense that HIP could be used without any changes to the application layer while improving visibility of HIP to the user.

We conducted usability tests with the graphical interface for HIP to understand how users perceive use of HIP with varying levels of visibility in the context of web. In general, the users perceived the interface reasonably well considering the maturity of the prototype. The experiment included usability tests with HITs that were visible to the application and also based on the transparent LoF implementation. In retrospect, the implemented graphical interface improved LoF security for HIP because a later security analysis from Pham et al. [179, p. 352] asserts that user interaction in LoF security is critical in HIP. Further, the authors state that the credentials (i.e., the binding from the FQDN to HI in this case) of the server should be stored permanently and its deletion should be left only for expert users. Our user interface met the former criteria but relaxed the latter as the deletion did not require special privileges.

To protect application-layer services with HIs in address-family agnostic and secure way, Publication IV experiments with a middlebox-based firewall that can filter HIP control and data plane. Basically, the firewall tracks persistent HIs instead of ephemeral IP addresses of mobile end-hosts, thus supporting access control for services using HIs. Compared to traditional firewalls, a limitation of the solution is that is does not inspect the traffic at the granularity of port numbers at the server side unless the end-hosts employ an unencrypted data plane. As described in the publication, this could be mitigated with service-specific identifiers when a single server host is offering multiple services.

Similarly as in the native API, the HIP-based firewall can be entirely managed homogeneously identifiers that hide details of the underlying network topology and the IP version. Also, the HITs could be used to define more fine-grained policies at the network layer than with VPNs to protect against attacks originating inside from the company; as the HIP firewall policies are based on lists of individual end-hosts (rather than network prefixes), it may be easier to exclude individual hosts out.

Publication III further continued the exploration of mitigation of unwanted traffic but this time focusing on service side. The use case here is mitigation of unwanted traffic in the form of email spam which continues to thrive especially considering that sending spam is cheap as spammers maximize their profits by maximizing spam rates [115, p. 4]. As our proposed solution, we modified an open-source email spam filter to throttle the spam rate with the computational puzzles of HIP. HIP also provides authenticated identifiers that can be easily used for access control but we realized soon that malign hosts could circumvent the access control with short-lived identifiers. As a resolution, we proposed extra application-layer logic to favor benign hosts with long-lived identifiers and penalize possibly malign hosts with short-lived identifiers[3]. Thus, extending the life time of the identifiers was aligned with the goal of supporting persistent identifiers in HIP. Also, our work with computational puzzles complements other research on HIP puzzles [219, 30] and may be considered a more concrete realization of more theoretical work by Goodman et al. [83].

3.3 Impact of HIP to End-users

Publication VI analyzed how end-users perceive security indicators consisting of visual and textual cues. The indicators were implemented for the HIP-based management and prompting user interface, HIP aware web browser and a test website. The indicators (or their absence) in these software modules visualized the communications based on plain IP, normal HIP, opportunistic HIP (as presented in Publication V), SSL over IP, and SSL over normal HIP.

The usability tests revealed that the management interface for HIP connectivity was rather unpolished. Unsurprisingly, we observed that users did not even want to see long and lengthy HITs but rather human-readable names. For a relatively new protocol such as HIP, users preferred familiar security indicators for web browsers, including lock symbols and colored address bars.

It appeared irrelevant for the users to see the difference how the security was implemented, whether it was normal HIP, TLS/SSL or both. They also did not perceive any noticeable difference between normal and opportunistic HIP even though the latter effectively disabled the security indicators in the browser because the opportunistic mode implementation shimmed the presence of identifiers from the application. However, this phenomena could be explained by the presence of the user interface that prompted for new HIP based connections.

We witnessed the recurring fact that security can easily go unnoticed by the users. This applied to the absence and presence of security indicators; users did not report the absence of indicators when connecting to an unsafe site, nor did they report the presence of the indicators when they were not prompted as the HIP of the server was already in the cached by the user interface. Nevertheless, the users ranked security levels correctly; connections intended to be secure were ranked clearly more secure

[3]Undoubtedly, many of the hosts of a botnet can belong to benevolent users. However, perhaps throttling of such hosts can eventually reveal that they are compromised so that they can be acted upon

than insecure ones.

Publication II introduced user-specific identifiers. The idea was that users could transfer the identity from a device to another and the HIP software module would then import the identity. Thus, a HI became a portable user identity.

It should be noted that the approach had its limitations; on-the-fly session migration was not supported as it would have required transferring of the transport-layer state as well [4]. As another limitation, the user-specific identifiers could be imported from any media, including USB memories or disks but importing of user identity to a host involves at least two security risks as mentioned in the publication. On a multi-user machine, the same identity could be used by other users, depending on the security granularity of HIP implementation. On a compromised host, the private key part of the identity could be replicated by the intruder. While the multi-user issue could be solved with fine-grained access control mechanisms, the other issue is more difficult to mitigate especially when the intruder has administrative privileges. The escalation of the situation could be contained by employing a smart card that would securely store the private key and sign data upon requests. At least, this would prevent compromising of the private key.

As outlined in Publication IV, the middle-box firewall could be deployed in WLAN access points to support passwordless authentication for end-users. However, the problem of distributing the keys from the users to the administrators is still present independently of the deployment scenario. As a solution, we suggested the EasyVPN [32] approach that bootstraps IPsec-based VPNs using TLS and web services.

3.4 Deployment Aspects

Any new protocol, whether it be research or industry originated, is subject to the scrutiny of realistic deployment scenario. In Publication I, we discuss the deployment of HIP and other related protocols from the view point of their APIs. While the findings lack longitudinal analysis, the adoption of certain API trends in the latest Ubuntu Long-Term Support (Ubuntu) (LTS) were apparent.

The deployment of IPv6 at the application space was relatively small

[4]It should be noted that session delegation with HIP has been further analyzed by others [98]

as only a quarter of the applications supported it. Partially, this could be explained by the fact that the analysis included also less popular applications with perhaps inactive code maintenance. However, it can be also speculated that the lack of IPv6 support may originate from application developers that endorse homogeneous APIs by clinging into IPv4. As explained earlier in section 3.2, management of IPv6 introduces extra complexity to manage the network connections of applications.

OpenSSL was used by 10.9% percent of the software even through it requires pervasive changes in the networking logic of the application. Other protocols requiring more modest changes were not so popular; SCTP and Datagram Congestion Control Protocol (DCCP) were used only by few applications. While the kernel and Sockets API changes for these two protocols are present in modern Linux systems, it appears that library-based solutions, such as OpenSSL, are more welcomed by application developers. Thus, it could be argued that also the native APIs for HIP will not be adopted rapidly as it is based on the Sockets API rather than a library.

Roughly two out of five applications were setting socket options. Based their popularity, it could argued that API extensions based only on them might have better chances for adoption due to their familiarity with developers. Examples of protocols employing such extensions are MPTCP [195, pp. 8-12], shared multihoming extensions for SHIM6 and HIP [122]. In contrast, native APIs for HIP [123] may experience slower adoption because the extensions are tightly bound to the new and yet unpopular DNS resolver.

According to the coarse-grained estimates, it appeared that roughly two thirds of the applications were selecting IPv4 source addresses explicitly. Assuming IPv6 will be more widely deployed, one could assume a similar ratio for the adoption of IPv6 source address selection. As described earlier in section 3.2, proper source address selection is important also in the context of HIP-based firewalls; ignoring the selection may result in the networking stack to choose a source HIT that will be filtered by the firewall at the middle. If not the firewall, the server-side application may be the source of a failed connectivity; the UDP multihoming bug as described in section 3.1 applies equally to legacy applications when the underlying host provides multiple HITs to the applications.

RFC5887 [49, p. 34] lists a number of potential sources of IPv4 address literals and refers to another study [204] that lists potential address dependencies in 34 out of 257 RFC protocol specifications. Based on empirical study of an IPv6-only testbed, Arkko et al [13, p. 14] described that some network software use IPv4 address literals instead relying on DNS-based names. In contrast, we did not observe such addresses as described in Publication II. However, this may have occurred because our investigation was limited to a static code analysis (instead of run-time one or traffic analysis) and we also excluded configuration files.

As a specific use case study of referrals, we experimented empirically with FTP in Publication II. This particular case did not appear problematic in practice because FTP typically passes addresses as "callbacks" between a client and a server, meaning that addresses are mirrored between a pair of hosts. This is not a problem unless the FTP server redirects to another server based on the IP address. In such a case, a HIP-aware application and application-layer protocol would be required to pass also the IP address to the other host.

While RFC 5218 [215] does not consider adoption issues specifically in the context of APIs, the native API for HIP in Publication II were designed to maximize familiarity for developers; the API included a simple resolver option to enforce the returning of HITs from DNS resolution, in addition to defining extra APIs for more pervasive use. It is also worth mentioning that the extensions for user-specific identifiers in the API were also designed to be compatible with standardized HIP to make their adoption more seamless. Yet, the native API may have a rocky adoption road in front of it as even its minimal use depends on the unpopular resolver.

In general, it would be easier to deploy HIP along with a new system such as P2P-SIP [109, p. 77],[45] that perhaps even already has been adapted to it [47]. Similarly to this, Publication III proposed a use case for HIP where its deployment was limited to server side. More specifically, HIP was deployed only on Simple Mail Transfer Protocol (SMTP) servers because the number of client-side hosts clearly outnumber the SMTP servers. Instead of end-to-end use, HIP was used here in a point-to-point fashion due to the nature of the SMTP service. In order for early adopters to obtain the benefits, the publication advised to introduce more delay for HIP-incapable servers as an adoption incentive. This could facilitate incremental deployment as the deployment of HIP would not require a flag day.

Publication III also clarified that the DNS-related changes are backwards compatible. HIP has new records in the DNS that do not interfere with HIP-incapable hosts and the format can be used without any changes with the most popular DNS server software, bind [94, p. 19]. Also, the look up of DNS records for HIP can be transparently handled at SMTP servers using an local DNS proxy.

Similarly as in the SMTP use case, the HIP-based firewall Publication IV was aimed for a specific group of people in order for early adopters to obtain the benefits. The HIP-based firewall can be used internally, e.g., within single a company in a similar fashion as a VPN. This benefits of the firewall can be useful for the entire company but especially for the network administrators. When HIP is being utilized, the HIP firewall filters based on persistent identifiers, thus facilitating network renumbering when a company changes its ISP or merges with another [5]. The hard-coded HIP-based identifiers in various configuration files for applications and services, can persist topology changes and, thus, cause less down time. Management of the firewall can be more simple for the administrators as separate rules for IPv4 and IPv6 are not needed, thus perhaps reducing the number of configuration errors.

Publication V experimented with the most conservative use of HIP that did not expose LSIs and HITs to the application layer at all. Instead, the applications used regular routable IP addresses. Consequently, no support from the DNS infrastructure was needed in order to deploy HIP. As SSH was successful with its LoF model, we experimented if HIP would be flexible enough to support this as well with its opportunistic mode.

The LoF for HIP was implemented using a shim library between the application and the sockets. The library used the opportunistic key exchange of HIP. The implemented library could be enabled at system, user or application granularity depending on the local policies. The library translated IP addresses of the application to HITs that were further processed by the IPsec module in the networking stack. As applications were not employing HITs, application-level referrals were not an issue when employing publicly-reachable addresses. Despite of the additional translation step from IP addresses to HITs, the library added negligible overhead to the throughput. As a configurable feature of the library, it implemented also a fallback mode [6] that bypassed HIP-based processing after a timeout when connecting to a HIP-incapable host.

As a major design compromise, the LoF effectively trades off ease of deployment at the cost of employing non-persistent identifiers. With the im-

[5]To support external access from other sites without HIP support, e.g., a web proxy is needed [125]

[6]The fall back as a generic mechanism was later also endorsed by others [9, p. 2]

plementation model used, persistence is lost for the initial contact as the opportunistic mode operates without prior knowledge of the HI of the responder. At the application layer, a legacy application witnesses only the initial locators (and not the identifiers) obtained during the connection set up while the actual locators as utilized by the stack can change. As another design compromise, the LoF model also fails to provide persistent identifiers in NATted environments, thus compromising Internet transparency. Also, routable IPv4 and IPv6 addresses are insecure by their nature, despite they function better as referrals (but only in scenarios not involving NATs).

From the view point of security, the DNS resolution is the weakest link in the use of HIP until DNSSEC is widely deployed [56, p. 11]. Thus, during the transition to DNSSEC, the LoF model as presented in Publication V is a viable option because even the non-opportunistic HIP is susceptible to man-in-middle attacks without DNSSEC. Further, while introducing DNS records is fairly trivial and does not cause any conflicts with HIP-incapable end-hosts, the LoF model avoids HIP-specific records altogether. However, the presence or absence of DNS records describes clearly when a host supports HIP or not. LoF does include this capability detection and the comes with the cost of timeouts when used “in the wild”. As briefly mentioned in Publication V, we suggested a solution for the timeouts to reduce the fallback latency to transition from HIP to non-HIP based connectivity. As IP options are typically filtered by firewalls, our solution for this was to exploit TCP options to detect HIP-capable hosts [35, pp. 24-26].

3.5 Summary and Lessons Learned

This section summarizes the contributions of the publications to achieve a consolidated namespace with HIP. The resulting architecture is viewed from the view point of end-users, network application developers and network administrators as illustrated in figure 3.1.

Publication I revisited the challenges and solutions for consolidated naming: non-persistent, heterogeneous, insecure addressing. The statistical analysis was constrained to certain aspects of application-layer solutions as network-layer solutions are difficult to trace due to their transparency [7].

[7]Chapter 2 revisited the challenges and solutions through literature references

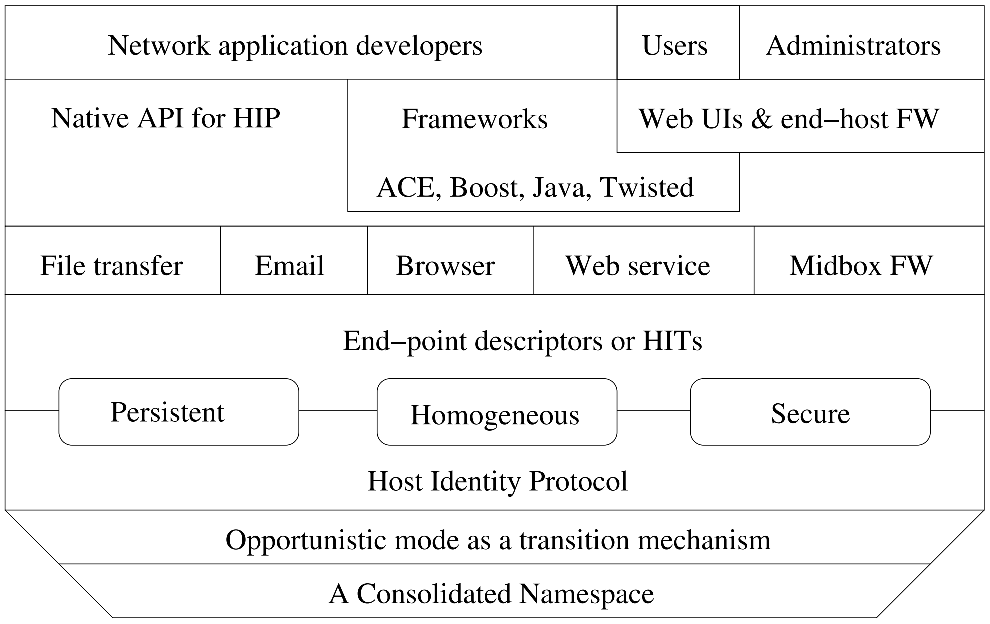


Figure 3.1. A visualization of the HIP-based solution for the challenges for a consolidated namespace

As an example of non-persistent addressing, we discovered a programming bug in the way developers implement UDP-based communications. The problem typically occurs on multihoming hosts and affects many of the UDP-based applications in Ubuntu Linux, including four network application frameworks where the problem was verified manually [8].

Regarding to heterogeneous addressing, we observed that a fourth of the applications in Ubuntu support both IPv4 and IPv6. This dual use of the addresses complicates the networking logic of applications and can also have an impact on the user experience as the related latency issues have to be solved redundantly by each developer in the absence of a de-facto solution.

As IP addresses offer little security per se, the developers typically use SSL/TLS. In Ubuntu, roughly every tenth application utilized it through the OpenSSL library. However, almost three out of five applications using the library were not initializing it in a secure way. We also observed a large number of the applications configuring various security details for the library, leaving us puzzled if some of the details should ever be exposed to the applications.

We proposed HIP as a unified solution for consolidated addressing and improved it to better fulfill this proposition. To support persistent identifiers for services, Publication III studied access control for HIP-based identifiers to mitigate against email spam with a mindset to motivate

[8]It should be noted that HIP was not integrated into the frameworks

end-hosts for long-lived identifiers. Publication IV presented a middlebox-based firewall to control access to HIP-based services for mobile HIP clients. To support heterogeneous identifiers, the firewall could again be used to homogenize IPv4 and IPv6 access control lists under a single list based on uniform HIs. The homogenization was also employed in the native API for HIP in Publication II that unified addressing for end-host applications using end-point identifiers. For secure addressing, the native API and also the graphical end-host firewall in Publication VI improved LoF-based security for HIP as later confirmed by others. It should be mentioned that the security of the identifiers played an important part in experimentation with the end host and middlebox-based firewall, and email spam.

In Publication III, we proposed a deployment model for HIP where it was used internally between SMTP servers. In the same publication, we also corrected a deployment-related misunderstanding according to which HIP records would require change to the DNS server implementations. In Publication V, we showed how HIP can actually be used in the opportunistic mode without any additional DNS records. From the view point of security, the opportunistic mode is based on the weaker LoF-based security. Nevertheless, LoF can be considered secure enough until DNSSEC fortifies the weakest link in HIP, that is, the look up of the HIP records from the DNS. However, we identified two short comings of the opportunistic mode for HIP. Firstly, the time out mechanism to fall back to non-HIP communications when encountering HIP-incapable hosts can be optimized. Secondly, the opportunistic mode regresses to non-persistent addressing in order to facilitate easier deployment.

To recap the contributions from the view point of the target groups, we tested usability of HIP in Publication VI where the end-users used a web browser to connect to a website. The presence of HIP was visualized using various cues, including the lock symbol and also using graphical prompt that operated at the system level. Despite the prototype was rather unpolished, the users understood when security was employed. The experiment also confirmed that HIP-based security should be visualized using traditional security indicators.

For developers, the native API for HIP, as presented in Publication II, allows to gain more control of HIP and supports application or user-specific identifiers. For network administrators, the middlebox-based firewall of Publication IV can simplify management issues as the persistent identifiers of HIP support mobile clients and survive network renumbering while unifying separate access control rules for IPv4 and IPv6 into single ones.

3.6 Future Directions

Publication I uncovered a number of issues in software for Linux. In general, it would be useful to automatize them using a static analysis tool such as Coverity [34]. Few ideas for improvement on the individual issues could be also mentioned. For instance, it remains uncertain how the multihoming issue with UDP-based applications relates to another discovered property of the applications. Namely, two thirds of the applications were capable handling at least some level of source address selection based on this coarse-grained estimate.

Publication II prototyped with user-specific identifiers. One reason why they were excluded in the finalized standardized version [123] was related to security. Namely, handling of user-specific identifiers requires strict access control measures on multi-user devices. Also, storing the user identity on a smart card or a TPM chip was proposed but not implemented. In practice, this would require delegation of privileges to the device from the card or chip and a revocation mechanism. For speed up the revocation, a cryptographically accelerated smart card could be used to signing a data plane based on asymmetric encryption [46] but only while the card is connected to the device. As an alternative, a TPM chip might be requested to sign a HIP-specific certificate [92] for a certain predefined time period. This way, the potential abuse of a portable long-term identifier would be limited spatially or temporally when using the identity in a compromised device, for instance, in a public Internet kiosk.

It should be also noted that parallel to our work, IPsec-policy APIs for applications [243] were developed and could be further integrated with the native API.

Publication III proposed to tackle spam by disconnecting TCP connections with misbehaving SMTP hosts and by introducing large computational puzzle values when they reconnect. In contrast, greylisting does not require the deployment of HIP and, for instance, can mitigate spam by disconnecting connections from previously unknown IP addresses. However, the assumption in greylisting is that spammers do not reconnect which may not be valid in the future as spammers get smarter. Thus, an additional mechanism as proposed by the publication may become necessary in the future.

While Publication III acted as a starting point for HIP deployment considerations in the context of SMTP, we have later also experimented with a HIP deployment internally in a cloud network [126]. Then, we have considered HIP deployment in general from from a techno-economical perspective [212]. A potential deployment direction for HIP from the techno-economical analysis was to implement HIP in a userspace library above transport layer [84] to avoid the kernel deployment hurdles required for the native HIP API.

Publication V used the opportunistic base exchange in HIP to achieve LoF for legacy applications. However, we were dissatisfied with the shim placement just above the sockets in contrast to others [87, p. 10] and explored with another implementation model where the shim layer captures datagrams at the network layer instead of socket calls [73, pp. 34-36]. While the two proof-of-concept implementations work with legacy applications at an end-host, we believe that the opportunistic mode is more suitable to in other scenarios. The shim layer capturing datagrams at the network layer could be used at middleboxes such as HIP-based proxies [192]. At end-hosts, this be used by HIP-aware applications at a client-side host to connect to a server via a rendezvous server to facilitate a HIP-layer anycast because the rendezvous server could choose the ultimate recipient with the opportunistic mode. At the server side, a server could use it to register itself one rendezvous server from a pool of rendezvous servers. Naturally, DNS round-robin schemes for rotating HI records would serve as an alternative to such a HIP-layer anycast.

In general, the number of distributed experiments with HIP has been small. For instance, the middlebox-based firewall and its experimentation in Publication IV could be extended. For production purposes, the prototype of the middle-box based firewall could be extended to support de-centralized access control to support fault tolerance and asymmetric routes. For routing, also the impact of off-the-path routing based on the optional HIP relays could also be measured. As HITs are not aggregatable, bloom filters could be used to reduce the overhead for large access control lists [42]. In addition, a performance comparison with another similar scheme such as MobIKE would be useful. MobIKE which can also be used for access control of mobile clients but requires a gateway that introduces triangular routing. In contrast, our approach does not besides the transparent firewall located at the edge of the site [9].

Publication VI prototyped a graphical user interface for HIP. The design was rather immature as the results of the usability tests indicated. Nevertheless, we believe this end-user firewall for HIP could be integrated seamlessly with existing end-user firewall products, such as offered by F-Secure, Symantec and other anti-virus companies. However, this remains to be verified.

[9]In general, firewalls are usually deployed at the edge of a site independently of whether the site supports a VPN or not

4. Conclusions

While users and services are identified using DNS-based names, such names were an extension to the TCP/IP architecture where IP addresses are omnipresent. IP addresses are used either directly or indirectly to develop network applications, and they are employed at the transport and network layers. Unfortunately, this originally simplifying choice is a source of inflexibility especially considering that computers of today are able to run sophisticated and complex software. However, it is economically challenging to change this addressing model as the IP addresses are metaphorically the glue that holds the Internet together. In this dissertation, we attempt to improve three sources of inflexibility in the TCP/IP architecture in a backward compatible way: non-persistent, heterogeneous and insecure addressing. A solution that meets the different aspects of these three challenges is referred as a consolidated namespace in the context of this work.

Non-persistent addressing stems from the nature of IP addresses that are dependent on the local topology. This causes problems for mobile and multihoming devices when they change their network attachment point. The change of the local IP address unnecessarily terminates TCP streams as they are still bound to the previous address but it is not only a problem for on-going flows of data but also for new data flow. For instance, overlapping private address realms as introduced by NAT technology are tainted by non-persistent addressing because devices cannot be uniquely identified based on their IP address alone and, thus, a mobile device may simply contact a wrong host when transition between two networks. As another example scenario, the issue manifests itself during a company merger or acquisition, or when the site changes its provider. Then, the entire network prefix of the site may change, leaving a number of stale IP addresses forgotten in a various configuration files, only to be discovered by network administrators during site renumbering.

Heterogeneous addressing as introduced by IPv6 addresses involves additional complexity for application developers and network administrators as they have to deal with two heterogeneous namespaces instead of a single. Insecure addressing is the basis for the TCP/IP model; IP addresses are insecure by their nature and additional measures are needed to guarantee security for network applications.

A number of backward compatible solutions that meet the architectural challenges set of a consolidated namespace do exist but fulfill the requirements only partially. From the surveyed solutions, five prominent solutions meet the described properties for persistent addressing and are compared in more detail for their other technical aspects. NUTSS architecture distinguishes from the others by facilitating off-path services but requires additional infrastructure to complete this task. While LIN6 solves renumbering in a relatively simple way by dividing an IPv6 address into identifier and locator portions, also it requires extra infrastructure. In contrast, ILNPv6 does not fare well with renumbering but mostly avoids deployment costs incurring from new infrastructure. BTMM is quite a complete approach but is unsuitable with multiple cascading NATs and is vendor-specific, proprietary technology. The last protocol, HIP is chosen for the empirical experimentation as it fairs well in the comparison. Similarly as BTMM, it is quite a complete solution but based on open standards and open-source implementations. As another difference, HIP is based on a unstructured identifiers which has some ramifications with the structured DNS but the impact of this so called referral issue still remains moot. As a trade off, the merits are crystal clear as HIP offers a topologically-independent namespace based on cryptographically secured, self-certifying identifiers that are compatible with legacy applications.

In the collection of articles, we have empirically experimented with HIP and improved it to better meet the criteria for a consolidated namespace as set by this dissertation. The results are disseminated from the view point of three different target groups of people: end-users, network application developers and network administrators.

First, we focused on understanding the development aspects by trying to understand the present state of networking applications. To be more precise, we analyzed how network applications and frameworks use the low-level networking APIs in Linux. Related to heterogeneous addressing, we observed that a fourth of the IPv4-based applications supported also IPv6. Related the persistent addressing, all of the four manually examined frameworks had a bug related to UDP that occurs during initial connectivity with multihoming end-hosts. We also observed that every tenth application employed SSL/TLS-based security using the OpenSSL library. Of these, almost three out of five were not initializing the library correctly from the view point of security. Hence, the challenges for consolidated addressing do exist and our work attempts to fill these gaps.

To repeat the adoption success of SSL/TLS-based security with HIP, we designed and implemented a native API for HIP that gives more control for developers of HIP-aware applications. The API also extends HIP to support application or user-specific identifiers instead of merely host specific ones. To better meet the requirements for a consolidated namespace, the API also unifies the heterogeneous two HIP identifier types used in legacy IPv4 and IPv6 applications into a single one. As later discovered by others, the separate API, such as the one we developed, is required to improve security when so called leap-of-faith security is employed [179, p. 353].

We exploited a feature of HIP, computational puzzles, to mitigate against unwanted traffic in the case of email spam. A modified spam filter throttled senders of spam by disconnecting sessions and offered more time consuming puzzles for them. The access control was based on the persistent identifiers of HIP that could be circumvented with temporary identifiers. To avoid penalties with puzzles, the proposed strategy was to reward benign hosts with long-lived identifiers with smaller puzzles. We proposed to adopt the mechanism only between email relays to avoid deployment hurdles at the client side.

As another use case, we developed a HIP-aware firewall that exploited the secure identifiers of HIP to control access to HIP-based services. The novelty of the approach is that the firewall supports mobile client devices by tracking and authenticating them based on their persistent identifiers instead of their ephemeral IP addresses. Changes in the addresses of the servers are also supported as the firewall can also support site renumbering as identifiers forgotten in various configuration files continue to work. As another relief for network administrators, a single identifier-based rule for access control replaces the two separate rules traditionally required for IPv4 and IPv6, thus supporting the goal for the heterogeneous addressing.

As an intermediate step to make the transition towards HIP easier, we explored the use of HIP as transparent mechanism that controversially does not meet goals for consolidated naming at the application layer. Our implementation of the leap-of-faith security, or opportunistic mode in other terms, was implemented as shim library between the application and the sockets layer to transparently translate IP addresses of the legacy application to HIP-based identifiers for the transport layer. As a trade off, the opportunistic mode is subject to man-in-the-middle attacks because the client learns the identity of the server during communications instead of obtaining it from pre-shared information or look up from a directory. Until DNSSEC is deployed more widely, we believe that operating HIP in this fashion offers reasonable security because DNS can be considered the weakest link for storing the identifiers for HIP. In addition, implemented approach avoids the referral issue at the application layer.

We conducted usability tests with a group test of persons to understand how end-users perceive different ways of using HIP in a web-based environment. The use of HIP was illustrated using traditional security indicators, such the lock symbol in the browser. The transparent use of HIP was illustrated using a new graphical system level prompt that essentially acted as an end-host firewall to allow the user to accept or deny HIP-based communications. The users clearly noticed when security was employed despite the difference between the LoF and normal HIP-based security was not obvious. While we observed room for polishing the prototype, others have identified that confirmation of the opportunistic mode from the user is critical from the view point of security [179, p. 352].

The research contributions of this dissertation have also real-world impact. The general investigation to the low-level networking APIs and network application frameworks revealed a number of bugs that can be fixed to improve the software quality in various Linux distributions. Some of the problems especially related to java-based framework may also affect the Linux-based Android distribution that has been dominating the mobile handset business lately. Besides the general contributions, the HIP-specific contributions have impacted the IETF standardization. The experimentation with the UDP-based multihoming, opportunistic mode, end-host and middlebox firewalls is referenced by two experimental standards [95, 94]. We improved the specification for the native API according to the feedback from the IETF community and it was published as an experimental standard [123]. The work further inspired joint activities with the SHIM6 working group and produced another API-related standard [122].

For future directions, HIP-based anycast combined with the security of native API may be an interesting research direction. To understand the deployment dimension better, we applied HIP in a cloud deployment scenario [126] and conducted a techno-economic survey [212] and, based on the findings, continued exploration of HIP as a stand-alone library at the application layer [84] during the time of this writing. Due the efforts of various researchers, HIP has become an extensively researched topic and its principles have also been adapted to other research architectures [79, 76, 26]. Besides the Tofino security product and production-environment deployment at Boeing, the time has become more mature for HIP to be embraced by the industry as it moving from the experimental to the standards track in the IETF.

Bibliography

[1] J. Abley, B. Black, and V. Gill. Goals for IPv6 Site-Multihoming Architectures. RFC 3582 (Informational), Aug. 2003.

[2] J. Abley, K. Lindqvist, E. Davies, B. Black, and V. Gill. IPv4 Multihoming Practices and Limitations. RFC 4116 (Informational), July 2005.

[3] B. Aboba, D. Simon, and P. Eronen. Extensible Authentication Protocol (EAP) Key Management Framework. RFC 5247 (Proposed Standard), Aug. 2008.

[4] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '04, pages 20–24, New York, NY, USA, 2004. ACM.

[5] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, SOSP '99, pages 186–201, New York, NY, USA, 1999. ACM.

[6] J. Ahrenholz. Host Identity Protocol Distributed Hash Table Interface. RFC 6537 (Experimental), Feb. 2012.

[7] S. Akhshabi and C. Dovrolis. The evolution of layered protocol stacks leads to an hourglass-shaped architecture. *SIGCOMM Comput. Commun. Rev.*, 41(4):206–217, Aug. 2011.

[8] H. T. Alvestrand. Overview: Real Time Protocols for Brower-based Applications, Mar. 2012. Internet draft, work in progress.

[9] A. Anand, F. Dogar, D. Han, B. Li, H. Lim, M. Machado, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste. Xia: an architecture for an evolvable and trustworthy internet. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 2:1–2:6, New York, NY, USA, 2011. ACM.

[10] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable internet protocol (aip). *SIGCOMM Comput. Commun. Rev.*, 38(4):339–350, Aug. 2008.

[11] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), Mar. 2005. Updated by RFC 6014.

[12] J. Arkko, J. Kempf, B. Zill, and P. Nikander. SEcure Neighbor Discovery (SEND). RFC 3971 (Proposed Standard), Mar. 2005. Updated by RFCs 6494, 6495.

[13] J. Arkko and A. Keranen. Experiences from an IPv6-Only Network. RFC 6586 (Informational), Apr. 2012.

[14] J. Arkko, V. Lehtovirta, and P. Eronen. Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA'). RFC 5448 (Informational), May 2009.

[15] J. Arkko and P. Nikander. *Weak Authentication: How to Authenticate Unknown Principals without Trusted Parties*, pages 5–19. Springer, 2002.

[16] R. J. Atkinson, S. Bhatti, and S. Hailes. Ilnp: mobility, multi-homing, localised addressing and security through naming. *Telecommunication Systems*, 42(3-4):273–291, 2009.

[17] F. Audet and C. Jennings. Network Address Translation (NAT) Behavioral Requirements for Unicast UDP. RFC 4787 (Best Current Practice), Jan. 2007.

[18] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972 (Proposed Standard), Mar. 2005. Updated by RFCs 4581, 4982.

[19] T. Aura, P. Nikander, and J. Leiwo. Dos-resistant authentication with client puzzles. In *Revised Papers from the 8th International Workshop on Security Protocols*, pages 170–177, London, UK, UK, 2001. Springer-Verlag.

[20] S. authors. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Amendment 6: Medium Access Control (MAC) Security Enhancements, July 2004.

[21] S. authors. Port-based Network Access Control, IEEE Std 802.1X-2010, Feb. 2010.

[22] J. Baek, J. Newmarch, R. Safavi-naini, and W. Susilo. A survey of identity-based cryptography. In *Proc. of Australian Unix Users Group Annual Conference*, pages 95–102, 2004.

[23] M. Bagnulo, P. Matthews, and I. van Beijnum. Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers. RFC 6146 (Proposed Standard), Apr. 2011.

[24] M. Bagnulo, A. Sullivan, P. Matthews, and I. van Beijnum. DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers. RFC 6147 (Proposed Standard), Apr. 2011.

[25] F. Baker, E. Lear, and R. Droms. Procedures for Renumbering an IPv6 Network without a Flag Day. RFC 4192 (Informational), Sept. 2005.

[26] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the internet. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 343–352, New York, NY, USA, 2004. ACM.

[27] H. Ballani, P. Francis, T. Cao, and J. Wang. Making routers last longer with viaggre. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, pages 453–466, Berkeley, CA, USA, 2009. USENIX Association.

[28] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, and Y. Hu. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Computer Communications*, 30(7):1655–1695, 2007.

[29] T. Bates and Y. Rekhter. Scalable Support for Multi-homed Multi-provider Connectivity. RFC 2260 (Informational), Jan. 1998.

[30] J. Beal and T. Shepard. Deamplification of DoS Attacks via Puzzles, Oct. 2004.

[31] S. Bellovin, J. Schiller, and C. Kaufman. Security Mechanisms for the Internet. RFC 3631 (Informational), Dec. 2003.

[32] M. C. Benvenuto and A. D. Keromytis. Easyvpn: Ipsec remote access made easy. In *Proceedings of the 17th USENIX conference on System administration*, pages 87–94, Berkeley, CA, USA, 2003. USENIX Association.

[33] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396 (Draft Standard), Aug. 1998. Obsoleted by RFC 3986, updated by RFC 2732.

[34] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler. A few billion lines of code later: using static analysis to find bugs in the real world. *Commun. ACM*, 53(2):66–75, Feb. 2010.

[35] B. Bishaj. Backwards Compatibility Experimentation with Host Identity Protocol and Legacy Software and Networks, June 2008.

[36] M. Blanchet and P. Seite. Multiple Interfaces and Provisioning Domains Problem Statement. RFC 6418 (Informational), Nov. 2011.

[37] M. Borella, J. Lo, D. Grabelsky, and G. Montenegro. Realm Specific IP: Framework. RFC 3102 (Experimental), Oct. 2001.

[38] C. Boulton, J. Rosenberg, G. Camarillo, and F. Audet. NAT Traversal Practices for Client-Server SIP. RFC 6314 (Informational), July 2011.

[39] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), Oct. 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633.

[40] S. Bradner, A. Mankin, and J. I. Schiller. A framework for Purpose-Built keys (PBK).

[41] M. Buddhikot, A. Hari, K. Singh, and S. Miller. Mobilenat: a new technique for mobility across heterogeneous address spaces. *Mob. Netw. Appl.*, 10(3):289–302, June 2005.

[42] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica. Rofl: routing on flat labels. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 363–374, New York, NY, USA, 2006. ACM.

[43] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. Diameter Base Protocol. RFC 3588 (Proposed Standard), Sept. 2003. Updated by RFCs 5729, 5719, 6408.

[44] T. Callahan, M. Allman, and V. Paxson. A longitudinal view of HTTP traffic. In *Proceedings of the 11th international conference on Passive and active measurement*, PAM'10, pages 222–231, Berlin, Heidelberg, 2010. Springer-Verlag.

[45] G. Camarillo, I. Mas, and P. Nikander. A framework to combine the session initiation protocol and the host identity protocol. In *WCNC*, pages 3051–3056. IEEE, 2008.

[46] G. Camarillo and J. Melen. Host Identity Protocol (HIP) Immediate Carriage and Conveyance of Upper-Layer Protocol Signaling (HICCUPS). RFC 6078 (Experimental), Jan. 2011.

[47] G. Camarillo, P. Nikander, J. Hautakorpi, A. Keranen, and A. Johnston. HIP BONE: Host Identity Protocol (HIP) Based Overlay Networking Environment (BONE). RFC 6079 (Experimental), Jan. 2011.

[48] B. Carpenter. Internet Transparency. RFC 2775 (Informational), Feb. 2000.

[49] B. Carpenter, R. Atkinson, and H. Flinck. Renumbering Still Needs Work. RFC 5887 (Informational), May 2010.

[50] B. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. RFC 3234 (Informational), Feb. 2002.

[51] G. Caruana and M. Li. A survey of emerging approaches to spam filtering. *ACM Comput. Surv.*, 44(2):9:1–9:27, Mar. 2008.

[52] I. Castineyra, N. Chiappa, and M. Steenstrup. The Nimrod Routing Architecture. RFC 1992 (Informational), Aug. 1996.

[53] H. K. Catharina Candolin, Janne Lundberg. Packet level authentication in military networks. In *Proceedings of the 6th Australian Information Warfare & IT Security Conference*, Geelong, Australia, Nov. 2005.

[54] G. Chen, K. Minami, and D. Kotz. Naming and Discovery in Mobile Systems. In P. Bellavista and A. Corradi, editors, *The Handbook of Mobile Middleware*, chapter 16, pages 387–407. 2006.

[55] D. R. Cheriton and M. Gritter. TRIAD: a Scalable Deployable NAT-based Internet Architecture, Jan. 2000.

[56] S. Cheshire, Z. Zhu, R. Wakikawa, and L. Zhang. Understanding Apple's Back to My Mac (BTMM) Service. RFC 6281 (Informational), June 2011.

[57] D. Clark, R. Braden, A. Falk, and V. Pingali. Fara: reorganizing the addressing architecture. In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, FDNA '03, pages 313–321, New York, NY, USA, 2003. ACM.

[58] D. Crocker. Telnet output formfeed disposition option. RFC 655 (Historic), Oct. 1974.

[59] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. Plutarch: an argument for network pluralism. *SIGCOMM Comput. Commun. Rev.*, 33(4):258–266, Aug. 2003.

[60] V. Devarapalli, R. Wakikawa, A. Petrescu, and P. Thubert. Network Mobility (NEMO) Basic Support Protocol. RFC 3963 (Proposed Standard), Jan. 2005.

[61] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176.

[62] A. L. Dul. Global IP Network Mobility using Border Gateway Protocol (BGP), Mar. 2006.

[63] L. Eggert and F. Gont. TCP User Timeout Option. RFC 5482 (Proposed Standard), Mar. 2009.

[64] T. Ernst. Network Mobility Support Goals and Requirements. RFC 4886 (Informational), July 2007.

[65] T. Ernst and H.-Y. Lach. Network Mobility Support Terminology. RFC 4885 (Informational), July 2007.

[66] P. Eronen. IKEv2 Mobility and Multihoming Protocol (MOBIKE). RFC 4555 (Proposed Standard), June 2006.

[67] P. Eronen, H. Tschofenig, and Y. Sheffer. An Extension for EAP-Only Authentication in IKEv2. RFC 5998 (Proposed Standard), Sept. 2010.

[68] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.

[69] P. Faltstrom and G. Huston. A Survey of Internet Identities, Dec. 2004. An expired Internet draft.

[70] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID Separation Protocol (LISP), Feb. 2012. Work in progress.

[71] D. Farinacci, D. Lewis, D. Meyer, and C. White. LISP Mobile Node, 2011 Oct. Work in progress.

[72] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[73] T. Finez. Efficient Leap of Faith Security with Host Identity Protocol, Dec. 2008.

[74] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC 6182 (Informational), Mar. 2011.

[75] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 13–13, Berkeley, CA, USA, 2005. USENIX Association.

[76] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris. Persistent personal names for globally connected mobile devices. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI '06)*, Seattle, Washington, November 2006.

[77] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris. User-relative names for globally connected personal devices. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS06)*, Santa Barbara, CA, February 2006.

[78] U. Forum. *"Internet Gateway Device (IGD) V 2.0"*, Apr. 2012.

[79] P. Francis and R. Gummadi. Ipnl: A nat-extended internet architecture. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 69–80, New York, NY, USA, 2001. ACM.

[80] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), Aug. 2011.

[81] V. Fuller, D. Farinacci, D. Meyer, and D. Lewis. LISP Alternative Topology (LISP+ALT), 2011 Dec. Work in progress.

[82] M. Goff. *Network Distributed Computing: Fitscapes and Fallacies*. Prentice Hall Professional Technical Reference, 2003.

[83] J. T. Goodman and R. Rounthwaite. Stopping outgoing spam. In *Proceedings of the 5th ACM conference on Electronic commerce*, EC '04, pages 30–39, New York, NY, USA, 2004. ACM.

[84] X. Gu. Host Identity Protocol Version 2.5, June 2012.

[85] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh. NAT Behavioral Requirements for TCP. RFC 5382 (Best Current Practice), Oct. 2008.

[86] S. Guha and P. Francis. Characterization and measurement of tcp traversal through nats and firewalls. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, IMC '05, pages 18–18, Berkeley, CA, USA, 2005. USENIX Association.

[87] S. Guha and P. Francis. An end-middle-end approach to connection establishment. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 193–204, New York, NY, USA, 2007. ACM.

[88] A. Gurtov and T. Polishchuk. Secure multipath transport for legacy internet applications. In *Broadband Communications, Networks, and Systems, 2009. BROADNETS 2009. Sixth International Conference on*, pages 1 –8, sept. 2009.

[89] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608 (Proposed Standard), June 1999. Updated by RFC 3224.

[90] E. Hammer-Lahav. The OAuth 1.0 Protocol. RFC 5849 (Informational), Apr. 2010.

[91] T. Heer. Direct End-to-Middle Authentication in Cooperative Networks, Dec. 2011.

[92] T. Heer and S. Varjonen. Host Identity Protocol Certificates. RFC 6253 (Experimental), May 2011.

[93] S. Heikkinen. Applicability of Host Identities in Securing Network Attachment and Ensuring Service Accountability, Nov. 2011.

[94] T. Henderson and A. Gurtov. The Host Identity Protocol (HIP) Experiment Report. RFC 6538 (Informational), Mar. 2012.

[95] T. Henderson, P. Nikander, and M. Komu. Using the Host Identity Protocol with Legacy Applications. RFC 5338 (Experimental), Sept. 2008.

[96] T. Henderson, S. C. Venema, and D. Mattes. HIP-based Virtual Private LAN Service (HIPLS), Mar. 2012.

[97] T. R. Henderson. Host mobility for IP networks: A comparison. *IEEE Network*, 17(6):18–26, Nov. 2003.

[98] S. Herborn, A. Huber, R. Boreli, and A. Seneviratne. Secure host identity delegation for mobility. In *COMSWARE*. IEEE, 2007.

[99] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), Feb. 2006. Updated by RFCs 5952, 6052.

[100] R. Hinden and B. Haberman. Unique Local IPv6 Unicast Addresses. RFC 4193 (Proposed Standard), Oct. 2005.

[101] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), Feb. 2006. Updated by RFCs 5991, 6081.

[102] C. Huitema and B. Carpenter. Deprecating Site Local Addresses. RFC 3879 (Proposed Standard), Sept. 2004.

[103] G. Huston. Architectural Approaches to Multi-homing for IPv6. RFC 4177 (Informational), Sept. 2005.

[104] G. Huston. A Rough Guide to Address Exhaustion, Mar. 2011.

[105] G. Iapichino and C. Bonnet. Host identity protocol and proxy mobile ipv6: a secure global and localized mobility management scheme for multihomed mobile nodes. In *Proceedings of the 28th IEEE conference on Global telecommunications*, GLOBECOM'09, pages 578–583, Piscataway, NJ, USA, 2009. IEEE Press.

[106] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.

[107] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. *SIGCOMM Comput. Commun. Rev.*, 34(4):145–158, Aug. 2004.

[108] D. Jen, M. Meisel, H. Yan, D. Massey, L. Wang, B. Zhang, and L. Zhang. Towards a new internet routing architecture: Arguments for separating edges from transit core. *HotNets-VII*, October 2008.

[109] C. Jennings, B. B. Lowekamp, E. Rescorla, S. A. Baset, and H. Schulzrinne. *REsource LOcation And Discovery (RELOAD) Base Protocol*. Internet Engineering Task Force, Oct. 2011. Internet draft, work in progress.

[110] C. Jennings, B. B. Lowekamp, E. Rescorla, S. A. Baset, and H. Schulzrinne. REsource LOcation And Discovery (RELOAD) Base Protocol, Mar. 2012. Internet draft, work in progress.

[111] P. Jokela, R. Moskowitz, and J. Melen. Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP), July 2012. Internet draft, work in progress.

[112] P. Jokela, R. Moskowitz, and P. Nikander. Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP). RFC 5202 (Experimental), Apr. 2008.

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

[114] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. Lipsin: line speed publish/subscribe inter-networking. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 195–206, New York, NY, USA, 2009. ACM.

[115] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: an empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 3–14, New York, NY, USA, 2008. ACM.

[116] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), Dec. 2005. Obsoleted by RFC 5996, updated by RFC 5282.

[117] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005. Updated by RFC 6040.

[118] A. Keränen, G. Camarillo, and J. Mäenpää. *Host Identity Protocol-Based Overlay Networking Environment (HIP BONE) Instance Specification for REsource LOcation And Discovery (RELOAD)*. Internet Engineering Task Force, Apr. 2011. Internet draft, work in progress.

[119] V. Khare, D. Jen, X. Zhao, Y. Liu, D. Massey, L. Wang, B. Zhang, and L. Zhang. Evolution towards global routing scalability. *IEEE Journal on Selected Areas in Communications*, 28(8):1363–1375, 2010.

[120] A. Khurri. Evaluating IP Security on Lightweight Hardware, Jan. 2011. ISBN 978-952-60-4004-2.

[121] S. L. Kinney. *Trusted Platform Module Basics: Using TPM in Embedded Systems (Embedded Technology)*. Newnes, 2006.

[122] M. Komu, M. Bagnulo, K. Slavov, and S. Sugimoto. Sockets Application Program Interface (API) for Multihoming Shim. RFC 6316 (Informational), July 2011.

[123] M. Komu and T. Henderson. Basic Socket Interface Extensions for the Host Identity Protocol (HIP). RFC 6317 (Experimental), July 2011.

[124] M. Komu, T. Henderson, H. Tschofenig, J. Melen, and A. Keranen. Basic Host Identity Protocol (HIP) Extensions for Traversal of Network Address Translators. RFC 5770 (Experimental), Apr. 2010.

[125] M. Komu, M. Sethi, R. Mallavarapu, H. Oirola, R. Khan, and S. Tarkoma. Secure Networking for Virtual Machines in the Cloud. In *International Workshop on Power and QoS Aware Computing (PQoSCom2012)*. IEEE, sep 2012. Accepted for publication.

[126] M. Komu, M. Sethi, R. Mallavarapu, H. Oirola, R. Khan, and S. Tarkoma. Secure networking for virtual machines in the cloud. In *The 2012 International Workshop on Power and QoS Aware Computing (PQoSCom'12), held in conjunction with IEEE Cluster'12*. IEEE Computer Society, Sept. 2012. Accepted for publication.

[127] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, Aug. 2007.

[128] T. Koponen, P. Eronen, and M. Särelä. Resilient connections for ssh and tls. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 30–30, Berkeley, CA, USA, 2006. USENIX Association.

[129] T. Koponen, J. Lindqvist, N. Karlsson, E. Vehmersalo, M. Komu, M. Kousa, D. Korzun, and A. Gurtov. Overview and Comparison Criteria for the Host Identity Protocol and Related Technologies, Nov. 2005.

[130] J. Korhonen. IP Mobility in Wireless Operator Networks, Nov. 2008. 978-952-10-5014-5.

[131] J. Koskela and S. Tarkoma. Simple peer-to-peer sip privacy. In *Security and Privacy in Mobile Information and Communication Systems*, volume 17 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 226–237. Springer Berlin Heidelberg, 2009.

[132] D. V. Krioukov and K. C. Claffy. Toward compact interdomain routing. *CoRR*, abs/cs/0508021, 2005.

[133] M. Kucherawy and D. Crocker. Email Greylisting: An Applicability Statement for SMTP. RFC 6647 (Proposed Standard), June 2012.

[134] M. Kulkarni, A. Patel, and K. Leung. Mobile IPv4 Dynamic Home Agent (HA) Assignment. RFC 4433 (Proposed Standard), Mar. 2006.

[135] M. Kunishi, M. Ishiyama, K. Uehara, H. Esaki, and F. Teraoka. LIN6: A New Approach to Mobility Support in IPv6. In *in Proc. of the Third International Symposium on Wireless Personal Multimedia Communications*, nov 2000.

[136] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational), Aug. 2007.

[137] J. Laganier and L. Eggert. Host Identity Protocol (HIP) Rendezvous Extension. RFC 5204 (Experimental), Apr. 2008.

[138] J. Laganier, T. Koponen, and L. Eggert. Host Identity Protocol (HIP) Registration Extension. RFC 5203 (Experimental), Apr. 2008.

[139] D. Lagutin. Securing the internet with digital signatures, Dec. 2010.

[140] D. Lagutin and H. Kari. Controlling incoming connections using certificates and distributed hash tables. In Y. Koucheryavy, J. Harju, and A. Sayenko, editors, *NEW2AN*, volume 4712 of *Lecture Notes in Computer Science*, pages 455–467. Springer, 2007.

[141] A. Langley. Transport Layer Security (TLS) Snap Start, June 2010. Expired Internet draft.

[142] A. Langley, N. Modadugu, and B. Moeller. Transport Layer Security (TLS) False Start, June 2010. Expired Internet draft.

[143] E. Lear. NERD: A Not-so-novel EID to RLOC Database, Apr. 2012. Work in progress, expires in October 2012.

[144] E. Lear and R. Droms. What's in a name: Thoughts from the NSRG. Internet-draft, IETF Secretariat, Fremont, CA, USA, Sept. 2003.

[145] T. Li. Design Goals for Scalable Internet Routing. RFC 6227 (Informational), May 2011.

[146] T. Li. Recommendation for a Routing Architecture. RFC 6115 (Informational), Feb. 2011.

[147] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard), Apr. 2010.

[148] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant Characteristics of Residential Broadband Internet Traffic. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 90–102, New York, NY, USA, 2009. ACM.

[149] J. Manner and M. Kojo. Mobility Related Terminology. RFC 3753 (Informational), June 2004.

[150] A. Medina, M. Allman, and S. Floyd. Measuring interactions between transport protocols and middleboxes. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pages 336–341, New York, NY, USA, 2004. ACM.

[151] J. Melen, J. Ylitalo, and P. Salmela. Host Identity Protocol-based Mobile Proxy, Aug. 2009. An expired Internet draft.

[152] C. Metz and J. ichiro itojun Hagino. IPv4-Mapped Addresses on the Wire Considered Harmful, Oct. 2003. Work in progress, expired in Oct, 2003.

[153] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984 (Informational), Sept. 2007.

[154] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, Apr. 2004.

[155] A. Mishra, M. Shin, and W. Arbaugh. An empirical analysis of the ieee 802.11 mac layer handoff process. *SIGCOMM Comput. Commun. Rev.*, 33(2):93–102, Apr. 2003.

[156] G. Montenegro. Reverse Tunneling for Mobile IP, revised. RFC 3024 (Proposed Standard), Jan. 2001.

[157] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.

[158] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host Identity Protocol. RFC 5201 (Experimental), Apr. 2008. Updated by RFC 6253.

[159] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021, 5896, 6111, 6112, 6113, 6649.

[160] C. Ng, P. Thubert, M. Watari, and F. Zhao. Network Mobility Route Optimization Problem Statement. RFC 4888 (Informational), July 2007.

[161] P. Nie, J. Vähä-Herttua, T. Aura, and A. Gurtov. Performance analysis of hip diet exchange for wsn security establishment. In *Proceedings of the 7th ACM symposium on QoS and security for wireless and mobile networks*, Q2SWinet '11, pages 51–56, New York, NY, USA, 2011. ACM.

[162] P. Nikander, T. Henderson, C. Vogt, and J. Arkko. End-Host Mobility and Multihoming with the Host Identity Protocol. RFC 5206 (Experimental), Apr. 2008.

[163] P. Nikander and J. Laganier. Host Identity Protocol (HIP) Domain Name System (DNS) Extensions. RFC 5205 (Experimental), Apr. 2008.

[164] P. Nikander, J. Laganier, and F. Dupont. An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID). RFC 4843 (Experimental), Apr. 2007.

[165] P. Nikander and K. Slavov. Proxying Approach to SHIM6 and HIP (PASH), Feb. 2007. An expired Internet draft.

[166] 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*, pages 87–99, San Diego, California, Feb. 2003. Internet Society. `http://www.tcm.hut.fi/~pnr/publications/NDSS03-Nikander-et-al.pdf`.

[167] P. Nikander, J. Ylitalo, and J. Wall. Integrating security, mobility, and multi-homing in a hip way. In *Network and Distributed Systems Security Symposium (NDSS'03)*, pages 87–99, Feb. 2003.

[168] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533 (Proposed Standard), June 2009.

[169] M. O'Dell. GSE - An Alternate Addressing Architecture for IPv6, Feb. 1997.

[170] L. Ong and J. Yoakum. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational), May 2002.

[171] R. H. Paine. *Beyond HIP: The End to Hacking As We Know It*. BookSurge Publishing, 2009.

[172] J. Pan, S. Paul, and R. Jain. A survey of the research on future internet architectures. *Communications Magazine, IEEE*, 49(7):26 –36, july 2011.

[173] S. Paul, J. Pan, and R. Jain. A survey of naming systems: Classification and analysis of the current schemes using a new naming reference model, 2009.

[174] S. Paul, J. Pan, and R. Jain. Architectures for the future networks and the next generation internet: A survey. *Comput. Commun.*, 34(1):2–42, Jan. 2011.

[175] X. Pérez-Costa, M. Torrent-Moreno, and H. Hartenstein. A performance comparison of mobile ipv6, hierarchical mobile ipv6, fast handovers for mobile ipv6 and their combination. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(4):5–19, Oct. 2003.

[176] C. Perkins. IP Mobility Support for IPv4, Revised. RFC 5944 (Proposed Standard), Nov. 2010.

[177] C. Perkins, D. Johnson, and J. Arkko. Mobility Support in IPv6. RFC 6275 (Proposed Standard), July 2011.

[178] H. Petander. A Network Mobility Management Architecture for a Heterogeneous Network Environment, Dec. 2007. ISBN 978-951-22-9098-7.

[179] V. Pham and T. Aura. Security Analysis of Leap-of-Faith Protocols. In *Seventh ICST International Conference on Security and Privacy for Communication Networks*, Sept. 2011.

[180] S. Pierrel, P. Jokela, J. Melen, and K. Slavov. *A Policy System for Simultaneous Multiaccess with Host Identity Protocol*. Munich, Germany, May 2007.

[181] V. K. Pingali, A. Falk, T. Faber, and R. Braden. Farads prototype design document, june 2003.

[182] O. Ponomarev and A. Gurtov. Embedding Host Identity Tags Data in DNS, 2009. An expired Internet draft.

[183] O. Ponomarev, A. Khurri, and A. Gurtov. Elliptic curve cryptography (ecc) for host identity protocol (hip). In *Proceedings of the 2010 Ninth International Conference on Networks*, ICN '10, pages 215–219, Washington, DC, USA, 2010. IEEE Computer Society.

[184] L. Popa, A. Ghodsi, and I. Stoica. Http as the narrow waist of the future internet. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 6:1–6:6, New York, NY, USA, 2010. ACM.

[185] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), Oct. 1985. Updated by RFCs 2228, 2640, 2773, 3659, 5797.

[186] R. Raghavendra, E. M. Belding, K. Papagiannaki, and K. C. Almeroth. Understanding handoffs in large ieee 802.11 wireless networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 333–338, New York, NY, USA, 2007. ACM.

[187] E. Rescorla and N. Modadugu. Datagram Transport Layer Security. RFC 4347 (Proposed Standard), Apr. 2006. Obsoleted by RFC 6347, updated by RFC 5746.

[188] J. Rexford and C. Dovrolis. Future internet architecture: clean-slate versus evolutionary research. *Commun. ACM*, 53(9):36–40, Sept. 2010.

[189] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2865 (Draft Standard), June 2000. Updated by RFCs 2868, 3575, 5080.

[190] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Proposed Standard), Apr. 2010. Updated by RFC 6336.

[191] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), Oct. 2008.

[192] P. Salmela and J. Melén. Host identity protocol proxy. In J. Filipe and L. Vasiu, editors, *ICETE*, pages 222–230. INSTICC Press, 2005.

[193] J. Saltzer. Naming and Binding of Objects. In *Operating Systems, Lecture notes in Computer Science, Vol. 60*. Springer-Verlag, 1978.

[194] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, Nov. 1984.

[195] M. Scharf and A. Ford. Mptcp application interface considerations, Oct. 2012. Work in progress.

[196] J. Schlyter and W. Griffin. Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints. RFC 4255 (Proposed Standard), Jan. 2006.

[197] S. Schütz, L. Eggert, S. Schmid, and M. Brunner. Protocol enhancements for intermittently connected hosts. *SIGCOMM Comput. Commun. Rev.*, 35(3):5–18, July 2005.

[198] R. Seggelmann, M. Tüxen, and E. P. Rathgeb. Dtls mobility. In *Proceedings of the 13th international conference on Distributed Computing and Networking*, ICDCN'12, pages 443–457, Berlin, Heidelberg, 2012. Springer-Verlag.

[199] R. Shacham, H. Schulzrinne, S. Thakolsri, and W. Kellerer. Session Initiation Protocol (SIP) Session Mobility. RFC 5631 (Informational), Oct. 2009.

[200] Z. Shelby and C. Bormann. *6LoWPAN: The Wireless Embedded Internet*. Wiley Publishing, 2010.

[201] C. Shields and J. J. Garcia-Luna-Aceves. The hip protocol for hierarchical multicast routing. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, PODC '98, pages 257–266, New York, NY, USA, 1998. ACM.

[202] J. Shoch. Inter-Network Naming, Addressing, and Routing. In *IEEE Proc. COMPCON*, pages 72–79. IEEE, 1978.

[203] A. C. Snoeren, H. Balakrishnan, and M. F. Kaashoek. Reconsidering internet mobility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, HOTOS '01, pages 41–, Washington, DC, USA, 2001. IEEE Computer Society.

[204] R. Sofia, P. Nesser, and II. Survey of IPv4 Addresses in Currently Deployed IETF Application Area Standards Track and Experimental Documents. RFC 3795 (Informational), June 2004.

[205] H. Soliman. Mobile IPv6 Support for Dual Stack Hosts and Routers. RFC 5555 (Proposed Standard), June 2009.

[206] N. specified. OpenID Authentication 2.0 - Final, Dec. 2007.

[207] P. Srisuresh, B. Ford, and D. Kegel. State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs). RFC 5128 (Informational), Mar. 2008.

[208] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), Sept. 2007. Updated by RFCs 6096, 6335.

[209] R. Stewart, M. Tuexen, K. Poon, P. Lei, and V. Yasevich. Sockets API Extensions for the Stream Control Transmission Protocol (SCTP). RFC 6458 (Informational), Dec. 2011.

[210] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, and M. Kozuka. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. RFC 5061 (Proposed Standard), Sept. 2007.

[211] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. *IEEE/ACM Trans. Netw.*, 12(2):205–218, Apr. 2004.

[212] A. K. Tapio Levä, Miika Komu and S. Luukkainen. Adoption Barriers of Network-layer Protocols: the Case of Host Identity Protocol. In *The International Journal of Computer and Telecommunications Networking*. Elsevier, Oct. 2012. Unpublished manuscript, submitted to Elsevier COMNET journal.

[213] F. Templin. The Internet Routing Overlay Network (IRON). RFC 6179 (Experimental), Mar. 2011.

[214] D. Thaler. Evolution of the IP Model. RFC 6250 (Informational), May 2011.

[215] D. Thaler and B. Aboba. What Makes For a Successful Protocol? RFC 5218 (Informational), July 2008.

[216] N. D. Tom Scavo. Shibboleth Architecture, Technical Overview, 2005 June.

[217] J. Touch, D. Black, and Y. Wang. Problem and Applicability Statement for Better-Than-Nothing Security (BTNS). RFC 5387 (Informational), Nov. 2008.

[218] J. Touch and R. Perlman. Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement. RFC 5556 (Informational), May 2009.

[219] S. Tritilanunt, C. Boyd, E. Foo, and J. M. G. Nieto. Examining the dos resistance of hip. In *OTM Workshops (1)*, volume 4277 of *Lecture Notes in Computer Science*, pages 616–625. Springer, 2006.

[220] H. Tschofenig, M. Shanmugam, and F. Muenz. *Using SRTP transport format with HIP*. IETF, Aug. 2006. expired Internet draft.

[221] G. Tsirtsis, V. Park, and H. Soliman. Dual-Stack Mobile IPv4. RFC 5454 (Proposed Standard), Mar. 2009.

[222] Z. Turányi, A. Valkó, and A. T. Campbell. 4+4: an architecture for evolving the internet address space back toward transparency. *SIGCOMM Comput. Commun. Rev.*, 33(5):43–54, Oct. 2003.

[223] S. Turner and L. Chen. Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms. RFC 6151 (Informational), Mar. 2011.

[224] J. Ubillos, M. Xu, Z. Ming, and C. Vogt. Name-Based Sockets Architecture, Sept. 2010. Experimental and expired Internet draft, work in progress.

[225] P. Urien, D. Nyami, S. Elrharbi, H. Chabanne, T. Icart, C. Pépin, M. Bouet, D. D. O. Cunha, V. Guyot, G. Pujolle, E. Gressier-Soudan, and J.-F. Susini. Hip tags privacy architecture. In *Proceedings of the 2008 Third International Conference on Systems and Networks Communications*, ICSNC '08, pages 179–184, Washington, DC, USA, 2008. IEEE Computer Society.

[226] S. Varjonen. Secure Connectivity With Persistent Identities, Mar. 2012.

[227] S. Varjonen, T. Heer, K. Rimey, and A. Gurtov. Secure resolution of End-Host identifiers for mobile clients. In *IEEE GLOBECOM 2011 - Next Generation Networking Symposium (GC'11 - NGN), Awarded the NGN Best Paper Award*, Piscataway, NJ, USA, 12 2011. IEEE.

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

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

[230] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC 2136 (Proposed Standard), Apr. 1997. Updated by RFCs 3007, 4035, 4033, 4034.

[231] C. Vogt. Six/one router: a scalable and backwards compatible solution for provider-independent addressing. In *Proceedings of the 3rd international workshop on Mobility in the evolving internet architecture*, MobiArch '08, pages 13–18, New York, NY, USA, 2008. ACM.

[232] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, Berkeley, CA, USA, 2004. USENIX Association.

[233] M. Wasserman and P. Seite. Current Practices for Multiple-Interface Hosts. RFC 6419 (Informational), Nov. 2011.

[234] B. Wellington. Secure Domain Name System (DNS) Dynamic Update. RFC 3007 (Proposed Standard), Nov. 2000.

[235] R. Whittle. Ivip (Internet Vastly Improved Plumbing) Architecture, 2010 Mar.

[236] N. Williams. IPsec Channels: Connection Latching. RFC 5660 (Proposed Standard), Oct. 2009.

[237] N. Williams and M. Richardson. Better-Than-Nothing Security: An Unauthenticated Mode of IPsec. RFC 5386 (Proposed Standard), Nov. 2008.

[238] D. Wing, S. Cheshire, M. Boucadair, R. Penno, and P. Selkirk. Port Control Protocol, Mar. 2012.

[239] D. Wing, P. Patil, and T. Reddy. Mobility with ICE (MICE), July 2012. Internet draft, work in progress.

[240] K. Winstein and H. Balakrishnan. Mosh: An Interactive Remote Shell for Mobile Clients. In *USENIX Annual Technical Conference*, Boston, MA, June 2012.

[241] J. Wu, J. Bi, X. Li, G. Ren, K. Xu, and M. Williams. A Source Address Validation Architecture (SAVA) Testbed and Deployment Experience. RFC 5210 (Experimental), June 2008.

[242] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Comput. Netw.*, 52(12):2292–2330, Aug. 2008.

[243] H. Yin and H. Wang. Building an application-aware ipsec policy system. In *Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*, pages 21–21, Berkeley, CA, USA, 2005. USENIX Association.

[244] J. Ylitalo. Secure Mobility at Multiple Granularity Levels over Heterogeneous Datacom Networks, Nov. 2008. ISBN 978-951-22-9530-2.

[245] J. Ylitalo and P. Nikander. "A new Name Space for End-Points: Implementing secure Mobility and Multi-homing across the two versions of IP". In *in Proc. of the Fifth European Wireless Conference*, Mobile and Wireless Systems beyond 3G, pages pp. 435–441. SCI UPC (Eds: Olga Casals, Jorge Carcia-Vidal, Jose M Barcelo, and Llorenc Cerda), Feb. 2004.

[246] V. C. Zandy and B. P. Miller. Reliable network connections. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, MobiCom '02, pages 95–106, New York, NY, USA, 2002. ACM.

[247] D. Zhang, X. Xu, J. Yao, and Z. Cao. Investigation in HIP Proxies, Oct. 2011. Work in progress, Internet draft.

[248] L. Zhang. An Overview of Multihoming and Open Issues in GSE, Sept. 2006.

[249] Z. Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges. *Communications Surveys & Tutorials, IEEE*, 8(1):24–37, Mar. 2006.

[250] X. Zhu, Z. Ding, and X. Wang. A multicast routing algorithm applied to hip-multicast model. In *Proceedings of the 2011 International Conference on Network Computing and Information Security - Volume 01*, NCIS '11, pages 169–174, Washington, DC, USA, 2011. IEEE Computer Society.

Publication I

Miika Komu, Samu Varjonen, Sasu Tarkoma and Andrei Gurtov. Sockets and Beyond: Assessing the Source Code of Network Applications. *Linux Symposium*, Proceedings of Ottawa Linux Symposium, Technical Paper, Ottawa, Canada, July 2012.

Sockets and Beyond: Assessing the Source Code of Network Applications

Miika Komu
Aalto University, Department of Computer Science and Engineering
miika@iki.fi

Samu Varjonen, Andrei Gurtov, Sasu Tarkoma
University of Helsinki and Helsinki Institute for Information Technology
firstname.lastname@hiit.fi

May 6, 2012

Abstract

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

1 Introduction

The Sockets API is the basis for all internet applications. While the number of applications using it directly is large, some applications use it indirectly through intermediate libraries or frameworks to hide the intricacies of the low-level Sockets API. Nevertheless, it is then the intermediaries that still have to interface with the Sockets API. Thus, the Sockets API is important for all network applications either directly or indirectly but has been studied little. To fill in this gap, we have statistically analyzed the usage of Sockets API to characterize how contemporary network applications behave in Ubuntu Linux. In addition to merely characterize the trends, we have also investigated certain programming pitfalls pertaining the Sockets API.

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

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

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

To be more precise, the problem was related to multi-homing with UDP.

In this article, we describe how to solve some of the discovered issues in applications and frameworks using the Sockets API. We also characterize some of the inherent limitations of the Sockets API, for instance, related to complexity.

2 Background

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

2.1 The Sockets API

The Sockets API is the de facto API for network programming due to its availability for various operating systems and languages. As the API is rather low level and does support object-oriented languages well, many networking libraries and frameworks offer additional higher-level abstractions to hide the details of the Sockets API.

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

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

2.1.1 Name Resolution

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

2.1.2 Delivery of Application Data

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

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

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

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

For datagram-oriented networking with UDP, the *sendto()* and the *recvfrom()* functions can be used. Complementary functions *sendmsg()* and *recvmsg()* offer more advanced interfaces for applications [19]. They operate on scatter arrays (multiple non-consecutive I/O buffers instead of just one) and support also so called ancillary data that refers to meta-data and information related to network packet headers.

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

2.1.3 Customizing Networking Stack

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

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

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

2.2 Sockets API Extensions

Basic Socket Interface Extensions for IPv6 [5] define additional data structures and constants, including *AF_INET* and *sockaddr_in6*. The extensions also define new DNS resolver functions, *getnameinfo()* and *getaddrinfo()*, as the old ones, *gethostbyname()* and *gethostbyaddr()*, are now obsoleted. The older ones are not thread safe and offer too little control over the resolved addresses. The specification also defines IPv6-mapped IPv4 addresses to improve IPv6 interoperability.

An IPv6 application can typically face a choice of multiple source and destination IPv6 pairs to choose from. Picking a pair may not be a simple task because some of the pairs may not even result in a working connectivity. IPv6 Socket API for Source Address Selection [13] defines extensions that restrict the local or remote address to a certain type, for instance, public or temporary IPv6 addresses. The extensions include new socket options to restrict the selection local addresses when, e.g., a client application connects without specifying the source address. For remote address selection, new flags for the *getaddrinfo()* resolver are proposed. The extensions mainly affect client-side connectivity but can affect also at the server side when UDP is being used.

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

Multihoming is becoming interesting because most of the modern handhelds are equipped with, e.g., 3G and WLAN interfaces. In the scope of this work, we associate "multihoming" to hosts with multiple IP addresses typically introduced by multiple network interfaces. Multihoming could be further be further characterized whether it occurs in the initial phases of the connectivity or during established communications. All of the statistics in this article refer to the former case because the latter requires typically some extra logic in the application or additional support from the lower layers.

When written correctly, UDP-based applications can support multihoming for initial connectivity and the success of this capability is investigated in detail in this article. However, supporting multihoming in TCP-based applications is more difficult to achieve and requires additional extensions. A solution at the application layer is to recreate connections when they are rendered broken. At the transport layer, Multipath TCP [4] is a TCP-specific solution to support multihoming in a way that is compatible with legacy applications with optional APIs for native applications [16].

The Stream Control Transmission Protocol (SCTP, [21]) implements an entirely new transport protocol with full multihoming capabilities. In a nutshell, SCTP offers a reliable, congestion-aware, message-oriented, in-sequence transport protocol. The minimum requirement to enable SCTP in an existing application is to change the protocol type in *socket()* call to SCTP. However, the application can only fully harness the benefits of the protocol by utilizing the *sendmsg()* and *recvmsg()* interface. Also, the protocol supports sharing of a single socket descriptor for multiple simultaneous communication partners; this requires some additional logic in the application.

Transport-independent solutions operating at the lower layers include Host Identity Protocol [11] and Site Multihoming by IPv6 Intermediation (SHIM6) [12]. In brief, HIP offers support for end-host mobility, multihoming and NAT traversal. By contrast, SHIM6 is mainly a multihoming solution. From the API perspective, SHIM6 offers backwards compatible identifiers for IPv6 - in the sense that they are routable at the network layer - where as the identifiers in HIP are non-routable. HIP has its own optional APIs for HIP-aware applications [9] but both protocols share the same optional multihoming APIs [8].

Name-based Sockets are a work-in-progress at the IETF standardization forum. While the details of the specification [23] are rather immature and the specification still lacks official consent of the IETF, the main idea is to provide extensions to the Sockets API that replace IP addresses with DNS-based names. In this way, the responsibility for the management of IP addresses is pushed down in the stack, away from the application layer.

2.3 NAT Traversal

Private address realms [18] were essentially introduced by NATs but also Virtual Private Networks (VPNs) and other tunneling solutions can also make use of private addresses. Originally, the concept of virtual address spaces was created to alleviate the depletion of the IPv4 address space, perhaps, because it appeared that most client hosts did not need publicly-reachable addresses. Consequently, NATs also offer some security as a side effect to the client side because they discard new incoming data flows by default.

To work around NATs, Teredo [7] offers NAT traversal solution based on a transparent tunnel to the applications. The protocol tries to penetrate through NAT boxes to establish a direct end-to-end tunnel but can resort to triangular routing through a proxy in the case of an unsuccessful penetration.

2.4 Transport Layer Security

Transport Layer Security (TLS) [22] is a cryptographic protocol that can be used to protect communications above the transport layer. TLS, and its predecessor Secure Socket Layer (SSL), are the most common way to protect TCP-based communications over the Internet.

In order to use SSL or TLS, a C/C++ application is usually linked to a library implementation such as OpenSSL or GNU TLS. The application then calls the APIs of the TLS/SSL-library instead of using the APIs of the Sockets API. The functions of the library are wrappers around the Sockets API, and are responsible for securing the data inside the TCP stream.

2.5 Network Frameworks

The Sockets API could be characterized as somewhat complicated and error-prone to be programmed directly. It is also “flat” by its nature because it was not designed to accommodate object-oriented languages. For these reasons, a number of libraries and frameworks have been built to hide the details of the Sockets API and to introduce object-oriented interfaces. The Adaptive Communication (ACE) [17] is one such framework.

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

Boost::Asio is another open source C++ library that offers high-level networking APIs to simplify development of networking applications. Boost::Asio aims to be portable, scalable, and efficient but, most of all, it provides a starting point for implementing further abstraction. Several Boost C++ libraries have already been included in the C++ Technical Report 1 and in C++11.

In 2006 a networking proposal based on Asio was submitted to request inclusion in the upcoming Technical Report 2.

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

Twisted is a modular, high-level networking framework for python. Similarly as ACE, also Twisted is based on inversion of control and asynchronous messaging. Twisted has built-in support for multiple application-layer protocols, including IRC, SSH and HTTP. What distinguishes Twisted from the other frameworks is the focus on service-level functionality based adaptable functionality that can be run on top of several application-layer protocols.

3 Materials and Methods

We collected information related to the use of Sockets API usage in open-source applications. In this article, we refer to this information as *indicators*. An indicator refers to a constant, structure or function of the C language. We analyzed the source code for indicators in a static way (based on keywords) rather than dynamically [1]. The collected set of indicators was limited to networking-related keywords obtained from the keyword indexes of two books [20, 15].

We gathered the material for our analysis from all of the released Long-Term Support (LTS) releases of Ubuntu: Dapper Drake 6.06, Hardy Heron 8.04, Lucid Lynx 10.04. Table 1 summarizes the number of software packages gathered per release. In the table, "patched" row expresses how many applications were patched by Ubuntu.

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

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

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

Table 1: Number of packages per release version.

We did not limit or favor the set of applications, e.g., based on any popularity metrics. We believed that an application was of at least of some interest if the application was being maintained by someone in Ubuntu. To be more useful for the community, we analyzed all network applications and did not discriminate some "unpopular" minorities. This way, we did not have to choose between different definitions of popularity – perhaps Ubuntu popularity contest would have served as a decent metric for popularity. We did perform an outlier analysis in which we compared the whole set of applications to the most popular applications (100 or more installations). We discovered that the statistical "footprint" of the popular applications is different from the whole. However, the details are omitted because this contradicted with our goals.

In our study, we concentrated on the POSIX networking APIs and Berkeley Sockets API because they form the de-facto, low-level API for all networking applications. However, we extended the API analysis to OpenSSL to study the use of security as well. All of these three APIs have bindings for high-level languages, such as Java and Python, and can be indirectly used from network application frameworks and libraries. As the API bindings used in other languages differs from those used in C language, we excluded other languages from this study.

From the data gathered [2], we calculated sums and means of the occurrences of each indicator. Then we also calculated a separate "reference" number. This latter was formed by introducing a binary value to denote whether a software package used a particular indicator (1) or not (0), independent of the number of occurrences. The reference number for a specific indicator was collected from all software packages, and these reference numbers were then summed and divided by the number of

[2]http://www.cs.helsinki.fi/u/sklvarjo/LS12/

packages to obtain a *reference ratio*. In other words, the reference ratio describes the extent of an API indicator with one normalized score.

We admit that the reference number is a very coarse grained metric; it indicates capability rather than 100% guarantee that the application will use a specific indicator for all its runs. However, it's binary (or "flattened") nature has one particular benefit that cancels out an unwanted side effect of the static code analysis, but this is perhaps easiest to describe by example. Let us consider an application where memory allocations and deallocations can be implemented in various ways. The application can call *malloc()* a hundred times but then calls *free()* only once. Merely looking at the volumes of calls would give a wrong impression about memory leaks because the application could have a wrapper function for *free()* that is called a hundred times. In contrast, a reference number of 1 for *malloc()* and 0 for *free()* indicates that the application has definitely one or more memory leak. Correspondingly, the reference ratio describes this for the entire population of the applications.

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

4 Results and Analysis

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

4.1 Core Sockets API

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

In the results, the reference ratios of indicators are usually shown inside brackets. All numeric values are from Ubuntu Lucid unless otherwise mentioned. Figure 1 illustrates some of the most frequent function indicators by their reference ratio and the following sections analyze the most interesting cases in more detail.

4.1.1 IPv6

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

4.1.2 Name Resolution

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

4.1.3 Packet Transport

Connection and datagram-oriented APIs were roughly as popular. Based on the usage of *SOCK_STREAM* and *SOCK_DGRAM* constants, we accounted for 25.1% TCP-only and 11.0% UDP-only applications. Hybrid applications supporting both protocols accounted for 26.3% - which leaves 37.6% of the applications that used neither of the constants. By combining the hybrids with TCP-only applications, the proportion of applications supporting TCP is 51.4% and, correspondingly, 37.3% for UDP. It should not be forgotten that typically all network applications implicitly access DNS over UDP by default.

4.1.4 Customizing Networking Stack

While the Sockets API provides transport-layer abstractions with certain system-level defaults, many applications preferred to customize the networking stack or

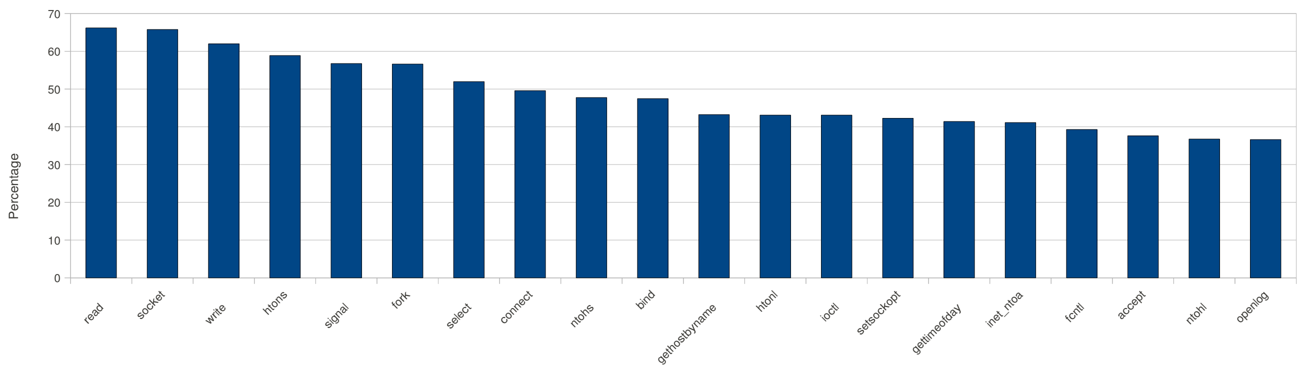


Figure 1: The most frequent functions in Ubuntu Lucid

to override some of the parameters. The combined reference ratio of *SOCK_RAW*, *setsockopt()*, *pcap_pkthdr* and *ipq_create_handle()* indicators was 51.4%. In other words, the default abstraction or settings of the Sockets API are not sufficient for the majority of the applications.

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

4.1.5 Multihoming and UDP

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

On Linux, we have reason to believe that many UDP-based applications may not handle multihoming properly for initial connections. The multihoming problem for UDP manifests itself only when a client-side application uses a server address that does not match with the default route at the server. The root of the problem lies in egress datagram processing at the server side.

The UDP problem occurs when the client sends a "request" message to the server and the server does not send a "response" using the exact same address pair that was used for the request. Instead, the sloppy server implementation responds to the client without specifying the source address, and the networking stack invariably chooses always the wrong source address - meaning that the client drops the response as it appears to be arriving from a previously unknown IP address.

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

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

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

4.2 Sockets API Extensions

In this section, we show and analyze statistics on SSL and the adoption of a number of Sockets API extensions.

4.2.1 Security: SSL/TLS Extensions

Roughly 10.9% of the software in the data set used OpenSSL and 2.1% GNU TLS. In this section, we limit the analysis on OpenSSL because it is more popular. Unless separately mentioned, we will, for convenience, use the term SSL to refer both TLS and SSL protocols. We only present reference ratios relative to the applications using OpenSSL because this is more meaningful from the viewpoint of the analysis. In other words, the percentages account only the 77 OpenSSL-capable applications and not the whole set of applications.

The applications using OpenSSL consisted of both client and server software. The majority of the applications using OpenSSL (54%) consisted of email, news and messaging software. The minority included network security and diagnostic, proxy, gateway, http and ftp server, web browsing, printing and database software.

The reference ratios of SSL options remained roughly the same throughout the various Ubuntu releases. The use of SSL options in Ubuntu Lucid is illustrated in Figure 2.

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

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

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

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

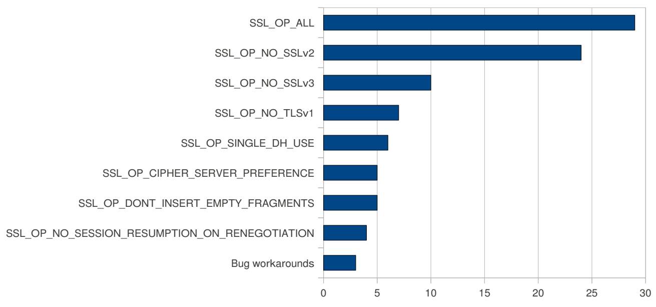


Figure 2: The number of occurrences of the most common SSL options

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

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

SSL_OP_NO_SSLV2 option (31.2%) turns off SSLv2 and respectively *SSL_OP_NO_SSLV3* (13.0%) turns off the support for SSLv3. The two options were usually combined so that the application would just use TLSv1.

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

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

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

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

4.2.2 IPv6-Related Extensions

During the long transition to IPv6, we believe that the simultaneous co-existence of IPv4 and IPv6 still represents problems for application developers. For example, IPv6 connectivity is still not guaranteed to work everywhere. At the client side, this first appears as a problem with DNS look-ups if they are operating on top of IPv6. Therefore, some applications may try to look up simultaneously over IPv4 and IPv6 [25]. After this, the application may even try to call *connect()* simultaneously over IPv4 and IPv6. While these approaches can decrease the initial latency, they also generate some additional traffic to the Internet and certainly complicate networking logic in the application.

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

An immediate solution to the concerns regarding address duplication is proposed in RFC4291 [6], which describes IPv6-mapped IPv4 addresses. The idea is to embed IPv4 addresses in IPv6 address structures and thus to provide a unified data structure format for storing addresses in the application.

Mapped addresses can be employed either manually or by the use of *AI_V4MAPPED* flag for the *getaddrinfo()* resolver. However, the application first has to explicitly enable the *IPV6_V6ONLY* socket option (0.1%) before the networking stack will allow the IPv6-based socket to be used for IPv4 networking. By default, IPv4 connectivity with IPv6 sockets is disallowed in Linux because they introduce security risks [10]. As a bad omen, of the total six applications referencing the *AI_V4MAPPED* flag, only one of them set the socket option as safe guard.

The constants introduced by the IPv6 Socket API for Source Address Selection [13] are available in Ubuntu Lucid even though the support is incomplete. The flags to extend the *getaddrinfo()* resolver and the proposed auxiliary functions remain unavailable and only source address selection through socket options is available. Nevertheless, we calculated the proportion of IPv6-capable client-side applications that explicitly choose a source address. As an upper bound, 66.9% percent applications choose source addresses explicitly based the dual use of *connect()* and *bind()*. This means that a majority of IPv6 applications might be potentially interested of the extensions for IPv6 Socket API for Source Address Selection.

4.2.3 Other Protocol Extensions

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

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

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

The APIs for HIP-aware applications [9] may also face a similar slow adoption path because the APIs require a new domain type for sockets in the Linux kernel. While *getaddrinfo()* function can conveniently look up "wildcard" domain types, the success of this new DNS resolver (23.5%) is still challenged by the deprecated *gethostbyname()* (43.3%). SHIM6 does not face the same problem as it works without any changes to the resolver and connections can be transparently "upgraded" to SHIM6 during the communications.

The shared multihoming API for HIP- and SHIM6-aware applications [8] may have a smoother migration path. The API relies heavily on socket options and little on ancillary options. This strikes a good balance because *setsockopt()* is familiar to application developers (42.8%) and *sendmsg()* / *recvmsg()* with its ancillary option is not embraced by many (7%). The same applies to the API for Multipath TCP [16] that consists solely of socket options.

4.2.4 A Summary of the Sockets API Findings and Their Implications

Table 2 highlights ten of the most important findings in the Sockets APIs. Next, we go through each of them and argue their implications to the development of network applications.

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

Table 2: Highlighted indicator sets and their reference ratios

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

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

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

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

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

Findings 6, 7 and 8. Roughly every tenth application was using OpenSSL but surprisingly many failed to initialize it appropriately, thus creating potential security vulnerabilities. Half of the OpenSSL-capable applications were modifying the default configuration in some way. Many of these tweaks improved backwards compatibility at the expense of security. This opens a question why backwards compatibility is not well built into OpenSSL and why so many "knobs" are even offered to the developer[3].

[3]Some of the implementations of SSL/TLS are considered "broken"; they do not implement at all or fix incorrectly some of the bugs and/or functionalities in SSL/TLS.

Finding 9. IPv6-mapped IPv4 addresses should not be leaked to the wire for security reasons. As a solution, the socket option *IPV6_V6ONLY* would prevent this leakage. However, only one out of total six applications using mapped addresses were actually using the socket option. Despite the number of total applications using mapped address in general was statistically small, this is an alarming sign because the number can grow when the number of IPv6 applications increases.

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

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

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

4.3 Network Application Frameworks

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

4.3.1 Generic Requirements for Modern Frameworks

Instead of applying the highlighted findings described in Section 4.2.4 directly, we some modifications due to the different nature of network application frameworks.

Firstly, we reorganize the analysis "top down" and split the topics into end-host naming, look up, multiplicity of names and transport protocols and security. We also believe that the reorganization may be useful for extending the analysis in the future.

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

Consequently, the following list reflects the Sockets API findings as modified requirements for network application frameworks:

R1: End-host naming

- R1.1 Does the API of the framework support symbolic host names in its APIs, i.e., does the framework hide the details of hostname-to-address resolution from the application? If this is true, the framework conforms to a similar API as proposed by Name Based Sockets as described in section 2.2. A benefit of this approach is that implementing requirements R1.2, R2.2, R3.1 and 3.3 becomes substantially easier.
- R1.2 Are the details of IPv6 abstracted away from the application? In general, this requirement facilitates adoption of IPv6. It could also be used for supporting Teredo based NAT traversal transparently in the framework.

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

R2: Look up of end-host names

- R2.1 Does the framework implement DNS look ups with *getaddrinfo()*? This is important for IPv6 source address selection and native HIP API extensions because they are dependent on this particular function.
- R2.2 Does the framework support parallel DNS look ups over IPv4 and IPv6 to optimize latency?

R3: Multiplicity of end-host names

- R3.1 IPv6 source address selection is not widely adopted yet but is the framework modular enough to support it especially at the client side? As a concrete example, the framework should support inclusion of new parameters to its counterpart of *connect()* call to support application preferences for source address types.
- R3.2 Does the server-side multihoming for UDP work properly? As described earlier, the framework should use *SO_BINDTODEVICE* option or *sendmsg()/recvmsg()* interfaces in a proper way.
- R3.3 Does the framework support parallel *connect()* over IPv4 and IPv6 to minimize the latency for connection set-up?

R4: Multiplicity of transport protocols

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

R5: Security

- R5.1 Does the framework support SSL/TLS?
- R5.2 Does the SSL/TLS interface provide reasonable defaults and abstraction so that the developer does not have to configure the details of the security?
- R5.3 Does the framework initialize the SSL/TLS implementation automatically?

4.3.2 ACE

ACE version 6.0.0 denotes one end of a transport-layer session with *ACE_INET_Addr* class that can be initiated both based on a symbolic host name and a numeric IP address. Thus, the support for IPv6 is transparent if the developer resorts solely on host names and uses *AF_UNSPEC* to instantiate the class. ACE supports also storing of IPv4 addresses in the IPv6-mapped format internally but translates them to the normal IPv4 format before returning them to the requesting application or using on the wire.

In ACE, IP addresses can be specified using strings. This provides a more unified format to name hosts.

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

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

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

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

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

ACE supports SSL albeit it is not as interchangeable as TCP with UDP. ACE has wrappers around *accept()* and *connect()* calls in its Acceptor-Connector pattern. This hides the intricacies of SSL but all of the low-level details are still configurable when needed. SSL is initialized automatically and correctly.

4.3.3 Boost::Asio

Boost::Asio version 1.47.0 provides a class for denoting one end of a transport-layer session called *endpoint* that can be initiated through resolving a host name or a numeric IP. By default, the resolver returns a set of endpoints that may contain both IPv4 and IPv6 addresses [4]. These endpoints can be given directly to the *connect()* wrapper in the library that connects sequentially to the addresses found in the endpoint set until it succeeds. Thus, the support for IPv6 is transparent if the developer has chosen to rely on host names. Boost::Asio can store IPv4 addresses in the IPv6-mapped form. By default, the mapped format is used only when the developer explicitly sets the family of the address to be queried to IPv6 and the query results contain no IPv6 addresses. The mapped format is only used internally and converted to IPv4 before use on the wire.

[4]IPv6 addresses are queried only when IPv6 loopback is present

Boost::Asio uses POSIX *getaddrinfo()* when the underlying OS supports it. On systems such as Windows (older than XP) and Cygwin, Boost::Asio emulates *getaddrinfo()* function by calling *gethostbyaddr()* and *gethostbyname()* functions. The resolver in Boost::Asio includes flags that could be used for implementing source address selection (and socket options are supported as well).

Boost::Asio does not support parallel IPv4 and IPv6 queries, nor does it provide support for simultaneous connection set up using both IPv4 and IPv6.

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

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

Boost::Asio supports SSL and TLS. The initialization is wrapped into the SSL context creation. In Boost::Asio, the library initialization is actually done twice as *OpenSSL_add_ssl_algorithms()* is a synonym of *SSL_library_init()* and both are called sequentially. PRNG is not automatically initialized with *RAND_load_file()*, *RAND_add()* or *RAND_seed()*, although Boost::Asio implements class *random_device* which can be easily used in combination with *RAND_seed()* to seed the PRNG.

4.3.4 Java.net

Java.net in OpenJDK Build b147 supports both automated connections and manually created ones. Within a single method that inputs a host name, its API hides resolving a host name to an IP address from DNS, creation of the socket and connecting the socket. Alternatively, the application can manage all of the intermediate steps by itself.

The API has a data structure to contain multiple addresses from DNS resolution. The default is to try a connection only with a single address upon request, albeit this is configurable. The internal presentation of a single address, *InetAddress*, can hold an IPv4 or IPv6 address and, therefore, the address family is transparent when the developer resorts solely on the host names. The API supports *v4_mapped address* format as an internal presentation format but it is always converted to the normal IPv4 address format before sending data to the network.

Before using IPv6, Java.net checks the existence of the constant *AF_INET6* and that a socket can be associated with a local IPv6 address. If java.net discovers support for IPv6 in the local host, it uses the *getaddrinfo()* but otherwise *gethostbyname()* function for name resolution. DNS queries simultaneously over IPv4 and IPv6 are not supported out-of-the-box. However, the SIP ParallelResolver package in SIP communicator [5] could be used to implement such functionality.

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

Java.net naming convention favors TCP because a "socket" always refers to a TCP-based socket. If the developer needs a UDP socket, he or she has to instantiate a *DatagaramSocket* class. Swapping between the two protocols is not trivial because TCP-based communication uses streams, where as UDP-based communication uses *DatagramPacket* objects for I/O.

IPv6 source address selection is implementable in java.net. TCP and UDP-based sockets could include a new type of constructor or method, and java has socket options as well. The method for DNS look ups, *InetAddress.getByName()*, is not extensive enough and would need an overloaded method name for the purpose.

Java.net supports both SSL and TLS. Their details are hidden by abstraction, although it is possible to configure them explicitly. All initialization procedures are automatic.

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

4.3.5 Twisted

With Twisted version 10.2, python-based applications can directly use host names to create TCP-based connections. However, the same does not apply to UDP; the application has to manually resolve the host name into an IP address before use.

With the exception of resolving of AAAA records from the DNS, IPv6 support is essentially missing from Twisted. Thus, mapped addresses and parallel connections over IPv4 and IPv6 remain unsupported due to lack of proper IPv6 support. Some methods and classes include "4" suffix to hard code certain functions only to IPv4 which can hinder IPv6 interoperability.

Introducing IPv6 source address selection to Twisted would be relatively straightforward, assuming IPv6 support is eventually implemented. For example, Twisted methods wrappers for *connect()* function input host names. Therefore, the methods could be adapted to include a new optional argument to specify source address preferences.

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

The UDP multihoming issue is also present in Twisted. We observed this by experimenting with a couple of client and server UDP applications in the Twisted source package.

TCP and UDP are quite interchangeable in Twisted when the application uses the *Endpoint* class because it provides abstracted read and write operations. However, two discrepancies exists. First, *Creator* class is tainted by TCP-specific naming conventions in its method *connectTCP()*. Second, applications cannot read or write UDP datagrams directly using host names but first have to resolve them into IP addresses.

Twisted supports TLS and SSL in separate classes. TLS/SSL can be plugged into an application with relative ease due to modularity and high-level abstraction of the framework. The details of SSL/TLS are configurable and Twisted provides defaults for applications that do not need special configurations. With the exception of seeding the PRNG, the rest of the details of TLS/SSL initialization are handled automatically.

4.3.6 A Summary of the Framework Results

We summarize how the requirements were met by each of the four frameworks in Table 3. Some of the requirements were unmet in all of the frameworks. For example, all frameworks failed to support UDP-based multihoming (R3.2) and parallel IPv4/IPv6 connection initialization for clients (R3.3). Also, SSL/TLS initialization (R5.3) was not implemented correctly in all frameworks. In total, 56 % of our requirements were completely met in all of the frameworks.

Req.	ACE	Boost::Asio	Java.net	Twisted
R1.1	✓		✓	(✓)
R1.2	✓	✓	✓	
R1.3	✓	✓	✓	N/A
R2.1	✓	✓	✓	
R2.2				
R3.1	✓	✓	✓	✓
R3.2				
R3.3				
R4.1	✓	✓		(✓)
R5.1	✓	✓	✓	✓
R5.2	✓	✓	✓	✓
R5.3	✓	(✓)	✓	(✓)

Table 3: Summary of how the frameworks meet the requirements

5 Related and Future Work

At least three other software-based approaches to analyze applications exist in the literature. Camara et al. [3] developed software and models to verify certain errors in applications using the Sockets API. Ammons et al. [1] have investigated machine learning to reverse engineer protocol specifications from source code based on the Sockets API. Palix et al. [14] have automatized finding of faults in the Linux kernel and conducted a longitudinal study.

We did not focus on the development of automatized software tools but rather on the discovery of a number of novel improvements to applications and frameworks using the Sockets API. While our findings could be further automatized with the tools utilized by Camara, Ammons and Palix et al., we believe such an investigation would be in the scope of another article.

Similarly to our endeavors with multihoming, Multiple Interfaces working group in the IETF tackles the same problem but in broader sense [2, 24]. Our work supplements their work, as we explained a very specific multihoming problem with UDP, the extent of the problem in Ubuntu Linux and the technical details how the problem can be addressed by developers.

6 Conclusions

In this article, we showed empirical results based on a statistical analysis of open-source network software. Our aim was to understand how the Sockets APIs and its extensions are used by network applications and frameworks. We highlighted ten problems with security, IPv6 and configuration. In addition to describing the generic technical solution, we also reported the extent of the problems. As the most important finding, we discovered that 28.6% of the C-based network applications in Ubuntu are vulnerable to attacks because they fail to initialize OpenSSL properly.

We applied the findings with C-based applications to four example frameworks based on the Sockets API. Contrary to the C-based applications, we analyzed the frameworks in a top-down fashion along generalized dimensions of end-host naming, multiplicity of names and transports, name look up and security. Consequently, we proposed 12 networking requirements that were completely met by a little over half of the frameworks in total. For example, all four frameworks consistently failed to support UDP-based multihoming and parallel IPv4/IPv6 connection initialization for the clients. Also the TLS/SSL initialization issue was present in some of the frameworks. With the suggested technical solutions for Linux, we argue that hand-held devices with multi-access capabilities have improved support for UDP, the end-user experience can be improved by reducing latency in IPv6 environments and security is improved for SSL/TLS in general.

7 Acknowledgments

We would like to thank Tao Wan for his initial work with the topic. We appreciate the discussion with Dmitriy Kuptsov, Antti Louko, Teemu Koponen, Antti Ylä-Jääski, Jukka Nurminen, Andrey Lukyanenko, Boris Nechaev, Zhonghong Ou, Cui Yong, Vern Paxon, Stefan Götz and Suvi Koskinen around the topic. The authors also express their gratitude to anonymous reviewers for their comments. This work was supported by grant numbers 139144 and 135230 from the Academy of Finland.

References

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

[2] M. Blanchet and P. Seite. Multiple Interfaces and Provisioning Domains Problem Statement. RFC 6418 (Informational), November 2011.

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

[4] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC 6182 (Informational), March 2011.

[5] R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens. Basic Socket Interface Extensions for IPv6. RFC 3493 (Informational), February 2003.

[6] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), February 2006. Updated by RFCs 5952, 6052.

[7] C. Huitema. RFC 4380: Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs), February 2006.

[8] M. Komu, M. Bagnulo, K. Slavov, and S. Sugimoto. Sockets Application Program Interface (API) for Multihoming Shim. RFC 6316 (Informational), July 2011.

[9] M. Komu and T. Henderson. Basic Socket Interface Extensions for the Host Identity Protocol (HIP). RFC 6317 (Experimental), July 2011.

[10] Craig Metz and Jun ichiro itojun Hagino. IPv4-Mapped Addresses on the Wire Considered Harmful, October 2003. Work in progress, expired in Oct, 2003.

[11] Robert Moskowitz, Pekka Nikander, Petri Jokela, and Thomas R. Henderson. RFC 5201: Host Identity Protocol, April 2008.

[12] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533 (Proposed Standard), June 2009.

[13] E. Nordmark, S. Chakrabarti, and J. Laganier. IPv6 Socket API for Source Address Selection. RFC 5014 (Informational), September 2007.

[14] Nicolas Palix, Gaël Thomas, Suman Saha, Christophe Calvès, Julia L. Lawall, and Gilles Muller. Faults in linux: ten years later. In Rajiv Gupta and Todd C. Mowry, editors, *ASPLOS*, pages 305–318. ACM, 2011.

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

[16] Michael Scharf and Alan Ford. MPTCP Application Interface Considerations, November 2011. Work in progress, expires in June, 2012.

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

[18] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), January 2001.

[19] W. Stevens, M. Thomas, E. Nordmark, and T. Jinmei. Advanced Sockets Application Program Interface (API) for IPv6. RFC 3542 (Informational), May 2003.

[20] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *Unix Network Programming, Volume 1, The Sockets Networking API*. Addison-Wesley, 2004. Fourth printing.

[21] R. Stewart. RFC 4960: Stream Control Transmission Protocol, September 2007.

[22] T.Dierks and E. Rescorla. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, August 2008.

[23] Javier Ubillos, Mingwei Xu, Zhongxing Ming, and Christian Vogt. Name Based Sockets, September 2010. Work in progress, expires in March 2011.

[24] M. Wasserman and P. Seite. Current Practices for Multiple-Interface Hosts. RFC 6419 (Informational), November 2011.

[25] D. Wing and A. Yourtchenko. Happy Eyeballs: Success with Dual-Stack Hosts. RFC 6555 (Proposed Standard), April 2012.

Publication II

Miika Komu, Sasu Tarkoma, Jaakko Kangasharju and Andrei Gurtov. Applying a Cryptographic Namespace to Applications. In *Dynamic Interconnection of Networks Workshop (DIN'05)*, Proceedings of the 1st ACM Workshop on Dynamic Interconnection of Networks (co-located with Mobicom 2005 Conference), Cologne, Germany, pp. 23-27, ISBN 1-59593-144-9, September 2005.

Applying a Cryptographic Namespace to Applications

Miika Komu
Helsinki Institute for Information Technology
Advanced Research Unit
P.O. Box 9800
FIN-02015 HUT, Finland
miika@iki.fi

Sasu Tarkoma
Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
P.O.Box 5400
FIN-02015 HUT
sasu.tarkoma@tml.hut.fi

Jaakko Kangasharju
Helsinki Institute for Information Technology
Advanced Research Unit
P.O. Box 9800
FIN-02015 HUT, Finland
jkangash@hiit.fi

Andrei Gurtov
Helsinki Institute for Information Technology
Advanced Research Unit
P.O. Box 9800
FIN-02015 HUT, Finland
gurtov@cs.helsinki.fi

ABSTRACT

The Host Identity Protocol (HIP) is a promising solution for dynamic network interconnection. HIP introduces a namespace based on cryptographically generated Host Identifiers. In this paper, two different API variants for accessing the namespace are described, namely the legacy and the native APIs. Furthermore, we present our implementation experience on applying the APIs to a number of applications, including FTP, telnet, and personal mobility. Well-known problems of callbacks and referrals, i.e., passing the IP address within application messages, are considered for FTP in the context of HIP. We show that the callback problem is solvable using the legacy API. The APIs are important for easy transition to HIP-enabled networks. Our experimentation with well-known network applications indicate that porting applications to use the APIs is realistic.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*applications, protocol architecture*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Experimentation, Security, Standardization

Keywords

Host Identity Protocol, sockets API, referral, personal mobility

1. INTRODUCTION

The interconnection of mobile nodes, mobile networks, and multi-homed hosts is a challenging task within the current Internet architecture. The Host Identity Protocol (HIP) being developed by the IETF [8] is a promising solution to address these issues. The HIP layer is located between the network and transport layers and provides a new cryptographic addressing space for applications, where communication endpoints are identified using public cryptographic keys instead of IP addresses. However, HIP by itself provides no benefits unless there are applications using the protocol. In this paper, we consider the important problems of accommodating legacy applications to run on top of HIP, and of designing a native HIP API for new networking applications.

The fundamental idea behind HIP is to divide the address of a network-addressable node to two parts: the *identifier* and *locator* parts. The identifier part uniquely names the host using a cryptographic namespace and the locator part uniquely defines the topological location of the node in the network.

The main benefits of the new HIP namespace are statistically unique identifiers, separation of identifiers from their topological location, better support for delegation and intermediaries, multi-homing/mobility support, and security features such as authentication and confidentiality [8, 1, 15]. Recently, many systems based on globally unique flat namespaces have been proposed, including Unmanaged Internet Protocol (UIP) [3], i3 [15], and Delegation Oriented Architecture (DOA) [1]. IPv6 addresses some of the concerns with IPv4 and Network Address Translations (NATs), but still couples the identity of the hosts with the location.

The introduction of a new namespace requires consideration of two new architectural issues: how the new namespace is used in packets, and how the identifiers are resolved and distributed. In addition, mechanisms that allow applications to leverage the properties of the namespace are needed. We focus on the applications and present two APIs, *legacy API* [4] and *native HIP API* [6]. The APIs are all included in our HIP for Linux implementation [2].

The rest of the paper is organized as follows. A brief overview of HIP architecture is given in Section 2. In Section 3, the legacy and native APIs for HIP are described. In Section 4, we illustrate the use of HIP APIs for FTP, Telnet, and personal mobility applications. Section 5 concludes the paper.

2. THE HIP NAMESPACE

HIP [8] introduces a new Host Identifier (HI) namespace for the Internet. The HIs are disjoint from the IPv4 and IPv6 namespaces in order to provide location independent identification of upper-layer endpoints. By decoupling the network-layer identifiers from the upper-layer identifiers, the HIP architecture provides a sound foundation on which to build mobility and multi-homing support. The upper layers have stable endpoint identifiers, but network-layer addresses can change dynamically.

The endpoints are identified using asymmetric cryptography. A HI is the public key component of an asymmetric key pair. The private key is owned by the endpoint, making impersonating another endpoint very difficult. HIP uses the HIs as Transport Layer Identifiers (TLIs). The locators, i.e., IP addresses, are used only in the network layer. There is a one-to-many binding between a HI and the corresponding locators [11]. As the HI is essentially a variable-sized public key, it is difficult to use in datagram headers. Further, long identifiers are difficult to support in the sockets API because it imposes a limit of 255 bytes to socket address structures. To address these problems, the HIP architecture also includes fixed-size representations of the HI. A Host Identity Tag (HIT) is a 128-bit long hash of the HI, and a Local Scope Identifier (LSI) is a 32-bit representation of the HIT.

In the traditional TCP/IP model, connection associations in the application layer are uniquely distinguished by the source IP address, destination IP address, source port, destination port, and transport protocol type. HIP changes this model by using HITs in the place of IP addresses. The HIP model is further expanded in the native HIP API model by using Endpoint Descriptors (EDs) instead of HITs. Now, the application layer uses source ED, destination ED, source port, destination port, and transport protocol type to distinguish between different connection associations. The namespace model used in the native HIP API is shown in Figure 1.

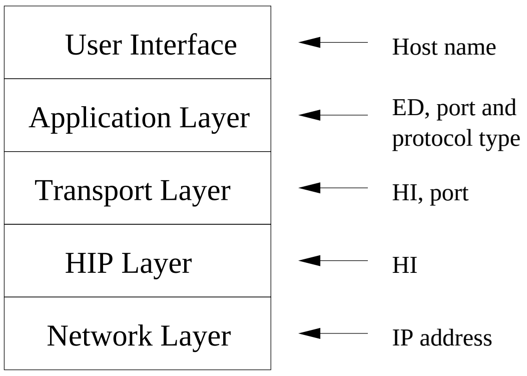


Figure 1: The HIP namespace model

The ED is used for hiding the representation of endpoints from applications in the native HIP API. It acts as a handle or an alias to the corresponding HI on the host. It is an integer having only local significance, similar to a file or socket descriptor. This kind of identifier with only local significance appears also in other namespace models, such as in OCALA [5].

The difference between the application and transport layer identifiers is that the transport layer uses HIs instead of EDs. The TLI is named with source HI, destination HI, source port, and destination port at the transport layer. Correspondingly, the HIP layer uses HIs as identifiers. The HIP Security Associations (SAs) are based on source HI and destination HI pairs. The network layer uses IP addresses, i.e., locators, for routing. The network layer interacts with the HIP layer to exchange information about changes in the addresses of local interfaces and peers.

The native HIP API socket bindings are visualized in Figure 2. A HIP socket is associated with one source and one destination ED, along with their port numbers and the protocol type. Multiple EDs and ports can be associated with a single HI. Further, the source HI is associated with a set of network interfaces at the local host. The destination HI, in turn, is associated with a set of destination addresses of the peer.

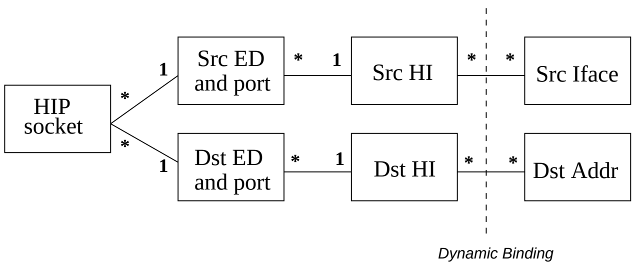


Figure 2: Native HIP API socket bindings

We believe that using EDs instead of HITs at the application layer has two useful properties. First, it simplifies implementing opportunistic base exchange, and second, EDs can be seen as a higher-layer concept to separate application-layer identifiers from those of lower layers.

In *opportunistic base exchange* the initiator does not know the responder's HIT, but only its IP address. Trying to implement this with the legacy API using the standard sockets API forces the application to associate its socket with the responder's IP address instead of its HIT. This increases the complexity of the HIP implementation, since a mapping from IPs to HITs is now needed, and it may not function reliably when IP addresses change due to mobility. In the native HIP API, however, the application binds to an ED. The HIP implementation can then transparently associate this ED with the responder's HIT that is learned later during the base exchange. Since EDs are already used in the native HIP API, supporting opportunistic mode does not increase the complexity of the HIP implementation.

It also seems to us that the new abstraction layer provided by the EDs may have some synergy with service identifiers [1] or session layer identifiers [14]. However, we are currently investigating this idea, and will not consider it further in this paper.

3. HIP API

In this section, we describe two APIs for applications to access the HIP namespace. The APIs are described in the C programming language, although Java is also supported by our implementation. First, we present the legacy API which is intended as an easy migration path towards HIP-enabled applications. Next, we present the native HIP API that allows applications to fully utilize the new namespace and protocol. Finally, we discuss problems related to referrals.

3.1 Legacy API

Network applications typically use host names to address peers. Host names have to be resolved to IPv6 addresses from the Domain Name System (DNS) in the resolver library before network connections can be established with peers. In the legacy API, the resolver routine has been modified to prefer HITs as the result of DNS queries instead of IPv6 addresses. Otherwise, the legacy API appears like the standard sockets API to the application.

We modified the resolver library to support HIP in two ways. In *transparent mode*, the DNS queries resolve silently to HITs instead of IPv6 addresses. For backwards compatibility, the resolver returns IP addresses if no HITs were found. The greatest benefit of the transparent mode is that it requires no changes in the application. However, a drawback of the transparent mode is that the resolver is not guaranteed to always return HITs. To address this shortcoming, applications can use the resolver in *explicit mode* by passing a flag explicitly to the resolver. This flag enforces the use of HIP by making the resolver return only HITs to the application. Effectively, this means that connections will be established using HIP or not at all. This way, HIP can be used with minimal changes in HIP-aware applications.

Example code using the legacy API is shown in Figure 3. The only modification from a standard socket application is the use of a flag flag to enable the explicit mode.

```
struct addrinfo hints, *res, *try;
char *hello = "hello";
int err, int bytes, sock;

memset(hints, 0, sizeof(hints));
hints.ai_flags = AI_HIP;
hints.ai_family = AF_INET6;
hints.ai_socktype = SOCK_STREAM;

err   = getaddrinfo("www.host.org", "echo",
                    &hints, &res);
sock  = socket(res->ai_family,
               res->ai_socktype,
               res->protocol);
for (try = res; try; try = try->ai_next)
  err = connect(sock, try->ai_addr,
               try->ai_addrlen);

bytes = send(sock, hello, strlen(hello), 0);
bytes = recv(sock, hello, strlen(hello), 0);
err   = close(sock);
err   = freeaddrinfo(res);
```

Figure 3: A "Hello, world" client using the legacy API.

3.2 Native HIP API

The legacy API requires only minor changes in applications, and therefore it cannot utilize all features of a HIP-enabled networking stack. Applications requiring more control over the HIP layer can use the native HIP API [6]. The most significant difference between the legacy and the native APIs is that the native HIP API can use public-key identities in the userspace sockets API. A direct benefit of this is that the users can provide their own public key identifiers to the networking stack. As a result, the identities are not bound to just hosts; they can be bound to users, processes, or groups. For instance, process migration systems [7] may benefit from this as the HI can be moved along with the process. In addition, if DNS is used to store public keys instead of HITs [10], the explicit public key handling in the native HIP API should become useful.

We propose a *PF_HIP* protocol family to be available in HIP-enabled network stacks. HIP-aware applications use the existing transport layer sockets API and specify this new protocol family when creating sockets. By creating a HIP-enabled socket, an application can detect whether HIP is supported on the local host. Similarly, an application can detect HIP support in a peer host by resolving the EDs of the peer. If the peer does not support HIP, the resolver returns an empty set.

The syntax of the native HIP API is similar to the legacy API. The crucial differences are the use of PF_HIP instead of AF_INET6, and a new socket structure for EDs. The resolver function is used in a similar way as the legacy API resolver [6]. An example use of the native HIP API is shown in Figure 4. The example uses an application-specified identifier from the file /home/mk/hip_host_dsa_key.

```
int sockfd, err, family = PF_HIP,
    type = SOCK_STREAM;
char *user_priv_key = "/home/mk/hip_host_dsa_key";
struct endpoint *endpoint;
struct sockaddr_ed my_ed;
struct endpointinfo hints, *res = NULL;

err = load_hip_endpoint_pem(user_priv_key,
                            &endpoint);
err = setmyeid(&my_ed, "", endpoint, NULL);
sockfd = socket(family, type, 0);
err = bind(sockfd, (struct sockaddr *) &my_ed,
           sizeof(my_ed));

memset(&hints, 0, sizeof(&hints));
hints.ei_socktype = type;
hints.ei_family = family;
err = getendpointinfo("www.host.org", "echo",
                      &hints, &res);

/* connect, send and recv as in Figure 3 */
```

Figure 4: A "Hello, world" client with application-specified identifiers in the native HIP API.

An application can control the HIP layer better using the native HIP API than the legacy API. For example, the application can set the base exchange puzzle to be more difficult for a specific server port number, request for higher SA lifetimes, use smaller (and less secure) key lengths, or even specify its own HIs. Quality of Service (QoS) related attributes can also be accessed through the native HIP API to allow the simultaneous use of multiple IP flows. This enables applications to benefit from soft-handover strategies, or to select a data path depending on the available QoS. For example, the application can be notified when a LAN interface of the host is activated, so that the application can use it for data traffic instead of a slow WLAN link.

3.3 The Referral Problem

HIP introduces a new address space for the transport layer. Basically, the address space is flat although it is possible to use type 2 [8, 10] HITs that contain a domain prefix. Using the prefix, HITs can be resolved to IP addresses from the DNS. However, the problem with this approach is that it has some security implications due to the increased probability of HIT collisions. As a consequence, we may need to have full-length type 1 HITs [8, 10] in the future.

However, this causes problems for applications that need a remote application to initiate a connection. Currently they communicate either their own IP address (*callback*) or that of a third party (*referral*) [12]. The remote application will later connect to the communicated address. Using the legacy API with such applications would replace these IP addresses with HITs. However, since HITs cannot be resolved to IP addresses in the current DNS infrastructure, the remote application cannot typically initiate the required connection.

There are at least three ways to solve this problem. One way is to modify the DNS infrastructure to support type 2 HIT lookup. The second way is to use an overlay based on a flat namespace such as Internet Indirection Infrastructure (i3) [17] to support resolving of HITs. Third, the overlay can also be used for packet routing, at least for the initial HIP signaling [9], but this is out of the scope of this paper.

4. HIP APPLICATIONS

In this section, we present three HIP-enabled applications. We begin with an FTP application, which has been labeled by the community as challenging for HIP. Then, we examine a Telnet application that was ported to use the native HIP API. Finally, we describe an application of personal mobility with HIP.

4.1 FTP and Referrals

File Transfer Protocol (FTP) [13] uses two separate channels (TCP connections) for communication, one for control and one for data. The data channel can be initiated in two ways. In a passive mode, the server passes its IP address and port number as a callback to the client using the control channel. Then the client initiates a data channel to the server based on the IP address and port number given by the server. In an active mode, it is the vice versa: the server initiates the data channel to the IP address and port given by the client.

The FTP way of passing addresses as callbacks can be considered problematic when HIP is used because HITs are used instead of IP addresses. The crux of the problem is that the application may not be able to resolve a given HIT to a routable IP address. We decided to experiment with this problem using the legacy API with an initial expectation of failure.

Our callback experiment used IPv6-enabled FTP client and server software (lftp version 3.1.3 and proftpd version 1.2.10). Both the client and the server used the legacy API, thus requiring no modifications. We carried out a simple test where the client contacted the HIT of the server and downloaded a file. Surprisingly, both modes, active and passive, worked properly. We also experimented with a mobility handover during file download by changing the currently active address of the client. This caused only a relatively small delay during the file download.

The reason why the callback worked in the case of FTP is that once the client (initiator) and server (responder) have established a security association, they are aware of each other's HIT-to-IP mappings. The mapping from the HIT to IP address(es) is not lost because it is valid at least for the lifetime of the IPsec SA.

However, the callbacks are only a part of the problem. In the FTP case, it is possible to use referrals instead of callbacks, but fortunately this feature is rarely used. The referral problem occurs when the client creates a new data channel using the FTP protocol to server A, but redirects it to another server B. As the redirection is based on a HIT, server B must resolve the HIT to an IP address, which requires support from the infrastructure. We already described three general solutions to this problem in Section 3.3. Additionally, it would also be possible to extend the FTP protocol to use either FQDNs or HITs and IP addresses together.

4.2 Telnet

We ported an IPv6-enabled Telnet client and daemon to use the native HIP API. We configured the native HIP API into the code as a compile-time option. The porting process itself was quite straightforward. As the native HIP API resolver name and related data structure are named differently from their IP-based API correspondents, the porting process consisted mainly of search and replace operations in the source code. The API names are different to emphasize the introduction of the new namespace in the resolver but the syntax is almost identical [6].

4.3 Personal Mobility

Personal mobility and device personalization are becoming an important part of applications and mass-market devices. Personal mobility occurs when the user changes devices. Personalization is needed to change the user experience on a new device to meet the user's expectations. Almost all recent mobile phones support personalization of the device to accommodate the user's preferences, for example in call settings, buddy-lists, and user interface appearance.

A HI can be used to support personal mobility and device personalization. This is accomplished by associating the HI with a user and using a smartcard or a USB stick to store the HI. Personal mobility takes place when the user inserts the identity storage device containing the HI into a terminal device. The HI can then be used to initiate a HIP connection, and to support mobility and multi-homing. The HI may also be used to locate, download, and synchronize data needed for device personalization, such as device, user interface, and preference profiles. Since the HI is a public cryptographic key, it allows authentication of the client as well as confidentiality of the personalization data.

We experimented with a scenario in which the HI is stored on a USB memory stick and can be moved between different machines. The insertion of the USB stick is detected automatically. After detection, the HI is loaded to the HIP kernel module, and then used by applications. We demonstrated HIP-based connections in personal mobility for file synchronization and for streaming audio playback.

In our scenario, the USB stick only contains the HI, and the connections are not persistent. In the future we also plan to store data related to the session state on the stick so that connections can be restored at the new location. In addition, we envision that the USB stick can be replaced with a smart card that can create and verify signatures directly. This way, users can use their personal identities even on untrusted hosts (for example in Internet cafes) without compromising their private keys. Further, users can prevent other people from tracking their personal identifier and location by using either short-lived HIs or "blinded" HITs [16].

5. CONCLUSION

In this paper we have presented the legacy and the native APIs of our Linux-based HIP implementation. The legacy API does not necessarily require changes to applications. The native API requires modifications, but allows applications to provide their own public key identities. The cryptographic namespace is useful for applications and we discussed the implications for three example applications: FTP, Telnet, and personal mobility and personalization. Initially, we expected HIP-enabled FTP to be hindered by the callback problem, but our analysis and experimentation showed that callbacks are not an issue. We observed that it was relatively straightforward to port a Telnet utility to use the native HIP API. As an example of the benefits of the native HIP API, we discussed personal mobility and device personalization using Host Identifiers and USB sticks.

As a conclusion, we envisage that simple network applications use the legacy API in a transparent fashion, and more advanced applications utilize the new namespace using the native HIP API. We expect that the migration to HIP may require changes in some applications, but our experiments with basic networking utilities and the legacy API indicate that the introduction of the new namespace is realistic.

6. ACKNOWLEDGMENTS

We thank Jukka Ylitalo, Jeff Ahrenholz, Teemu Koponen, Abhinav Pathak and Thomas Henderson for their comments on the the paper. Niklas Karlsson developed a small feature to the HIP module that was required for the FTP daemon to work (binding to a HIT using the legacy API). We also thank the anonymous reviewers for their helpful comments.

7. REFERENCES

[1] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A Layered Naming Architecture for the Internet. In *Proc. of ACM SIGCOMM'04*, pages 343–352, Aug. 2004.

[2] C. Candolin, M. Komu, M. Kousa, and J. Lundberg. An implementation of HIP for Linux. In *Proc. of the Linux Symposium*, July 2003.

[3] B. Ford. Unmanaged Internet Protocol: taming the edge network management crisis. *ACM Computer Communication Review*, 34(1):93–98, 2004.

[4] T. R. Henderson. Using HIP with legacy applications: draft-henderson-hip-applications-01, July 2005. Work in progress. Expires in January 19, 2006.

[5] J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. Supporting legacy applications over i3. Technical Report UCB/CSD-04-1342, University of California at Berkeley, May 2004.

[6] M. Komu. *Native Application Programming Interfaces for the Host Identity Protocol: draft-mkomu-hip-native-api-00.* Internet Engineering Task Force, Sept. 2004. Work in progress. Expires August, 2005.

[7] T. Koponen, A. Gurtov, and P. Nikander. Application mobility with Host Identity Protocol. In *Proc. of NDSS Wireless and Security Workshop*, San Diego, CA, USA, Feb. 2005. Internet Society.

[8] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host Identity Protocol: draft-ietf-hip-base-03, June. 2005. Work in progress. Expires in December, 2005.

[9] P. Nikander, J. Arkko, and B. Ohlman. Host Identity Indirection Infrastructure (Hi3). In *Proc. of The Second Swedish National Computer Networking Workshop 2004 (SNCNW2004)*, Karlstad, Sweden, Nov. 2004.

[10] P. Nikander and J. Laganier. Host Identity Protocol (HIP) Domain Name System (DNS) extensions: draft-ietf-hip-dns-01.txt, Feb. 2005. Work in progress. Expires in August, 2005.

[11] P. Nikander, J. Ylitalo, and J. Wall. Integrating Security, Mobility, and Multi-homing in a HIP way. In *Proc. of Network and Distributed Systems Security Symposium (NDSS'03)*, San Diego, CA, USA, Feb. 2003. Internet Society.

[12] E. Nordmark. *Multi6 Application Referral Issues.* Internet Engineering Task Force, Jan. 2005. Internet draft, work in progress.

[13] J. Postel and J. Reynolds. *RFC959: File Transfer Protocol (FTP).* Internet Engineering Task Force, Oct. 1985.

[14] M. A. C. Snoeren. *A Session-based Architecture for Internet Mobility.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Feb. 2003.

[15] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proc. of ACM SIGCOMM'02*, Pittsburgh, PA, USA, Aug. 2002.

[16] J. Ylitalo and P. Nikander. BLIND: A complete identity protection framework for end-points. In *Proc. of the Twelfth International Workshop on Security Protocols*, Apr. 2004.

[17] S. Zhuang, K. Lai, I. Stoica, R. Katz, and S. Shenker. Host Mobility using an Internet Indirection Infrastructure. Technical report, University of California at Berkeley, 2002.

Publication III

Miika Komu, Sasu Tarkoma and Andrey Lukyanenko. Mitigation of Unsolicited Traffic Across Domains with Host Identities and Puzzles. In *15th Nordic Conference on Secure IT Systems (NordSec 2010)*, Springer Lecture Notes in Computer Science, Volume 7127, pp. 33-48, ISBN 978-3-642-27936-2, Espoo, Finland, October 2010.

Mitigation of Unsolicited Traffic across Domains with Host Identities and Puzzles

Miika Komu[1], Sasu Tarkoma[2], and Andrey Lukyanenko[1]

[1] Aalto University
[2] University of Helsinki

Abstract. In this paper, we present a general host identity-based technique for mitigating unsolicited traffic across different domains. We propose to tackle unwanted traffic by using a cross-layer technique based on the Host Identity Protocol (HIP). HIP authenticates traffic between two communicating end-points and its computational puzzle introduces a cost to misbehaving hosts. We present a theoretical framework for investigating scalability and effectiveness of the proposal, and also describe practical experiences with a HIP implementation. We focus on email spam prevention as our use case and how to integrate HIP into SMTP server software. The analytical investigation indicates that this mechanism may be used to effectively throttle spam by selecting a reasonably complex puzzle.

1 Introduction

One challenge with the current Internet architecture is that it costs very little to send packets. Indeed, many proposals attempt to introduce a cost to unwanted messages and sessions in order to cripple spammers' and malicious entities' ability to send unsolicited traffic. From the network administration viewpoint, spam and DoS traffic comes in two flavors, *inbound* and *outbound* traffic. Inbound traffic originates from a foreign network and outbound traffic is sent to a foreign network. Typically, spam and packet floods originate from networks infested with zombie machines. A *zombie* machine is a host that has been taken over by spammers or persons working for spammers, e.g., using Trojans or viruses.

We address the problem of unsolicited network traffic. We use two properties unique to the *Host Identity Protocol (HIP)* protocol: First, hosts are authenticated with their public keys which can be used for identifying well-behaving SMTP servers. Second, a computational puzzle introduces a cost to misbehaving hosts. Our approach has a *cross-layer* nature because a lower-layer security protocol is used to the benefit of higher-layer protocols.

2 Host Identity Protocol

The Host Identity Protocol (HIP) [9] addresses mobility, multi-homing, and security issues in the current Internet architecture. HIP requires a new layer in the networking stack, logically located between the network and transport layers, and provides a new, cryptographic namespace. HIP is based on *identifier-locator split* which separates the *identifier* and *locator* of an Internet host. The identifier uniquely names the host in a cryptographic namespace, and the locator defines a topological location of the node. Communication end points are identified using public cryptographic keys instead of IP addresses. The public keys used for HIP are called *Host Identifiers (HIs)* and each host generates at least one HI for itself.

The HIs can be published as separate HIP-specific records in the DNS [11]. Legacy applications can use HIP transparently without any changes. Typically, the application calls the system resolver to query the DNS to map the host name to its corresponding address. If a HIP record for the host name does not exist, the resolver returns a routable IPv4 or IPv6 address. Otherwise, the resolver returns a Host Identifier fitted into an IPv4 or IPv6 address. *Local-Scope Identifier (LSI)* is a virtual IPv4 address assigned locally by the host and it refers to the corresponding HI. *Host Identity Tag (HIT)* is an IPv6 address derived directly from the HI by hashing and concatenation. An LSI is valid only in the local context of the host whereas a HIT is statistically globally unique.

When an application uses HIP-based identifiers for transport-layer communications, the underlying HIP layer is invoked to authenticate the communication end-points. This process is called the *base exchange*, during which the end points authenticate to each other using their public keys. The host starting the base exchange, the initiator, is typically a client, and the other host, the responder, is typically a server. During the base exchange, the initiator has to use a number of CPU cycles to solve a computational puzzle. The responder can increase the computational difficulty of the puzzle to throttle new incoming HIP sessions. Upon successful completion, both end-hosts create a session state called *HIP association.*

The base exchange negotiates an end-to-end tunnel to encapsulate the consecutive transport-layer traffic between the two communicating end-hosts. The tunnel is required because routers would otherwise discard traffic using virtual, non-routable identifiers. Optionally, the tunnel also protects transport-layer traffic using a shared key generated during the base exchange. By default, the tunnel is based on IPsec [7] but S-RTP [14] can be used as well. It should be noted that a single tunnel can encompass multiple transport-layer connections.

With HIP, transport-layer connections become more resilient against IP address changes because the application and transport layers are bound to the location-independent virtual identifiers, HITs or LSIs. The HIP layer handles IP-address changes transparently from the upper layer using the *UPDATE* procedure [10]. In the first step of the procedure, the end host sends all of its locators to its connected peers. Then, the peers initiate so called *return routability test* to protect against packet-replay attacks, i.e., to make sure that the peer locator is correct. In the test, each node sends a nonce addressed to each of the received peer locators. The peer completes the test by signing each nonce and echoing it back to the corresponding peer. Only after the routability test is successfully completed, the peer can start using the locator for HIP-related communications.

HIP sessions can be closed using the *CLOSE* [9] mechanism. It is consist of two packets, in which one of the peer sends a CLOSE message to the other, which then acknowledges the operation using CLOSE-ACK. After this, all state is removed and the tunnel is torn down on both sides.

HIP employs *rendezvous servers* [5] to address the *double jump* problem. This occurs when two connected HIP hosts lose contact with each other when they are relocated simultaneously to new networks. The rendezvous server has a stable IP address and offers a stable contact point for the end hosts to reach each other.

The computational puzzles of HIP [1] play a major role in this paper and have been investigated by others as well. Beal et al. [3] developed a mathematical model to evaluate the usefulness of the HIP puzzle under steady-state DDoS attacks. They also stated that the difficulty of the DoS-protection puzzle should not be too high because otherwise an attacker can just choose a cheaper method such as simple flooding of the network. Tritilanunt et al. [13] explored HIP puzzles further with multiple adversary models and variable difficulties. They also noticed that solving of HIP puzzles can be distributed and a non-distributable puzzle algorithm would provide more resilience against DDoS. Our work differs from Beal et al. and Tritilanunt et al. because our use case is spam rather than DDoS and our approach is based on cross-layer integration.

3 System Model

The basic idea is to assign each node in the network with an identity based on a public key. The hosts may generate their private keys by themselves, or a third party can assign them. Computational puzzles are a well-known technique for spam prevention [4,2,6] but are typically used on a per message basis. In our case, puzzles are applied to each pair of Host Identifiers. The difficulty of the puzzle is varied based on the amount of unwanted traffic encountered.

Our example use case for the technique is spam prevention. Typical spam prevention techniques are applied in a sequence starting from black, white or gray listing techniques and sender identification, and ending in content filtering. Our approach involves a similar sequence of spam testing but relies on the identity of the sender rather than its IP address.

3.1 Basic Architecture for Spam Mitigation

In the email systems deployed in the Internet, there are *outbound email servers* which are used for sending email using SMTP. Typically, the users access them either directly or indirectly with a web-based email client. Usually users are authenticated to these services with user names and passwords. In many cases, direct access to outbound SMTP servers is restricted to the local network as a countermeasure against spam. However, spam is still a nuisance and there are networks which still allow sending of spam. In this paper, we use the term *spam* *relay* for a malign or compromised outbound email server that allows sending spam, and the term *legitimate relay* for a well-behaving outbound email server.

Correspondingly, *inbound email servers* process incoming emails arriving from outbound emails servers. Users access these servers either indirectly via web interfaces or directly with protocols such as POP or IMAP. Typically, the inbound email server tags or drops spam messages and also the email client of the user filters spam messages.

Our idea in a nutshell is to require a HIP session with an SMTP server before it will deliver any email. The sender has to solve a computational puzzle from the server to establish the session. If the sender sends spam, the server ends the HIP session after a certain spam threshold is met. To continue sending spam, the sender has to create a new session, but this time it will receive a more difficult puzzle from the server.

The proposed architecture follows the existing SMTP architecture but requires some changes. First, the inbound and outbound SMTP servers have to be HIP capable. Second, we assume the spam filter of the inbound server is modified to control the puzzle difficulty. Third, we assume the inbound SMTP servers publish their Host Identifiers in the DNS.

3.2 Deployment Considerations

A practical limitation in our approach is that HIP itself is not widely deployed. Even though we compare the HIP-based approach to the current situation later in this paper, the benefits of our design can be harnessed to their full extent only when HIP, or a similar protocol, has been deployed more widely in the Internet. Alternatively, our design could be applied to some other system with built-in HIP support such as HIP-enabled P2P-SIP [8].

We assume that Host Identities are published in the DNS which requires some additional configuration of the DNS and also SMTP servers. However, based on our operational experience with HIP, this can be accomplished in a backward-compatible way and also deployed incrementally. First, the DNS records do not interfere with HIP-incapable legacy hosts because the records are new records and thus not utilized by the legacy hosts at all. Second, *bind*, a popular DNS server software, does not require any modifications to its sources in order to support DNS records for HIP. Third, SMTP servers can utilize a local DNS proxy [12] to support transparent lookup of HIP records from the DNS.

3.3 Pushing Puzzles to Spam Relays

We considered two implementation alternatives for pushing puzzle computation cost to spam relays. In the first alternative, the UPDATE messages could be used to request a solution to a new puzzle. However, this is unsupported by the current HIP standards at the moment. In the second alternative, which was chosen for the implementation, inbound servers emulate puzzle renewal by terminating the underlying HIP session. The termination is necessary because current HIP specifications allow puzzles only in the initial handshake. When the spam relay reconnects using HIP, a more difficult puzzle will be issued by the server.

3.4 Re-generating a Host Identity

One obvious way to circumvent the proposed mechanism is to change to a new Host Identity after the server closes the connection and increases the puzzle difficulty. Fortunately, creating Host Identities is comparable in cost to solving puzzles, which can discourage rapid identity changes. In addition, non-zero puzzle computation time in the initial session further discourages creation of new identities.

3.5 Switching Identities

It is reasonable to expect that a server relaying spam is able to generate new host identities. Let C_K denote a key-pair generation time and C_N the cost of making the public key and the corresponding IP address available in a lookup service. We expect a spam relay to reuse its current identity as long as the following equation holds:

$$C_j < C_K + C_N + C_0, \tag{1}$$

where C_j is the puzzle computation time of the jth connection attempt. In other words, the spam relay continuously evaluates whether or not to switch to a new identity. If the next puzzle cost is greater than the initial cost, the spam relay has motivation to switch the identity. We note that the spam relay may devise an optimal strategy if the cost distribution is known.

When the puzzle cost is static, there is no incentive for the spam relay to change its identity unless blacklisted because the cost would be greater due to the C_K and C_N terms. For a dynamic cost, the spam relay is expected to change identities when the cost of a new identity and a new connection is less than maintaining the current identity and existing connection. For a DNS-based solution, the C_N term has a high value because DNS updates are slow to take effect.

Our proposed approach addresses identity-switching attacks using three basic mechanisms. First, a node must authenticate itself. This means that the node must be able to verify its identity using the corresponding private key. This does not prevent the node from using multiple identities or changing its identity, but ensures that the key pair exists. Second, a node must solve a computational puzzle before any messages are transported.

Third, a level of control is introduced by the logically centralized lookup service. The DNS maps host names to identities and IP addresses. A node must have a record in the lookup service. The limitation of this approach that control is introduced after something bad (e.g. spam) has already happened. The bad reputation of malicious nodes can be spread with, for example, DNSBL lookups performed by SMTP servers.

Nevertheless, identity switching could used to reduce the proposed system and, therefore, we have taken it into account in the cost model analysis of the next section.

4 Cost Model

In this section, we present an analytical cost model for the proposed identity-based unsolicited traffic prevention mechanism. We analyze the performance of the proposed mechanism when a number of legitimate senders and spam relays send email to an inbound SMTP server. It should be noted that our model excludes puzzle delegation in the case of multiple consecutive relays because it is not advocated by the HIP specifications.

4.1 Preliminaries

Let us consider a set of N_L legitimate email relays and N_S spam relays. Each legitimate relay sends messages at the rate of λ_L messages per second and each spam relay at λ_S. We assume that the inbound email server has a spam filtering component. It has a false negative of probability α, which refers to undetected spam. Thus, $(1-\alpha)$ gives the probability for detecting a spam message. The filter has also a false positive of probability β, which denotes good emails classified as spam. Even though the inbound server could reject or contain the spam, we assume the server just tags the message as spam and passes it forward.

An inbound SMTP server has a spam threshold κ given as the number of forwarded spam messages before it closes the corresponding HIP session. After the session is closed, the outbound email relay has to reopen it. Let the number of reopened sessions be ξ in case of spam relays, and η in case of legitimate email relays. The base exchange has an associated processing cost for the SMTP source, T_{BE}, given in seconds. This processing cost includes also the time spent in solving the puzzle. Let T_M denote the forwarding cost of a message. The finite time interval T, for which we inspect the system, is expressed in seconds.

4.2 Cost Model

To demonstrate scalability, we derive the equation for the load of the inbound SMTP server with and without HIP. The server load is determined by the number of HIP sessions at the server and the number of email messages forwarded. Without HIP, the email processing cost in seconds at the server is

$$R_N = T \cdot T_M \cdot (N_L \cdot \lambda_L + N_S \cdot \lambda_S). \tag{2}$$

In case of HIP, let us define the accumulated puzzle computation time function $G(\xi) = \sum_{i=0}^{\xi} C_i$. First, we consider the case with constant puzzle computation time that is independent of number of session resets, i.e. $C_1 = C_2 = \ldots C_N = T_{BE}$, and $G(\xi) = \xi \cdot T_{BE}$.

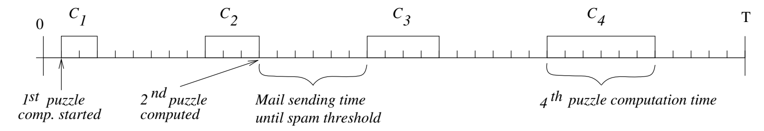

0
C_1
C_2
C_3
C_4
T
1st puzzle comp. started
2nd puzzle computed
Mail sending time until spam threshold
4th puzzle computation time

Fig. 1. Division of system inspection time (T) into puzzle re-computation and mail delivery stages with different puzzle computation times c_i, where i is the number of session resets. All $C_i = T_{BE}$ if the puzzle computation time is constant.

Next, we derive the number of HIP sessions due to session resets caused by spam under the condition of identical puzzle computation time. The following equation presents the number of session resets for a single spam relay:

$$\xi = \frac{(T - T_{BE} \cdot \xi)\lambda_S(1-\alpha)}{\kappa} \tag{3}$$

From equation 3, we can deduce that

$$\xi = \frac{T \cdot \lambda_S \cdot (1-\alpha)}{\kappa + \lambda_S \cdot (1-\alpha) \cdot T_{BE}}. \tag{4}$$

The equation for the number of HIP sessions η needed by the legitimate SMTP relays is similar to equation 4, but the false positive rate β is used instead of $(1-\alpha)$, and correspondingly λ_L is used instead of λ_S. We assume that legitimate relays do not send significant amount of spam so that only false positives need to be considered. The cost to a paying customer is given by η, and ξ is the cost to a spam relay. Given a small false positive probability, η is small. Therefore, the mechanism is not harmful to paying customers.

Next, we derive the equation describing the HIP load of the inbound server R_H consisting of both legitimate and spam messages:

$$R_H = N_L \cdot (\eta \cdot T_{BE} + T \cdot \lambda_L \cdot T_M) + N_S \cdot (\xi \cdot T_{BE} + T_S \cdot \lambda_S \cdot T_M), \tag{5}$$

The equation can be simplified by substituting the total time used for sending spam messages T_S with $T - T_{BE} \cdot \xi$ and by applying equation 2:

$$R_H = R_N - T_M \cdot T_{BE} \cdot (\lambda_S \cdot N_S \cdot \xi) + T_{BE} \cdot (N_L \cdot \eta + N_S \cdot \xi) \tag{6}$$

We assume β is small and, therefore, we used T instead of $T - T_{BE} \cdot \eta$ (with η denoting the number of session resets for a legitimate host). To evaluate the effectiveness of the HIP-based solution against a solution without HIP, we define ratio φ as:

$$\varphi = \frac{R_H}{R_N}. \tag{7}$$

Now, consider the case when puzzle computation time is not constant, but rather a function of the number of session attempts. This has to be reflected in equation 4, which becomes

$$\kappa \cdot \xi + G(\xi)\lambda_S(1-\alpha) - T \cdot \lambda_S(1-\alpha) = 0. \tag{8}$$

The equation can be solved using numerical iteration.

4.3 A Comparison of HIP with Constant Puzzle Cost to the Scenario without HIP

For numerical examples, we use HIP base exchange measurements obtained from an experimental setup described further in Section 5. We plot the ratio of non-HIP versus HIP approaches φ shown in equation 7. The HIP base exchange with a 10-bit puzzle was measured to take 0.215 s of HIP responder's time and 0.216 s for the initiator. We note that our analysis excludes the impact of parallel network and host processing. The email forwarding overhead without HIP is set to 0.01 seconds. We assume that the false negative probability of the server is 1/3 and the false positive probability is $1/10^4$. In other words, 2/3 of spam messages will be correctly detected as spam, and good messages are rarely classified as spam. Let N_L be 10^4, N_S be 100, $\lambda_L = 1/360$, and $\lambda_S = 10$. The time-period T for the analysis is 24 hours.

Figure 2 presents the ratio of HIP versus non-HIP computational cost as a function of the puzzle computation time. Both x and y axes are logarithmic. Ratio in the figures denotes φ, the ratio of the HIP and non-HIP capable mechanisms. The point at which the HIP mechanism has less overhead is approximately at 2 seconds. This means that the proposed HIP mechanism becomes superior to the constant non-HIP benchmark case with an 2-second or greater puzzle computation time. Naturally, this point depends on the selection of the values for the parameters.

As the spam relay sending rate increases, the HIP spam prevention mechanism becomes considerably better than the non-HIP benchmark case. With low spam rates, HIP sessions are reset seldomly and spam flows mostly through. When the spam rate increases, the spam relay spends more time on puzzle computation and the spam forwarding rate decreases. Then, the performance of the proposed HIP mechanism improves in comparison to the non-HIP benchmark case.

4.4 A Comparison of HIP with Exponential Puzzle Cost the Scenario without HIP

We also analyze the scenario where the puzzle cost grows exponentially for each new session. The parameters are the same as before, but the computation time of the puzzle grows exponentially with the puzzle difficulty. Moreover, we introduce a *cut-off point* after which the puzzle difficulty does not increase anymore. After the number of sessions reaches the cut-off point, the computation time of the puzzle (and the number of bits) remains at the current level. As an example, given a cut-off point of 23 and an initial puzzle size of 20 bits for the first throttled session, spam relays experience puzzle sizes $\{20, 21, 22, 23, 23, \dots\}$ as they reconnect.

Figure 3 presents the effect of the exponential base exchange time with a varying cut-off point. The y axis is logarithmic. The figure shows that the proposed mechanism performs considerably better than the non-HIP benchmark with a cut-off point of 22 or greater.

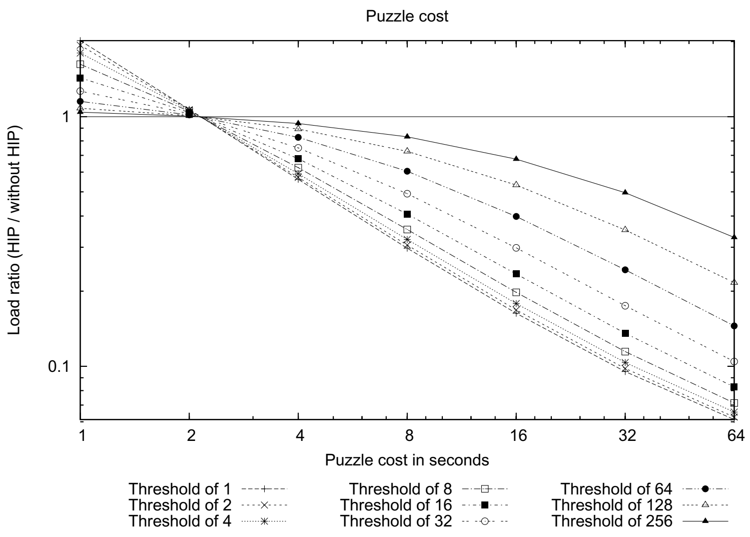


Fig. 2. Fixed-cost puzzles with different spam threshold κ

Now, we have compared HIP with both constant and variable-sized puzzles to the benchmark scenario without HIP. In the next sections, we focus on the identity-switching attacks (without cut-off points) against the proposed HIP-based architecture.

4.5 Optimal Strategies for a Spam Relay

Directly from equation 8, we know that

$$\xi \cdot \frac{\kappa}{\lambda_S \cdot (1-\alpha)} + G(\xi) = T. \tag{9}$$

This means that, for the entire time during which a server relays spam, it splits its performance into ξ steps (one step is one session reset). To contact the inbound server, the spam relay spends $G(\xi)$ time for all puzzle computations, and during every step it sends exactly κ messages and each step consumes $\frac{\kappa}{\lambda_S(1-\alpha)}$ time.

The inbound server chooses the form of the function G, while a spam relay selects the number of session resets to tolerate, ξ. Here, we consider first a naive strategy for the spam relay. It chooses G based on the number of messages to send and does not try to whitewash its own history at the inbound server (i.e. by changing its identity according to equation 1). Under such an assumption, the spam relay has to optimize (maximize) a function of the following form:

$$p_Z \cdot \kappa \cdot \xi - c_Z \cdot G(\xi), \tag{10}$$

where p_Z is the profit for one delivered message and c_Z is the payment for the puzzle computation time. The strategy for the spam relay is to select

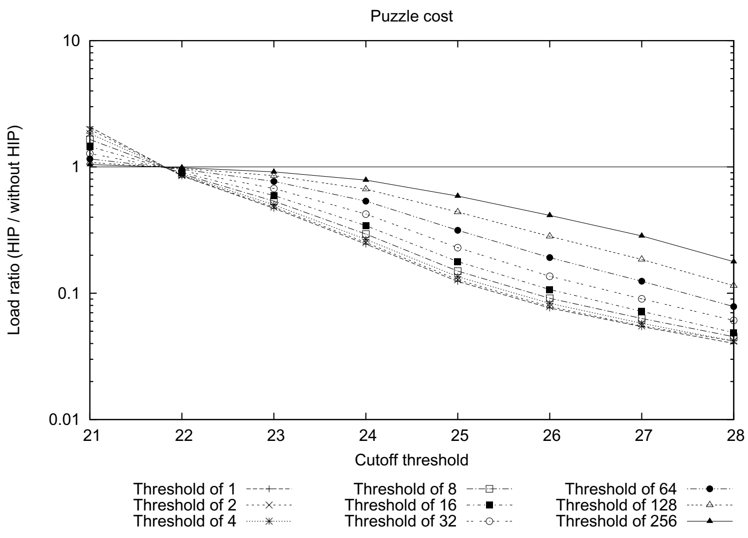


Fig. 3. Variable-sized puzzles with initial puzzle size of 20 and different cut-off points

the number of rounds for which it would like to send κ-sets of messages and the number of rounds to recompute puzzles. Let this value be ξ.

Note that if puzzle difficulty is constant (i.e. $G(\xi) = T_{BE} \cdot \xi$), then solution is one of the boundary cases

$$\xi = \begin{cases} 0, & \text{if } p_Z\kappa \leq c_Z \cdot T_{BE}, \\ \infty, & \text{if } p_Z\kappa > c_Z \cdot T_{BE}, \end{cases} \tag{11}$$

More important is the case when the puzzle computation time is changing. Let the puzzle complexity growth be exponential compared to the increase of puzzle difficulty. Consider that the puzzle computation time on every reset has an exponential form of $C_i = aq^i + b$, then by definition

$$G(\xi) = \sum_{i=0}^{\xi}(aq^i + b) = a\frac{q^{\xi+1} - 1}{q - 1} + b = \frac{aq}{q-1}q^{\xi} + b - \frac{a}{q-1}. \tag{12}$$

Let us generalize this function as $G(\xi) = kg^{\xi} + s$, where g is an exponential growth parameter, s is initial shift, and k is the coefficient.

Now, a spam relay has to maximize the function

$$p_Z \cdot \xi \cdot \kappa - c_Z \cdot (k \cdot g^{\xi} + s). \tag{13}$$

Let us find the points where the derivative of this function with respect to ξ is zero:

$$p_Z \cdot \kappa - c_Z \cdot k \cdot \ln g \cdot g^{\xi} = 0. \tag{14}$$

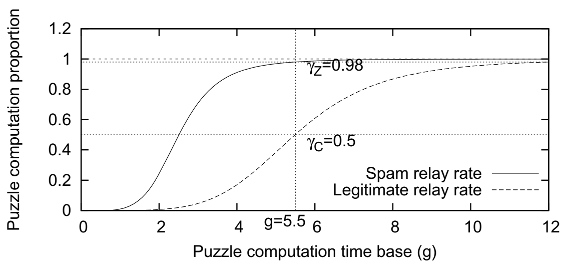


Fig. 4. An example plot to illustrate the proportion of time used by a legitimate and spam relay for puzzle computation

Thus, the maximum point is

$$\xi^* = \log_g \frac{p_Z \cdot \kappa}{c_Z \cdot k \cdot \ln g}. \tag{15}$$

4.6 Optimal Strategies for an Inbound Server

The previous section suggests an optimal strategy for a spam relay under the assumption that there is a payment involved in sending of spam. Otherwise, infinite number of messages would be the optimal strategy for the spam relay. In this section, we have a look at the situation from the view point of an inbound server.

First of all, the main goal for the inbound server is to slow down the flood of spam. It may be formulated in terms of the portion of time which spam relays spend for the puzzle computation time, compared to the overall time. Here, we assume that the inbound relay knows the number of HIP session resets during which spammer reuses its current identity according to equation 1. As previously, let it be ξ. To process ξ resets, a spam relay has to waste $G(\xi)$ of its own time for puzzle computation. The overall time, which it may use for message delivery, we also define as a function of ξ. Thus, the definition of the overall time follows from equation 9

$$T(\xi) = \xi \frac{\kappa}{d} + G(\xi), \tag{16}$$

where d is equal to $\lambda_Z(1-\alpha)$ in case of a spam relay, and is equal to $\lambda_C\beta$ in case of a legitimate email relay. We assume that an inbound server classifies (or receives classification) with relatively good accuracy and, hence, $1-\alpha$ is considerably higher than β.

Then, the proportion of time used for puzzle computation by spam relays (on left side) and legitimate email relays (on the right side) can be calculated as

$$\frac{G(\xi)}{G(\xi) + \frac{\kappa \cdot \xi}{\lambda_Z \cdot (1-\alpha)}}, \qquad \frac{G(\xi)}{G(\xi) + \frac{\kappa \cdot \xi}{\lambda_C \cdot \beta}}. \tag{17}$$

The inbound server has control over variables k, g, s of function $G(\xi) = kg^{\xi} + s$. For simplicity, let k and s be constants because the most relevant variable is the growth base g for the puzzle computation time. The values grow as a function of the parameter g. The function results in values ranging from 0 to 1, where 0 means that the time spent for the puzzle computation is negligible, while 1 means that the puzzle computation takes all of the time.

For the functions (17), the objective of the inbound server is to maximize the time spam relays spend on computing puzzles. Correspondingly, the inbound server should minimize this time for legitimate relays. These are somewhat contradictory conditions because $\alpha < 1$ and $\beta > 0$. Therefore, punishment for possible spam relays affects also legitimate relays.

To overcome this dilemma, we introduce a new constant γ: $0 \leq \gamma \leq 1$, which we select as the maximum value for the possibly legitimate client computation rate, i.e.

$$\frac{G(\xi)}{G(\xi) + \frac{\kappa \cdot \xi}{\lambda_C \cdot \beta}} \leq \gamma, \tag{18}$$

where γ defines the portion of the overall time which a possibly legitimate client spends for puzzle computations. From the inequality 18 it follows, that

$$g \leq \left(\frac{\gamma \cdot (\kappa \cdot \xi + s \cdot \lambda_C \cdot \beta)}{k \cdot \lambda_C \cdot \beta \cdot (1 - \gamma)} \right)^{\frac{1}{\xi}}. \tag{19}$$

On the other hand, the inbound server should maximize puzzle computation rate for possible spam relays (the left function in equation 17, which grows exponentially towards 1 as a function of g). The optimal strategy for the server is

$$g^* = \left(\frac{\gamma \cdot (\kappa \cdot \xi + s \cdot \lambda_C \cdot \beta)}{k \cdot \lambda_C \cdot \beta \cdot (1 - \gamma)} \right)^{\frac{1}{\xi}}. \tag{20}$$

The optimal strategy both for a spam relay, $\xi^*(g)$, as shown in equation 15, and for an inbound server, $g^*(\xi)$, as shown in equation 20, results in an equilibrium point (ξ^*, g^*) in terms of game theory.

The optimal strategies are illustrated in figure 4. For the legitimate relay, the bound for the computation rate is fixed as $\gamma_C = 0.5$ The set of parameters is assigned as $\alpha = 0.5$, $\beta = 0.01$, $\kappa = 100$, $\lambda_C = \lambda_Z = 10$, and we assume that the number of session resets is 5 ($\xi = 5$). Under such parameters, the legitimate relay has $g \approx 5.5$. The resulting puzzle computation for a possible spam relay is $\gamma_Z = 0.98$. In other words, the spam relay spends 0.98 of its time for puzzle computations whereas the legitimate relay spends half of its time. As g grows, both parties are eventually spending all of their time for puzzle computation. Thus, it is a local policy for the inbound server to decide a "good" value for g in terms of how much legitimate servers can be throttled with puzzles. For low spam rates, the value can be small but, with high spam rates, the server may increase the value at the cost of throttling also legitimate relay servers.

5 Experimental Evaluation

In this section, we describe how we integrated puzzle control to an inbound SMTP server and its spam filtering system. We show some measurements with variable-sized puzzles and compare this against identity-generation costs to give some engineering guidance against identity-switching attacks. The source code for HIP for Linux and the spam extensions are available as open source [1]. It should be noted that evaluation the mathematical models presented in section 4 e.g. with network simulators is future work.

5.1 Setup

The experimented environment consisted of two low-end commodity computers with the Linux Debian distribution and HIP for Linux (HIPL) [12] implementation. One computer served as a sending SMTP relay (1.60GHz Pentium M) and the other represented a receiving SMTP server (Pentium 4 CPU 3.00GHz). The receiving server detects the spam messages and closes the HIP session when a threshold is reached for the session. The inbound server was configured not to reject any email. We were mostly interested in software changes required to deploy HIP in SMTP servers and in the effects of increasing the puzzle size.

5.2 Results

We implemented the spam throttling mechanism successfully by using unmodified sendmail. We turned on the IPv6 option in the configuration of sendmail in order to use HITs.

The receiving SMTP server was equipped with a modified version of *SpamAssassin Milter*. The changes were straightforward to implement. The milter increased the puzzle size by one for every κ spam message detected and closed the HIP session to induce a new base exchange. The puzzle computation time grew exponentially with the size of the puzzle and the spam sender was throttled, as expected, by the mechanism.

We faced some implementation challenges during the experimentation. Firstly, sendmail queues the email messages and this makes it difficult to provide measurements from the spam filtering process itself. Secondly, if the session with the SMTP server is lost temporary, for example, because the HIP association are is closed, e-mails can accumulate in the queue for an extended time. Thirdly, when sending excessive amounts of email, the built-in connection throttling mechanism in sendmail takes over and queues the emails for long periods. However, sendmail's queuing process was robust and eventually emptied the queue successfully.

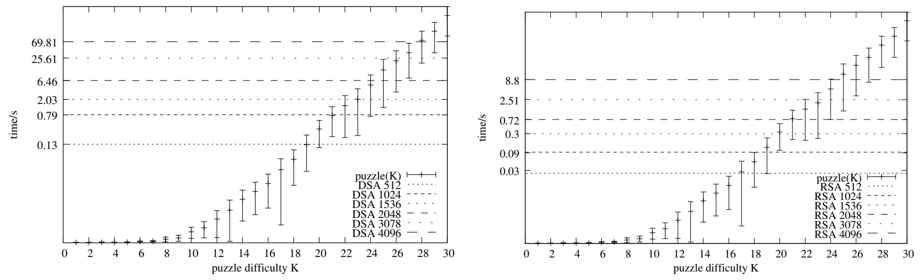
One challenge with proof-of-work techniques is that there are many different devices on the network and their computing capabilities vary. By default, the puzzle difficulty is zero in HIPL. A puzzle with difficulty of 25 bits took 12.4 s on

[1] `https://launchpad.net/hipl/`

average on the low-end machine used in the performance tests. The time was 4.4 seconds on a more recent CPU (Intel Core 2, 2.3 Mhz) on a single CPU core. The puzzle algorithm used in HIP does not prevent parallel computation. Thus, the computation time could be decreased by fully utilizing multi-core architectures.

For identity changing attacks, the strategy should also take into account the public key algorithm. RSA keys can be created faster than DSA keys with a corresponding size. As a consequence, the responder should give initiators that use RSA public keys more difficult puzzles than initiators with DSA keys. Further, it should be noted that creation of insecure, albeit perfectly valid keys, can be faster than creation of secure ones.

Figures 5(a) and 5(b) contrast secure key-pair generation time (horizontal lines) with puzzle solving time (vertical lines). It should be noticed that the y-axis is logarithmic. From the figures, it can be observed that the puzzling solving time is, as expected, exponential with the number of bits used in the puzzle difficulty. The standard deviation grows as puzzle difficulty is increased. In addition, the time to generate DSA key-pairs is considerably higher than RSA. On the average, the creation of a 2048-bit DSA key pair took 6.46 seconds and this was equal to the solving time of a 24-bit puzzle. With RSA, creation of a 2048-bit key pair took 0.72 seconds which corresponded to a 21-bit puzzle. This indicates that the key-generation algorithm and key length need to be taken into account when deciding the initial puzzle size to discourage identity-switching attacks.



(a) Puzzle solving vs. DSA key generation (b) Puzzle solving vs. RSA key generation

Fig. 5. Puzzle computation time results

6 Conclusions

In this paper, we proposed a cross-layer identity solution for mitigating unsolicited traffic between administrative domains. The proposed architecture primarily concentrates on inbound session control but is applicable also to the outbound direction as well. As an example application of the system, we focused on email spam prevention.

The Host Identity Protocol introduces a public key for the hosts. They key can be used for identifying well-behaving SMTP servers. The proposed approach introduces a cost to sending spam using the computational puzzles in HIP. Large-scale changes to the SMTP architecture are not required because HIP is backwards compatible. However, a practical limitation of the approach is that it requires wide-scale adoption of HIP as a signaling protocol and requires integration of HIP puzzle control to inbound email servers.

We presented a formal cost model that considered static and exponential base exchange puzzle costs. The analytical investigation indicates that the proposed spam prevention mechanism is able to mitigate unwanted traffic given a set of reasonable parameters. We used parameter values based on experimental results for server-side cost of HIP and the puzzle computation time. A spam mitigation approach based on HIP puzzles caused less load at the email server than an approach that was not using HIP.

The exponential cost of the puzzle introduces more work for email servers relaying spam. However, it also results in an incentive for the spammer to switch its identity when it is throttled with more difficult computational puzzles. We identified this as a potential weakness of the proposed system and analyzed this from the viewpoint of the spammer and the email server. As a theoretical result, we provided a method for the server to choose an optimal strategy against identity switching. When choosing a strategy, it should be noted that increasing puzzle costs for spammers also increase costs for legitimate hosts.

We implemented a simple prototype of the system based on a popular email server, sendmail. We integrated throttling support for HIP puzzles with minimal changes to SpamAssassin, a popular spam filtering software. We reported the practical experiences of running such a system and showed real-world measurements with HIP puzzles.

While the simple prototype was a success, we observed that the use of computational puzzles with email relays is challenging. Malicious hosts can overwhelm and exhaust the resources of a relay unless preventive measures are taken. Potential solutions to this include refusal to solve large puzzles for hosts, message rejection, and blacklisting. More work with simulation or larger test beds is needed to establish the efficacy of the proposed cross-layer system and to validate our mathematical models.

Acknowledgements. We would like to thank the following people for providing valuable feedback for this paper: Jaakko Kangasharju, Teemu Koponen, Kristian Slavov, Antti Järvinen, Oleg Ponomarev, Andrei Gurtov and Tuomas Aura. This work was supported by Tekes InfraHIP project and the Academy of Finland, grant numbers 135230, 122329 and 135230.

References

1. Aura, T., Nikander, P., Leiwo, J.: Dos-Resistant Authentication with Client Puzzles. In: Christianson, B., Crispo, B., Malcolm, J.A., Roe, M. (eds.) Security Protocols 2000. LNCS, vol. 2133, pp. 170–177. Springer, Heidelberg (2001)

2. Back, A.: Hashcash (May 1997), `http://www.cypherspace.org/hashcash/`
3. Beal, J., Shepard, T.: Deamplification of DoS Attacks via Puzzles (October 2004), `http://web.mit.edu/jakebeal/www/Unpublished/puzzle.pdf`
4. Dwork, C., Naor, M.: Pricing via Processing or Combatting Junk Mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993)
5. Eggert, L., Laganier, J.: Host Identity Protocol (HIP) Rendezvous Extension. IETF (April 2008), Experimental RFC
6. Goodman, J., Rounthwaite, R.: SmartProof. Microsoft (2005), `http://research.microsoft.com/en-us/um/people/joshuago/smartproof.pdf`
7. Jokela, P., Moskowitz, R., Nikander, P.: RFC5202: Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP) Internet Engineering Task Force (April 2008), `http://www.ietf.org/rfc/rfc5202.txt`
8. Keränen, A., Camarillo, G., Mäenpää, J.: Host Identity Protocol-Based Overlay Networking Environment (HIP BONE) Instance Specification for REsource LOcation And Discovery (RELOAD). Internet Engineering Task Force (July 2010) (internet draft, work in progress)
9. Moskowitz, R., Nikander, P., Jokela, P., Henderson, T.: RFC5201: Host Identity Protocol. Internet Engineering Task Force (April 2008); Experimental RFC
10. Nikander, P., Henderson, T., Vogt, C., Arkko, J.: End-Host Mobility and Multihoming with the Host Identity Protocol. Internet Engineering Task Force (April 2008); Experimental RFC
11. Nikander, P., Laganier, J.: Host Identity Protocol (HIP) Domain Name System (DNS) Extension. IETF (April 2008); Experimental RFC
12. Pathak, A., Komu, M., Gurtov, A.: Host Identity Protocol for Linux. Linux Journal (November 2009), `http://www.linuxjournal.com/article/9129`
13. Tritilanunt, S., Boyd, C., Foo, E., Nieto, J.M.G.: Examining the DoS Resistance of HIP. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4277, pp. 616–625. Springer, Heidelberg (2006)
14. Tschofenig, H., Shanmugam, M., Muenz, F.: Using SRTP transport format with HIP. Internet Engineering Task Force (August 2006); expired Internet draft

Publication IV

Janne Lindqvist, Essi Vehmersalo, Miika Komu and Jukka Manner. Enterprise Network Packet Filtering for Mobile Cryptographic Identities. *International Journal of Handheld Computing Research (IJHCR),* Volume 1, Issue 1, pp. 79-94, ISSN 1947-9158, January 2010.

Enterprise Network Packet Filtering for Mobile Cryptographic Identities

Janne Lindqvist, Helsinki University of Technology, Finland

Essi Vehmersalo, Helsinki University of Technology, Finland

Miika Komu, Helsinki Insitute for Information Technology, Finland

Jukka Manner, Helsinki University of Technology, Finland

ABSTRACT

Firewalls are an essential component of the Internet and enterprise network security policy enforcement today. The configurations of enterprise firewalls are typically rather static. Even if client's IP addresses can be dynamically added to the packet filtering rules, the services allowed through the firewall are commonly still fixed. In this paper, we present a transparent firewall configuration solution based on mobile cryptographic identifiers of Host Identity Protocol (HIP). HIP allows a client to protect the data transfer with IPsec ESP, and supports dynamic address changes for mobile clients. The HIP-based firewall learns the identity of a client when it communicates with the server over HIP. The firewall configures the necessary rules based on HIP control messages passing through the firewall. The solution is secure and flexible, and introduces only minimal latency to the initial HIP connection establishment.

Keywords: *Firewalls, Network Security, Host Identity Protocol, HIP, Packet Filtering*

INTRODUCTION

Remote access to corporate service is very challenging to set up and configure in a secure way. The simplest way is on a per-service basis, by using HTTP and TLS and introducing a login function to public servers. Unfortunately, this leaves the server open for various attacks, e.g., DoS, since it must be open to any remote client regardless of the source IP address, and unlawful access attempts are caught very late in the login process.

To enable better security, and to disable most forms of attacks on the services, a VPN solution can be used, together with a tightly controlled firewall configuration. Through a VPN, corporate services are available for the client only after a successful VPN login transaction. Yet, also here the VPN server needs to be available to the public, and can be attacked. Moreover, configuration of the firewall is a

DOI: 10.4018/jhcr.2010090905

further cause of concern. First of all, setting up proper filtering rules for corporate firewalls is not a trivial task. Secondly, changing the connectivity provider results in network renumbering which further requires a full reconfiguration of the firewall. It is also also possible that the address of single VPN client changes due to device mobility or DHCP lease renewal. In such a case, the client has to reinitialize the VPN connection.

Two additional security concerns arise from the use of a typical VPN service. First, typically all the corporate services become available to the user when VPN gateway or firewall accepts a VPN connection. Then, a malicous user, virus or worm can try to mount an attack on any service of the corporation because VPNs do not offer protection against "internal" attacks. Second, the corporate services become vulnerable to attacks to "external" attacks if the device of a user is compromised. The attacker can route its own packets using the compromised device through the VPN tunnel to the corporate network. Hence, the attacker can practically mount any type of attack on the corporate services.

The second security concern was highlighted as part of the Microsoft Windows Vista routing compartments functionality, which was supposed to be included in the new operating system. The basic idea was that remote access from the user device is controlled per application, and not per host, making it impossible to route packets between interfaces, WLAN and VPN interfaces in our example. Yet, it is still not included, and one can only guess what the reasons are. Nevertheless, the security vulnerability still remains.

Firewalls are, unfortunately, a critical component of corporate and personal networks in the Internet today. Packet filtering is typically based on the 5-tuple of sender and receiver IP addresses and port numbers, and the transport protocol. Sophisticated firewalls can also filter based on the content of application layer protocols. Commonly, the filtering rules are quite static and constrained. The firewall passes only certain services and a known set of hosts through. In more dynamic networks, for example, offeering public or subscription-based WLAN access, or nomadic enterprise environments, the firewalls are controlled and rules set up based on some authentication exchange. Typically, a client is authenticated and authorized to use a WLAN service based on a web browser login application. If the login is successful, the firewall opens predefined services for the MAC and IP address of the client device. Only then the client can start access Internet to, for example, browse the web, or initiate VPN connections.

The current situation has at least four downsides. First, authentication for network access has a number of different implementation choices, which may or may not work with the device of the user, for example, on laptop computers, PDAs, or smart mobile phones. Second, the firewall allows the client to only use certain pre-defined services even when the client is authenticated successfully and authorized to use the Internet. It would be more useful to have a separate signaling protocol dynamically manage the filtering rules associated with a given authenticated client. Third, a third party can still listen to the network communications, collect varying information, and steal the identity of an authenticated client. Fourth, network renumbering becomes a problem, because all static rules on firewalls that are based on IP address must be changed when renumbering occurs. The same problem of updating firewall rules appears in access networks, where the IP address assigned to a client can change during the session, for example, in a mobile access network when the client performs a handover.

Setting up IP-based rules in a firewall to protect servers with roaming clients is difficult. Since the firewall cannot know the IP addresses of roaming employees, the rules that protect the network services must be quite liberal, or access is only possible through a separate VPN tunnel.

There seems to be a need for a remote access mechanism, that does not expose the corporate services to the security vulnerabilities described in this section, requires minimal configuration, is easy to set up and is operating-system independent. We have compared at a number of alternatives and eventually we started to investigate the use of the Host Identity Protocol (HIP) as such a mechanism.

In this paper, we present a firewall architecture that allows efficient, scalable and secure network packet filtering. Our solution solves all the problems discussed above. The firewall is based on the Host Identity Protocol (HIP) (Moskowitz & Nikander, 2006) and tracking the protocol control messages and IPsec ESP SPI values. Although the standard IPsec architecture could be used to implement firewalls (Aura et al., 2005), our architecture provides a simple way to centrally enforce security policies regardless of host IPsec security policies. The architecture also allows to group and present services with cryptographically tamper-proof identities.

Our solution primarily targets the initial connection set up. Once the HIP control message exchange has been authenticated, subsequent message filtering is simply based on the source and destination IP addresses and SPI numbers of ESP packets. Thus, the processing overhead only applies in the beginning of the data connection. Our measurements show that this processing adds a negligible overhead to the connection initiation. A modern firewall with our architecture can support thousands of connection initiations per second.

One of the key features of using HIP and a HIP-enabled firewall is that the administration of the network does not need to care about IP addresses. Thus, the network can perform renumbering, and support mobile users without changes in the firewall rules. Moreover, when the client is using HIP, it does not need to employ any additional protocol for authentication and firewall control, either inside or outside the enterprise network. Furthermore, the solution also allows encrypting the data transfer end-to-end.

The firewall solution introduced in this paper does not require Internet-wide deployment of HIP. An enterprise can deploy HIP gradually to harness the integrated security, mobility, and multihoming capabilities for employees. Services and clients that do not use HIP continue to operate with the old system. In summary, by using HIP to access a service, the client is able to perform simultaneously network access authentication and authorization, firewall control, data transfer protection, and mobility management.

One of the key strengths of our design is that HIP can be used in any kind of wired or wireless network, for example, xDSL, Ethernet, WLAN, WIMAX, 3G, and any technology beyond 3G. For example, a modern mobile intelligent device with multiple different wireless link technologies can use the same mechanism for firewall traversal and configuration regardless of the active wireless connectivity.

In the next section we discuss related work, and in Section 3 we present the Host Identity Protocol in detail. In Section 4 we describe the firewall architecture and implementation, followed by performance evaluations, and a discussion of the solution.

RELATED WORK

We have reviewed a number of solutions for firewall control that would support secure mobility and multihoming. The solutions included proposals from Tschofenig et al. (Tschofenig et al., 2005a) and the IETF NSIS working group (Stiemerling et al., 2008). However, these approaches required explicit signaling with the firewall that contradicts our goal of transparent firewall control.

Firewalls have been a well-established technology throughout most of the modern Internet. The basic IP level filtering has been complimented with different extensions to filter transport layer protocols or different application layer technologies, for example, stateful filtering for TCP (van Rooij, 2000). SOCKS (Leech et al., 1996) is a framework for application and transport level gateway technologies for monitoring network connections. The SOCKS gateway is essentially a proxy, which authenticates a client establishing a connection and relays the connection request to server. Different authentication technologies can be incorporated into the SOCKS functionality. SOCKS gateway may also contain translation functionalities to provide communication between IPv4 and IPv6 nodes (Kitamura, 1999).

SANE is a protection architecture for enterprise networks (Casado et al., 2006). It uses a centralized domain controller to implement security policies for the whole network. Clients contact the domain controller and need explicit permission to access any resources. The security policies can be expressed in natural ways, e.g. “give the multimedia group access rights to the company’s mp3 server”. The architecture introduces a new layer between link and IP layer, and is implemented in network switches. The architecture supports mobility, but only within the enterprise network. Also, SANE cannot be used to enforce security policies to internetworks such as the Internet.

Delegation oriented Architecture (DoA) proposes an extension to the current Internet architecture to facilitate the deployment of middleboxes (Walfish et al., 2004). It introduces a new layer and protocol between the network and transport layers. The new layer uses cryptographically secure identifiers similar to HIP. Using the new “middlebox” layer and identifiers, end-hosts can enforce the use of middleboxes, such as firewalls, even when they are not located on the path. Chaining of middleboxes is also possible by allowing the identifiers to be resolvable recursively to other identifiers.

SPINAT (Ylitalo et al., 2005) tackles problems related to IPsec awareness in NATs. One problem in traversing IPsec aware NATs is that the end-hosts determine the IPsec SPIs. This may cause SPI collisions especially when the end-host population within a single NAT is large. A straw-man solution is to drop the keyexchange messages with colliding SPIs and require the key exchange daemons to retry with different SPIs after a timeout. However, SPINAT proposes a more e□cient solution with the IPsec SEET (Ylitalo et al., 2005) mode, which allows NATs to translate ESP SPIs upon collisions. The approach is applicable to asymmetric communication paths and can be used to integrate IPsec to overlay routing. Possibility of IPsec traffic filters are mentioned briefly, but not discussed in detail. The main problem of the SPINAT approach is related to deployment because many existing middleboxes do not support IPsec.

A number of IPsec related asecurity vulnerabilities are described by Aura et al (Aura et al., 2005). They conclude that security policies based on IP addresses can be circumvented in many ways. In HIP, security policies are based on hashes of public keys that makes HIP resilient against those types of attacks.

Ioannidis et al. (Ioannidis et al., 2000) have presented an implementation of a distributed firewall system. The authors use KeyNote to distribute the firewall policies to end-hosts. Their approach supports centralized management of security policies. A drawback is that it requires the end-hosts to update their security policies regularly. As a benefit, the approach allows fine-grained filtering at application layer and a centralized firewall is not a bottleneck for the network.

A preliminary version of this work appeared as a two-page extended abstract in the posters session of Usenix ATC 2007 (Lindqvist et al., 2007). The abstract presented motivation for the approach and notes on the preliminary implementation and performance.

HOST IDENTITY PROTOCOL ARCHITECTURE

This section presents the Host Identity Protocol Architecture explains its protocol mechanisms.

The Host Identity Protocol (HIP) (Moskowitz & Nikander; 2006 Moskowitz et al., 2008) renews the current TCP/IP architecture by introducing a new, cryptographic namespace, the Host Identity namespace, between the transport and network layers as shown in Figure 1. The new namespace consists of Host Identifiers (HIs). A HI is the public keys component of a private-public key pair. HIs can be either public or anonymous. The public HIs can be published, for instance, in the Domain Name System

(DNS) (Nikander & Laganier, 2008). The public identifiers are intended to be long-lived and the anonymous short-lived. For practical purposes, the public keys - the Host Identifiers - are represented by self-certifying hashes of the keys. The hash is called the Host Identity Tag (HIT). The advantage of using HITs instead of HIs, is that HITs are same size as IPv6 addresses and are compatible with current applications. For example, the HITs can be used to replace IPv6 address fields in other existing protocols and Application Programming Interfaces (APIs) (Moskowitz et al., 2008; Komu & Henderson, 2008). HIP also supports Local Scope Identifiers (LSIs) that can be used by legacy IPv4 software because the size of LSIs equals to the size of IPv4 addresses.

HIP architecture proposes so called identity-locator split which relieves IP addresses from their dual role of both identifying and locating end-hosts. In this new TCP/IP architecture, HIs identify endpoints and IP addresses are used to route packets between hosts. By splitting the dual role of unicast IP addresses, HIP supports end-host mobility and multihoming in a relatively straighforward way (Nikander et al., 2008).

The identifiers of the new namespace are deployed locally to the end-hosts in HIP. Alternatively, they can be deployed to global name services, such as DNS, or any overlay, such as OpenDHT (Rhea et al., 2005). However, HIP can be used without any support from the infrastructure by learning the peer's identifier in an opportunistic fashion during HIP key exchange negotiation (Moskowitz et al., 2008).

The HIP specification (Moskowitz et al., 2008) defines base exchange, which creates a secure communication context, called a HIP association, between two hosts. During the base exchange, two hosts authenticate to each other using their public keys and can create a pair of ESP Security Associations (SAs) (Jokela et al., 2008). The base exchange consists of four messages shown in Figure 2. The two hosts are referred as the Initiator (client) and Responder (server).

The HIP control packets consist of a fixed header and variable amount of parameters. The header contains the source and destination HITs and some other fields. All of the packets are

Figure 1. HIP layering model

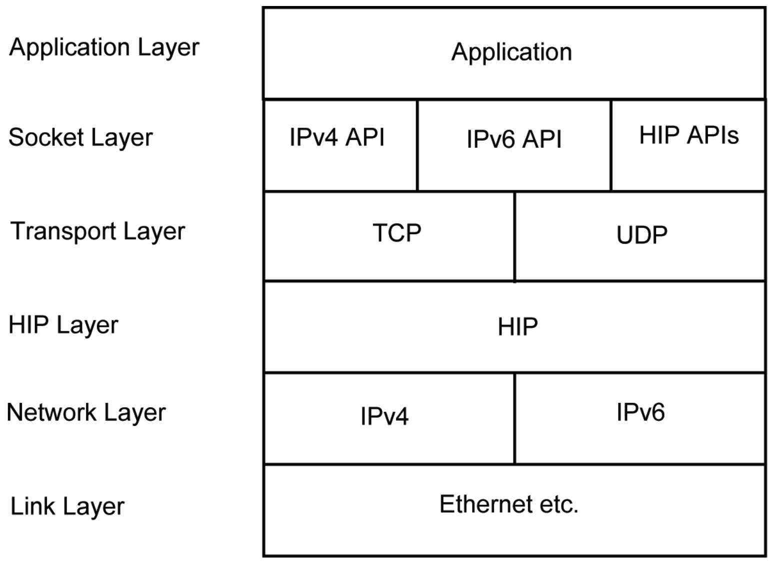
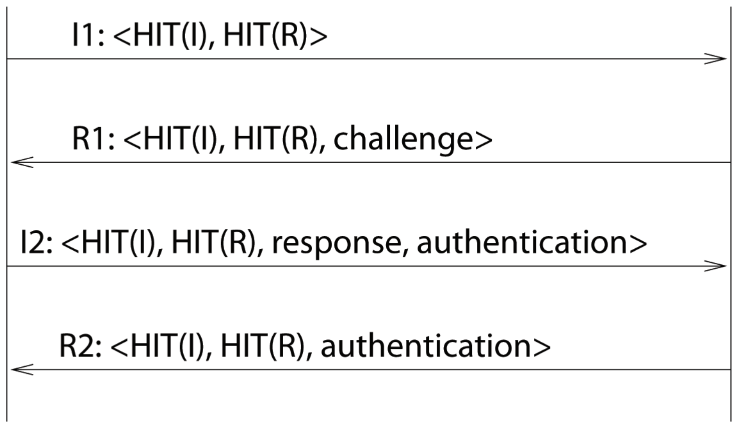


Figure 2. HIP base exchange



protected with public-key signatures except the first one. The first packet, I1, does not contain any parameters. The second packet, R1, contains the HI the Responder, Diffie-Hellman keying material and a computational puzzle (challenge) for the Initiator to solve. The puzzles are used as a mechanism for Denial of Service protection (Moskowitz et al., 2008). The Initiator sends solution to the puzzle, its HI, Security Parameter Index (SPI) for identifying incoming IPsec ESP flow and Diffie-Hellman keying material in I2 packet. The Responder remains stateless until it receives a valid I2. Upon receiving the I2, the Responder verifies the solution to the puzzle, creates state and concludes the base exchange with an R2 packet that contains its SPI number for incoming ESP flow.

HIP mobility and multihoming (Nikander et al., 2008; Nikander et al., 2003) takes place with UPDATE packets after a successful base exchange. A host moving to a different network reestablishes communications with its associated peers by sending an UPDATE packet to its peers. The packet contains parameter called LOCATOR which lists all locators of the mobile host. The parameter can be used in HIP control messages to inform other hosts about alternate addresses at which the originating peer can be reached. This ensures that address bindings can be updated dynamically without breaking the connections. HIP can also be used to establish efficient IPv4 to IPv6 handovers without tunnelling (Jokela et al., 2003).

Rendezvous servers (Laganier & Eggert, 2008) are complementary middleboxes in the HIP architecture. The rendezvous servers have a fixed IP address and serve as a stable contact points for end-hosts. End-hosts update their current location to their rendezvous servers always when they move. As an example, the rendezvous server are useful when two communicating end-hosts cannot publish their new location to each other directly after relocating simultaneously. Instead, they contact each other indirectly through their rendezvous servers that always know where the end-hosts are located. The rendezvous servers forward the first HIP control messages until the end-hosts have synchronized their new locations to each other and can communicate directly.

FIREWALL ARCHITECTURE

Our firewall operates on HIP control messages and ESP flows introduced in the previous section. Next, we describe the details of filtering design and the security consequences.

The Basic Firewall Design

The HIP-based firewall uses HITs to filter packets, but also certain other properties of network packets can be used in the firewall rules. When an Initiator sends an I1 through the firewall, it verifies that the HITs of the I1 message match the filtering rules and then records the HITs and IP addresses of the Initiator and Responder. The firewall has no means to validate the I1 because it does not contain any signatures. Therefore, a forged I1 can reach the Responder through the firewall. However, the firewall blocks the ESP data packets between the two hosts until the base exchange is completed successfully.

The responder sends an R1 and the firewall checks the HITs from its ACLs. This can be used to enforce access control restrictions on the Responders behind the firewall. The firewall records the HITs of the Initiator and the Responder and their IP addresses from the R1.

Upon receiving the R1, the Initiator solves the puzzle and replies with an I2 packet. The signed I2 packet contains the public key of the Initiator. The firewall verifies the signature either using the public key in the I2 packet or a preconfigured public key. If the verification fails, the firewall discards the I2 packet. Similarly, the firewall verifies signature of the concluding R2 packet from the Responder. The I2 and R2 packets contain the SPI values for IPsec ESP that the firewall requires to track ESP tra□c. The firewall also tracks UPDATE messages to continue the tracking of IPsec ESP flows when the IP address of an end-host changes.

Further, the firewall expires the associated state when there is no traffic between the two related end-hosts for a certain time period. This guarantees that the state is removed when the firewall is no longer on the path between the two end-hosts. This can occur, for example, when an end-host moves to a di□erent network or shuts down.

Service Identifiers

The IPsec architecture supports encryption between two hosts. The firewall architecture presented in this paper filters traffic based on HIP control messages and ESP flows. The firewall does not receive the ESP encryption keys of two communicating hosts, and therefore cannot inspect e.g. port numbers in the related ESP flows. Thus, the filtering granularity is lesser than for unprotected traffic where the five-tuple is visible. However, this problem exists even without HIP. For example, it is present in all communications that use IPsec ESP. The problem is also present in TLS, although the port numbers are visible in TLS.

Since the port numbers are not visible in the payload of IPsec ESP, a HIP-aware middlebox requires another way to distinguish between different ports, i.e. services, available at a server. In order to allow more granularity in filtering, we propose using the Host Identifiers also as service identifiers. For each service a server offers, it creates a different public/private key pair. This way, a HIP-aware middlebox or the server itself can separate different services and allow only certain clients to access certain services.

This service identifier approach is compatible with current name look up services. It is a common practice to have separate host names for different services, such as smtp.my.org and www.my.org. Introducing HIP-based service identifiers to the existing DNS would just require adding HIs to DNS (Nikander & Laganier, 2008), with each service a different HI. The HIs can be owned by a single host or multiple hosts. The approach is backwards compatible in the sense that existing servers could still be able to serve non-HIP clients and use existing filtering methods.

An alternative solution to this problem is to introduce a protocol extension to HIP that allows to share the ESP encryption key with the firewall. This can be used also for content filtering purposes, such as, removing viruses from the traffic. This approach is, however, beyond the scope of this paper.

Implementation

Figure 3 shows the design of the firewall implementation (Vehmersalo, 2005). It is based on Linux Netfilter framework (Ziegler, 2001) to intercept network traffic. We used C-based HIPL implementation (Candolin et al., 2003) in our experimentation.

OVERVIEW

The main module of the firewall receiving packets from network interfaces and analyses the packets. It uses the other components of the firewall to produce verdicts based on properties of the packets received. The verdict decides whether the firewall accepts or drops the packet.

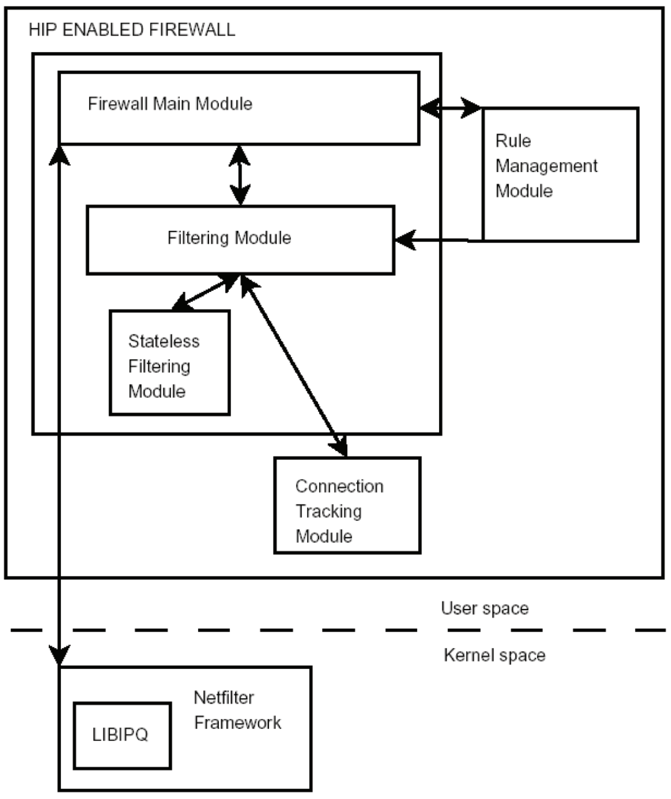
The firewall rules define the local security policies and are contained in the firewall rule set. The firewall rule management module manages the rules and verifies the rule syntax.

The Linux netfilter module contains hooks to the Linux networking stack to intercept packets. The HIP firewall registers to QUEUE target of netfilter and subscribes to HIP-related packet events. The QUEUE target allows userspace applications to read packets from the networking stack and assign verdicts on them.

PACKET FILTERING

The packet filtering consists of two functionalities. First, the firewall analyses packets based on the properties defined in the firewall policies. Second, the firewall assigns a verdict based on the analysis results and policy.

Figure 3. Overall implementation of the firewall architecture. Arrows denote interactions between different components



The firewall provides a number of static properties for packet analysis. The properties include include identity-based authentication of packets, HIP packet type and the direction of the packet (incoming and outgoing). The authentication compares source and destination identities with the firewall rules and also verifies packet signatures. The identities can be specified either has HITs or HIs in the firewall rules.

CONNECTION TRACKING

The packet filtering module calls the connection tracking module when a packet has to be filtered according to the connection state. The main purpose of connection tracking module is to maintain necessary state information to map individual packets to HIP associations. Connection tracking uses source and destination HITs of the HIP control headers, or HIs when available, to associate a HIP packet to a HIP association.

HIP base exchange and mobility-related packets include SPI numbers that the firewall uses to map ESP data packets to the corresponding HIP associations. When firewall analyzes a base exchange, the connection tracking module associates the SPI numbers to the HIP association. This way, the connection tracking can filter unwanted ESP communications based on identities.

The connection tracking module can also authenticate packets and verify packet signatures similarly as the packet filtering module. However, instead of static verification, connection tracking module extracts HIs from HIP traffic dynamically and uses the HI to authenticate the end-point in further communications. The authentication has different nature in connection tracking than in packet filtering. Connection tracking does not verify the identity against static rules but instead attempts to assure the property of sender invariance (Tschofenig et al., 2005b). The sender invariance guarantees that independent of the particular identity, the traffic can be trusted to be originating from the same responder throughout the lifetime of the connection. This is necessary, for instance, in a situation where trusted host inside the network initiates a connection to a previously unknown external host. Thus, the sender invariance makes it more difficult for attacking hosts to abuse the dynamically created access through the firewall.

The connection tracking module also analyzes UPDATE packets. When host introduces a new destination address related to an SPI, or an entirely new SPI, the connection tracker saves the new information to its state structures. The connection tracking must take into account that the two end-points maintain separate state information. This affects, for example, rekeying situations, where old information must remain valid until the other endpoint has acknowledged the new information. In practice, data packets with an old SPI could still be on the way when new SPI is announced. This principle is also discussed by van Rooij (2000) in the context of TCP protocol.

The connection tracking module inserts a timestamp into a connection data structure. The timestamp is then updated whenever valid packets of the connection are encountered. For detecting idle connections, the connection tracker checks the timestamps against a predefined timeout value. Idle connection could result, for example, when a host roams to another network where data is no longer intercepted by the firewall, or just shutdowns. Also, the state created in the firewall by the insecure I1 - R1 exchange does not reserve resources of the firewall indefinitely because of the time-out mechanism.

DATA STRUCTURES

HIP connection tracking module has structures similar to Netfilter's connection tracking. The structures are illustrated in Figure 4. As with Linux Netfilter, a tuple data structure contains information that directly carried by a packet. The implementation provides tuples for both HIP and ESP packets, each in their own data structures.

PERFORMANCE EVALUATION

We conducted some measurements with the firewall prototype to understand its performance. The evaluation environment consisted of a server and five clients. The five clients were located in their own network, separated from the server using a single router that acted also as the HIP-based firewall. The network interfaces operated at 100 Mbit speed and we used IPv6 for connectivity. All of the hosts had a single Pentium 4 processor (3 Ghz) and their Linux kernel version was 2.6.17.3. We used 1024 bit RSA keys as Host Identifiers. The symmetric keys for IPsec were AES (128 bits) for HIP encryption, SHA1 (160 bits) for IPsec authentication and 3DES (192 bits) for IPsec encryption.

We measured the time observed by an client application to complete UNIX connect() system call, which executes a TCP handshake. This time was under 1 ms on the average without HIP. HIP and HIT verification at the firewall, this time was 65 ms on the average due to the extra processing cost of the base exchange. The verification of public key signatures at the firewall caused an extra delay of 1 ms at the maximum. The TCP handshake performance is summarized in Figure 5.

Thus, the firewall prototype introduced only a millisecond delay to a HIP-based TCP connection establishment in our test environment. In other words, the firewall implementation can support thousands base exchanges and mobility updates per second. Filtering ongoing connection creates a similar processing load as, for example, IP address based packet filtering in a traditional firewall. Hence, the HIP-based firewall architecture can scale well on middleboxes and migitates most of the processing cost at the end-hosts.

Figure 4. Connection tracking data model. Arrows represent pointer references between data structures

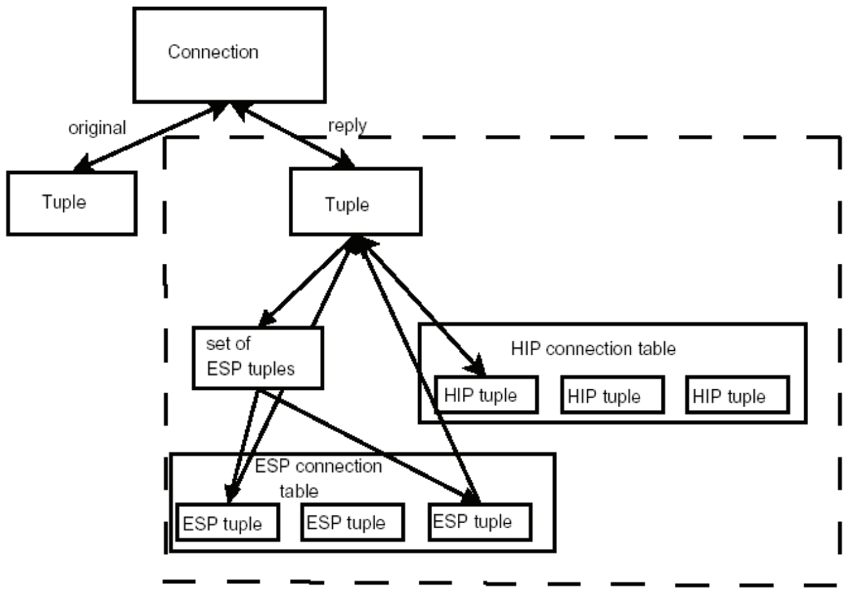
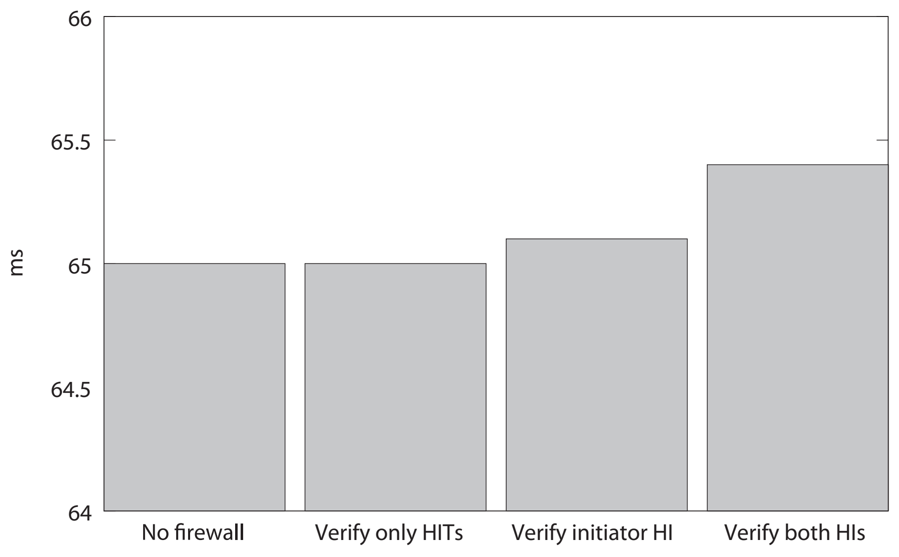


Figure 5. TCP connection establishment time



In addition to latency, we measured also throughput with TCP. The clients streamed 1 minute of TCP stream from the server through the firewall. We used "iperf" tool with default options for the measurements and varied the number of simultaneous file transfers. The results in Figure 6 indicate that the firewall did not affect TCP throughput significantly in our testing environment. The difference between HIP-based and non-HIP-based TCP throughput was approximately 2 Mbit/s.

We also measured TCP data transfer performance under two DoS scenarios and a under third scenario without DoS. In the first scenario, there was 1-4 rogue initiators that were flooding the responder with I1 packets while there was a data transfer in process from the server to the client. The second scenario was similar as first one, but the initiators were using a forged HIT that allows I1 traversal through the ACLs of the firewall. The third scenario includes throughput of varying number of simultaneous and legimitate data transfers. The results are show in Figure 7.

The two DoS attack scenarios have the same throughput performance. With random HITs, the I1 packets of the attackers stop at the firewall. With forged HITs, the firewall accepts the I1 packets of the attackers and they arrive at the responder. However, this causes insignificant processing cost at the responder because the responder has precreated a spool of R1 packets.

I1 flooding slowly degrades TCP throughput, but the throughput with multiple attackers and single legimitate transfer was still larger than with multiple, legimitate TCP data transfers. For example, data transfer with four attackers offered a performance of 29 Mbit/s, where as five simultaneous legimitate data transfers offered 18 Mbit/s. We assume that this was affected by the small size (40 bytes) of I1 packets. TCP packets are much larger and therefore multiple TCP streams reduce the throughput more than smaller I1 packets.

As a summary, the results are quite promising. The overhead of the firewall is negligible, both with control and data traffic. However, we realize that it would be useful to repeat the

Figure 6. Throughput of simultaneous data transfers

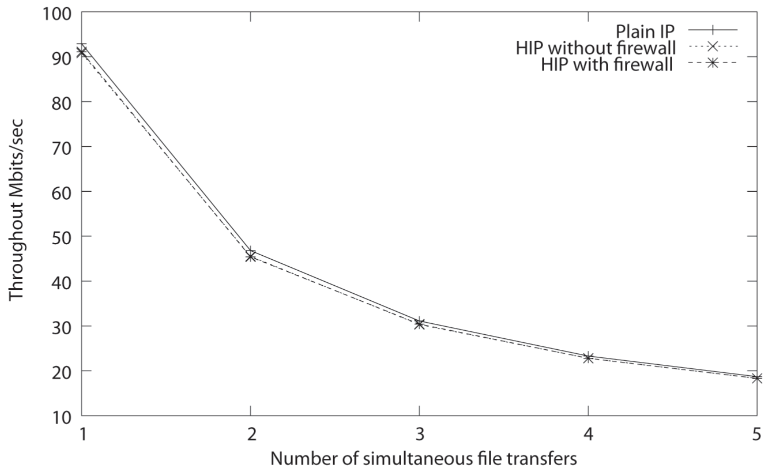
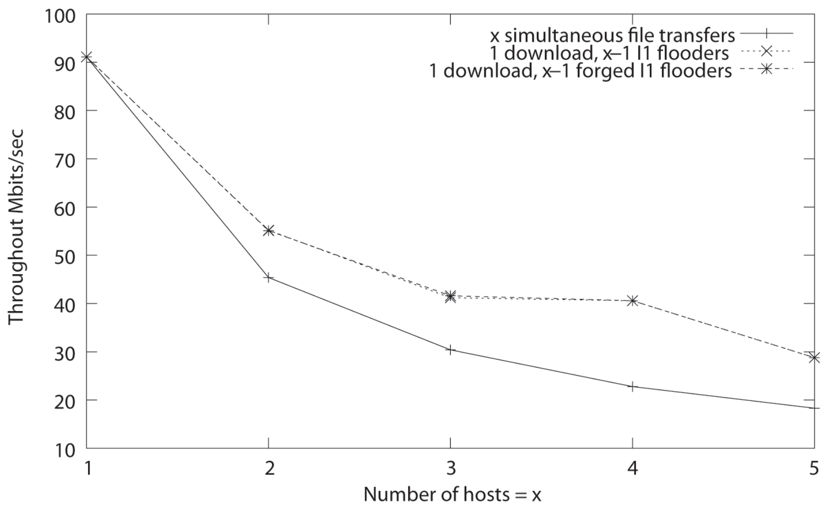


Figure 7. Simultaneous flooding and data transfers



measurements with a larger number of attackers and in gigabit networks.

DISCUSSION

The firewall implementation presented in this paper is scalable according to our measurements. However, we have identified a number of other challenges. The use of flat identifiers for access control introduces side-effects for identity and network management.

Additional Security Issues

The HIP base exchange establishes security associations and keying material between two end-hosts. The firewall tracks the SPIs of the ESP packets. An attacker can thus send ESP packets with valid SPIs through the firewall.

Naturally, the end-hosts discard these packets, but this introduces a possibility of flooding attacks directed towards the end-hosts. These issues are discussed in more detail in (Heer et al., 2009).

Multiple Identities per Host

A HIP-enabled operating system can support multiple Host Identities (Karlsson, 2005). One practical problem arises from the fact that users may have multiple affiliations, and also may want to use anonymous identifiers for privacy reasons.

The privacy management problem has been tackled by introducing a privacy management interface for MAC, IP and HIP layers (Lindqvist and Takkinen, 2006). A user can choose whether the identifiers in these layers are anonymous or public. However, this requires user expertise, and we cannot assume that all users can make decisions on this.

The complexity of the issue arises from the possibility to have multiple public identifiers. Typically, client side network applications do not care about the source identifier and just select the first one from the stack. The problem, in this case, is that an application can choose an identifier that is not configured to a firewall that protects the network. A tempting solution for the user is to communicate all identifiers to the firewall administrator, but this violates the privacy of the user as part of the identifiers are anonymous. In order to solve this, the client host has to have a local policy to enforce the selection of the correct source identifier for a given destination identifier.

Rulesets

A typical firewall today includes a rather large and complex set of filtering rules and setting up this ruleset is a challenging task. One problem is that one rule may be overlap with another. A service may be blocked due to a new badly formatted rule. Rules may leave unwanted access open in the firewall. Serious consequences may occur if the firewall rules are set up accidentally in a wrong order.

Typically, the outcome is that either certain service become out of reach of users because the administrator made a small change in the existing ruleset, or the firewall fails not filter all the intended tra□c. Moreover, if users need access to additional services, inside or outside the firewall, an administrator must manually make a change in the rules; typically this is a long road, and the company security policy may not even allow it.

With our firewall design, rulesets is rather simple and supports mobile clients and renumbering of entire networks. Each client has a fixed firewall rule indepently of the physical location of the client. The rules of compromised clients can be revocated indepently of other clients. This allows also better protection against attacks internal to the corporation.

Management Issues

Although the HIP firewall can reduce administration effort, for example, with network renumbering, the management of HIs can be cumbersome. Further work is needed to find out ways how the public keys are introduced to the firewall. With HIP, the end-host creates and manages its own private keys. A problem in this approach is how to submit the public keys to the administration of the firewall. Next, we present two straw-man solutions to clarify the problem.

In the first solution, the administration is responsible for the installation of the operating system for e.g. a laptop. This is the usual case in many enterprise settings. However, granting access to the private key of the end-host introduces a possibility for privacy violations.

In the second solution, the user creates its own public/private key pairs, but the problem here is that how the keys are communicated to the firewall in a trustworthy way. Perhaps the user could hand the right key on a USB stick to administratation, but that is rather cumbersome in networks with thousands of users. Hence, a network-based transfer with a user-friendly interface would seem more scalable here. An enterprise network could adopt a similar way to configure the HIP firewall as the EasyVPN (Benvenuto & Keromytis, 2003) approach. The EasyVPN approach uses a WWW server and TLS connections for configuring IPsec VPN gateways to clients. For HIP-enabled firewall, a similar management system could be implemented.

Another problem is that the identifiers in HIP are flat. IP addresses are hierarchical and contain a network prefix, which can be used for grouping IP addresses. There is no similar mechanism for the flat public key based identifiers in HIP. Public keys matching a certain pattern are computationally difficult to create and might potentially introduce further security problems.

One additional drawback of the current architecture is that it does not support easy grouping of services and administrators require a high-level management interface. The firewall implementation provides an interface that allows extending the management to e.g. web based management interface. The management interface could be used to attach semantic data to the flat identifiers. The data could consist of groups, user names and expiration dates for access control.

Combined Firewall and Rendezvous Service for Mobile Nodes

We have presented a solution where the firewall requires being located on the path. When a mobile host moves outside of the corporation, the firewall cannot filter its traffic anymore. NAT traversal extensions for HIP (Komu et al., 2009) introduce a new type of rendezvous service that forwards all HIP control traffic. Such service could be used to implement off-path filtering when a firewall service is coupled with the new rendezvous service. The mobile hosts would just have to deny all incoming base exchanges from other hosts than the new rendezvous server. This type of firewall would protect also communications for a mobile node that moves e.g. from an enterprise network to an public network at the cost of triangular routing.

CONCLUSION

We have presented a firewall architecture that allows both users to benefit of end-to-end mobility and multihoming in a secure way and allows the enterprise network management to centrally enforce their corporate network security policies.

Our initial measurements indicate that the overhead introduced by the firewall architecture is relatively small as the existence of the firewall in the network adds a delay of under 1 ms. The approach does not require an additional firewall control protocol when both the client and server support HIP. Despite that the HIP architecture seems promising for establishing the identifier/locator split to the Internet and providing seamless secure mobility and multihoming, it is not without tradeoffs. The management of the flat identifiers used in HIP introduces new challenges to operating systems and network management. We have proposed and implemented viable approaches to solving these problems. The possibility to use public keys as secure service identifiers and cryptographically secure authentication provided by our architecture is not available in the current Internet architecture or previous proposals.

ACKNOWLEDGMENT

The authors thank Sanna Liimatainen, Laura Takkinen, Kristian Slavov, Samu Varjonen and Antti Ylä-Jääski for their comments on the paper. The authors are also grateful to N. Asokan for fruitful discussions and suggesting the service identifier approach.

REFERENCES

Aura, T., Roe, M., & Mohammed, A. (2005). Experiences with host-to-host IPsec. *Security Protocols, 13th International Workshop.*

Benvenuto, M. C., & Keromytis, A. D. (2003). EasyVPN: IPsec Remote Access Made Easy. *17th USENIX Large Installation Systems Administration (LISA) Conference.*

Candolin, C., Komu, M., Kousa, M., & Lundberg, J. (2003). An implementation of HIP for Linux. In *Proc. of the Linux Symposium 2003*, Ottawa, Canada.

Casado, M., Garfinkel, T., Akella, A., Freedman, M. J., Boneh, D., McKeown, N., & Shenker, S. (2006). Sane: A protection architecture for enterprise networks. In *Proc. of 15th USENIX Security Symposium.*

Heer, T., Hummen, R., Komu, M., Götz, S., & Wehrle, K. (2009). End-host authentication and authorization for middleboxes based on a cryptographic namespace. In *ICC2009 Communication and Information Systems Security Symposium.*

Ioannidis, S., Keromytis, A. D., Bellovin, S. M., & Smith, J. M. (2000). Implementing a distributed firewall. In *Proceedings of the 7th ACM conference on Computer and communications security.*

Jokela, P., Moskowitz, R., & Nikander, P. (2008). *Using the encapsulating security payload (ESP) transport format with the host identity protocol (HIP).* RFC 5202, IETF.

Jokela, P., Nikander, P., Melen, J., Ylitalo, J., & Wall, J. (2003). Host identity protocol: Achieving IPv4 - IPv6 handovers without tunneling. In *Proc. of Evolute workshop 2003: "Beyond 3G Evolution of Systems and Services*

Karlsson, N. (2005). *Enabling Multiple HIP Identities on Linux*. Master's thesis, Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory.

Kitamura, H. (1999). Entering the IPv6 communication world by the SOCKS-based IPv6/IPv4 Translator. In *Proc. of INET99*.

Komu, M., & Henderson, T. (2008). *Basic Socket Interface Extensions for Host Identity Protocol (HIP).* IETF.

Komu, M., Henderson, T., Tschofenig, H., Melen, J., & Keranen, A. (2009). *Basic hip extensions for traversal of network address translators.*

Laganier, J., & Eggert, L. (2008*). protocol (HIP) rendezvous extension*. RFC 5204, IETF. Host identity

Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., & Jones, L. (1996). *Socks protocol version 5.* RFC 1928, IETF.

Lindqvist, J., & Takkinen, L. (2006). Privacy management for secure mobility. In *Proceedings of the 5th ACM CCS Workshop on Privacy in Electronic Society.*

Lindqvist, J., Vehmersalo, E., Komu, M., & Manner, J. (2007, June 17-22). Enterprise Network Packet Filtering for Mobile Cryptographic Identities (extended abstract). In *USENIX annual technical conference poster session*, Santa Clara, CA.

Moskowitz, R., & Nikander, P. (2006). *Host Identity Protocol Architecture*. RFC 4423, IETF.

Moskowitz, R., Nikander, P., Jokela, P., & Henderson, T. (2008). *Host identity protocol.* RFC 5201, IETF.

Nikander, P., Arkko, J., Vogt, C., & Henderson, T. (2008). *End-Host mobility and Multi-Homing with the host identity protocol*. RFC 5206, IETF.

Nikander, P., & Laganier, J. (2008). *Host identity protocol (HIP) domain name system (DNS) extension.* RFC 5205, IETF.

Nikander, P., Ylitalo, J., & Wall, J. (2003). Integrating security, mobility, and multi-homing in a HIP way. In *Proc. of Network and Distributed Systems Security Symposium (NDSS'03)*, San Diego, CA, USA. Internet Society.

Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., et al. (2005). OpenDHT: A public DHT service and its uses. In *Proc. of ACM SIGCOMM'05,* Philadelphia, PA, USA: ACM Press.

Stiemerling, M., Tschofenig, H., Aoun, C., & Davies, E. (2008). *NAT/Firewall NSIS Signaling Layer Protocol (NSLP).*

Tschofenig, H., Nagaraja, A., Shanmugam, M., Ylitalo, J., & Gurtov, A. (2005a). Traversing Middleboxes with Host Identity Protocol. In *Proc. of ACISP'05.*

Tschofenig, H., Nagarajan, A., Torvinen, V., Ylitalo, J., & Grimminger, J. (2005b). *NAT and firewall traversal for HIP: draft-tschofenighiprg-hip-natfw-traversal-02.*

van Rooij, G. (2000). Real Stateful TCP Packet Filtering in Ipfilter. *2nd International SANE Conference*, Maastricht, The Netherlands.

Vehmersalo, E. (2005). *Host Identity Protocol Enabled Firewall - A Prototype Implementation and Analysis*. Master's thesis, Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory.

Walfish, M., Stribling, J., Krohn, M., Balakrishnan, H., Morris, R., & Shenker, S. (2004). Middleboxes no longer considered harmful. In *Proc. of the 7th USENIX Symposium on Operating System Design and Implementation (OSDI 2004)*, San Fransisco, CA, USA: ACM Press.

Ylitalo, J., Salmela, P., & Tschofenig, H. (2005). SPINAT: Integrating IPsec into Overlay Routing. In *First International Conference on Security and Privacy for Emerging Areas in Communication Networks*.

Sams. Janne Lindqvist is a specialist researcher and PhD candidate with the Department of Computer Science and Engineering of Helsinki University of Technology (TKK). He received his MSc (Tech) on February 2005 from TKK. He will publicly defend his doctoral thesis on Practical Privacy Enhancing Technologies for Mobile Systems *on June 5, 2009. His research interests are in systems security and privacy, especially in building secure, privacy-preserving and usable systems.*

Essi Vehmersalo is a senior software engineer working at Nokia Corporation. She received her MSc in computer science from the Helsinki University of Technology (TKK) 2005 with thesis topic related to Host Identity Protocol enabled firewall technology. After that she has worked on mobility solution of the networking middleware of S60/Symbian smart phone platform. Currently she is working on Nokia music service.

Miika Komu graduated as MSc on 2004 and he is working on his postgraduate studies at Helsinki University of Technology. He is affiliated as a researcher at Helsinki Institute for Information Technology. He has been participating to various HIP related research, engineering, standardization and deployment activities since 2001.

Jukka Manner (born 1972) received his MSc (1999) and PhD (2004) degrees from the University of Helsinki. He is a full professor (tenured) and holds the chair of networking technology at the Department of Communications and Networking (Comnet) of the Helsinki University of Technology (TKK). He is also Adjunct Professor and a senior researcher at the University of Helsinki, and contributes to the research at the Helsinki Institute for Information Technology (HIIT). His research and teaching focuses on development of a future Internet, particularly in topics related to networking beyond IP, energy efficiency, mobility management, QoS, and transport protocols. He is the Academic Coordinator for the Finnish Future Internet research programme. He is an active peer reviewer and member of various TPCs. He has contributed to standardization of Internet technologies in the IETF for over 10 years, and is currently the co-chair of the NSIS working group. He has been principal investigator and project manager for over 15 national and international research projects. He has authored over 50 publications, including several IETF RFCs. He is also a member of the IEEE.

Publication V

Miika Komu and Janne Lindqvist. Leap-of-Faith Security is Enough for IP Mobility. In *Consumer Communications and Networking Conference (CCNC'09)*, Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference, Las Vegas, pp. 830-834, ISBN 978-1-4244-2308-8, February 2009.

Leap-of-Faith Security is Enough for IP Mobility

Miika Komu
Helsinki Institute for Information Technology
Helsinki University of Technology and University of Helsinki
Email: miika.komu@iki.fi

Janne Lindqvist
Helsinki University of Technology
Department of Computer Science and Engineering
Email: janne.lindqvist@tml.hut.fi

Abstract— **Host mobility presents a challenge for security protocols. For example, many proposals exist for integrating IPsec to Mobile IP. However, the existing approaches are cumbersome to configure and contain many round trips for security and mobility updates. The Host Identity Protocol (HIP) is being developed in the IETF to provide secure host mobility and multihoming. The default way to operate the protocol is that the connection initiator knows the peer's public key or a hash of the public key. This requires either infrastructure support or pre-configuration which introduces difficulties for deploying the protocol. In this paper, we present an implementation and evaluation of HIP that creates leap-of-faith security associations. The implemented approach establishes end-to-end security without requiring any new infrastructure to be deployed. We argue that since worldwide PKI is nowhere near, and seems to nearly impossible to br deploy in practice, leap-of-faith security is enough for Internet access and mobility. In our view, the deployment of opportunistic HIP even makes the deployment of DNSSEC unnecessary for most applications.**

I. Introduction

In the vast number of approaches to host mobility, many of the proposals ignore security issues. For example, extensively researched Mobile IP(v6) protocol introduces difficulties with IPsec [1].

Host Identity Protocol [2] integrates to IPsec to secure mobility and multihoming. In the HIP architecture, the IP addresses are relieved from their role as identifiers by public keys or hashes of the public keys. When the IP address of the host changes, the connection is still bound the the same cryptographically secure identity. Thus, transport layer connections can tolerate changes in IP addresses using HIP.

In Mobile IPv6, the server side does not need any changes, but it can support mobility optimizations. The security of Mobile IPv6 was designed to avoid introducing any new security threats to the Internet [3]. Mobile IP uses IP address as the identifier. On the other hand, Host Identity Protocol was designed to introduce a new cryptographic identity space for Internet and to use IPsec as the default mechanism to protect transport layer communication.

The concept of leap-of-faith security or weak authentication between untrusted principals [4] has been used and implemented in many security protocols. For example, Secure Shell (SSH) protocol uses leap-of-faith security as as follows. When a client connects to a server the first time, the user sees and verifies the fingerprint of the server's public key, and the SSH software stores the public key to disk. Next time the client connects to the server, the SSH client software compares the server key to the one stored on the disk. If they do not match, SSH alerts the user of a possible man-in-the-middle attack. Thus, the assumption of the leap-of-faith security is that there is no active attacker in the network during the first connection.

In this paper, we present the design and implementation of a leap-of-faith security approach to HIP called the opportunistic mode. The opportunistic mode is briefly described in the base specification of HIP, but, for example, API issues are left aside. The literature does not contain any experimental results on opportunistic mode and therefore we have experimented with a way of of implementing the opportunistic mode and its APIs which do not interfere with the normal operation of HIP. The implementation supports incremental deployment because it allows fallback to non-HIP based communication when the peer does not support HIP. We argue that the opportunistic end-to-end security approach is enough for Internet access for heterogeneous wired and wireless networks since the deployment of global public-key infrastructure is virtually impossible. It should be noted that AAA architectures are beyond the scope of this paper, since we focus on end-to-end security. We have implemented the approach with HIP, but the same experiences should be applicable to other end-to-end mobility protocols.

The rest of the paper is organized as follows. We first proceed to introduce the Host Identity Protocol architecture. Then, we discuss related work. Next, we give the design and implementation details, followed by discussion section. We finish the article in conclusions.

II. Host Identity Protocol

Host Identity Protocol (HIP) [2] introduces a new global namespace, the Host Identifier (HI) namespace, for transport-layer connections. The namespace separates transport layer and network layer locators. This allows transport layer connections to survive when the network-layer address changes due to end-host mob5Bility or multihoming. The new global namespace makes it also possible to name and contact hosts behind private-address realms controlled by NAT boxes [5].

The HIP namespace is unmanaged in the sense that no central authority for creating the names exists. A name in the namespace is statistically unique. Namespace collisions are highly unlikely due to the size of the addresses space which corresponds to the IPv6 address space. The namespace is cryptographic by its nature; a HI is essentially a public key.

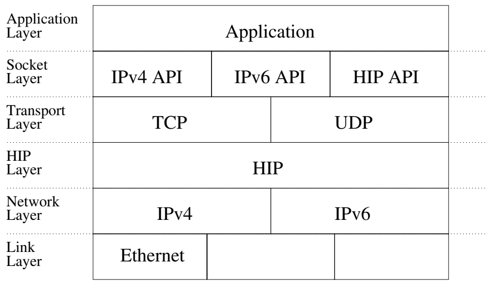


Fig. 1. HIP layering and naming architecture

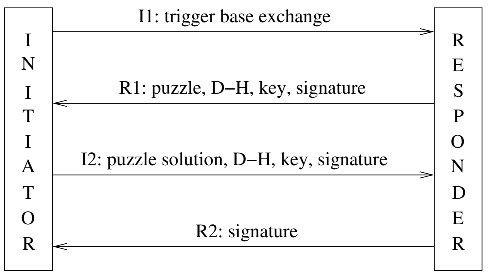

INITIATOR
RESPONDER
I1: trigger base exchange
R1: puzzle, D−H, key, signature
I2: puzzle solution, D−H, key, signature
R2: signature

Fig. 2. HIP base exchange

Forging of such names is very difficult. In addition, HIs can be used for strong end-to-end and end-to-middle authentication.

Public keys can be of variable length and longer keys cannot be used as socket endpoints in the existing sockets API [6]. To be able to pass public keys through the current sockets API with existing applications and networking stacks, HIP employs two shorter, fixed-size presentations of public keys. The shorter representations are also used in the protocol messages to ease protocol encoding and to reduce the length of control messages. First, Host Identity Tag (HIT) is a hash of the public key. The HIT has a fixed 28 bit prefix to separate it from routable IPv6 addresses. Second, Local Scope Identifier (LSI) is an identifier that can fit into an IPv4 address to support IPv4-only applications and is valid only in the context of the local host [7].

The HIP namespace requires a new layer in the networking stack to handle translation between HIs and routable addresses (i.e. locators). The new layer is located between transport and network layers, and it translates HIs to locators and vice versa. The HIP layer can map a HI either to an IPv4 or IPv6-based locator dynamically as illustrated in Figure 1.

Typically, a user inputs the client application the server host name and service name. Then, the system resolver translates these names to their machine readable representations. Effectively, the resolver searches the host name from DNS and returns the corresponding IP address(es) and numerical service port number. The address information may also also the HITs of the server if they were found from DNS. When the application connects to a HIT, this triggers a base exchange between the hosts. Upon successful completion, IPsec ESP [8] secures the application data between the client and server.

HIP uses IPsec Encapsulated Security Payload (ESP) to secure and encrypt communication between two hosts [8]. However, before IPsec can be used, the hosts need to create shared secret keys with each other. The procedure to create the keys in HIP is called the base exchange [9] and it is illustrated in Figure 2. The base exchange is a four way Diffie-Hellman key exchange during which hosts negotiate the IPsec keys and algorithms, and learn each others public keys. All base exchange messages, except the first one, are signed with DSA or RSA private keys to protect the integrity of the messages. The initiator starts the base exchange with an *I1* message that does not contain any signature to prevent DoS attacks that try to trick the responder to consume its time in verifying signatures from unauthenticated initiators. Normally, the I1 message would contain the responder's HIT (HIT(R)), but in opportunistic mode, the HIT(R) field is empty. Responder replies with an *R1* message that contains always the responder's HIT. It includes also responder's public key and a computational puzzle. The puzzle protects the responder from denial of service attacks because the responder can increase its difficulty when it is under attack. The initiator solves the puzzle and responds with a signed I2 message that contains also the initiator's public key. The responder receives the *I2* and validates the puzzle and other parameters, and concludes the base exchange with the *R2*. After this, both hosts have authenticated each other and have established keys for ESP protected application data transfer.

HITs are problematic from the viewpoint of DNS which supports only hierarchical identifiers. HITs are flat in the sense that they do not contain any hierarchical information. For this reason, resolving a HIT to a hostname or IP address using the currently deployed DNS is impossible. There are at least two scenarios where this can be a problem. First, a problem can occur e.g. when a server receives a connection from a HIT of a client and tries to verify the HIT from DNS. An example case of this are IRC servers that typically try to check the client information with a reverse DNS query and force the IRC connection to timeout when the check fails. In the second case, which is referred to as the *referral problem*, an application can communicate the HIT of its peer application to a third application running on some other host. The host of the third application may not have any means to route packets to the HIT because the host cannot resolve the HIT to an IP address using DNS. For example, FTP supports this kind of behaviour [6]. In such a case, it is possible to use an overlay to store and retrieve HIT-to-IP mappings, or use the overlay to route the packets without any IP address information at the end-hosts [10].

Basic mobility in HIP is based on a moving host always informing its peers on its new location using a signed message. Each peer then verifies that the message is from the authentic host and authentic location before sending any ESP traffic to the new location. During the verification procedure, which is called as the return routability test, the peer effectively sends a signed nonce to the host in the new location. Then, the host in the new location signs the nonce with its own public key and sends the nonce back. This way, the hosts can authenticate handovers and the return routability test protects against replay attacks.

III. Related Work

In this section, we describe different leap-of-faith security approaches that, however, do not provide global Internet-wide mobility and multihoming.

Koponen et al. [11] have proposed suspend-mode mobility for SSH and TLS. Their approach is based on the assumption that suspend and resume activity for laptops is the usual case for mobility, and that host mobility in Mobile IP and HIP models is not needed. As a counter argument, it is perceivable that deployment of heterogeneous networks necessitates mobility as supported by HIP or Mobile IP. Koponen et al's approach is applicable only to SSH-based connections. In contrast, our approach supports both unmodified legacy applications and also UDP-based network communication.

RFC4322 [12] describes opportunistic encryption for IKE. The idea in the RFC is to distribute public keys in the DNS and use DNSSEC [13] to protect against active attacks [12]. If DNSSEC is deployed, our opportunistic security approach can also leverage it. A benefit of the approach is that it does not require new Resource Record fields to the DNS. A drawback is that the approach does not support mobility and multihoming.

An approach for implementing early opportunistic key agreement for ad hoc networks is described by Candolin et al in [14]. The approach is based is to utilize ICMP to establish IPsec security associations that can be used to secure e.g. neighbor discovery. The drawback of the approach is that it does not support global host mobility.

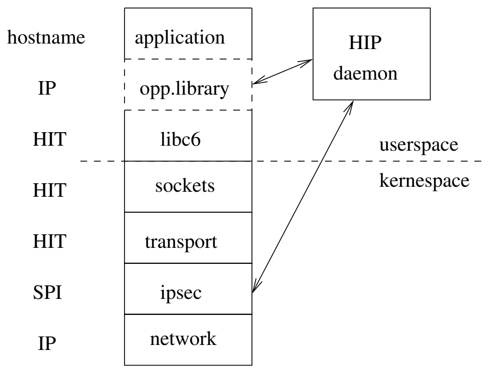


Fig. 3. Layering and Software Module Organization

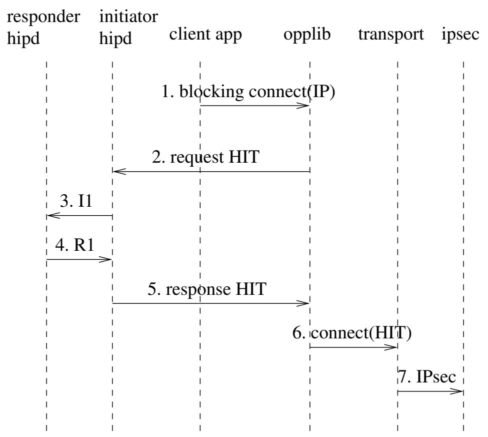


Fig. 4. Flow Diagram of Opportunistic Base Exchange

IV. Implementation

This section presents the implementation design of our leap-of-faith security approach and performance tests.

A. Design

We have implemented opportunistic mode in HIP for Linux implementation. We implemented the opportunistic mode as an interposition library that intercepts socket calls and translates IP addresses to HITs as shown in Figure 3. The library supports legacy applications because it does not require any modifications to the source code of applications.

A developer, system administrator or user can use the library using two methods. In the first method, a developer links the source code of an invidual application to the library. The second method, dynamic linking, is easier for administrators and users, and does not require access to the source code. Dynamic linking means that the user or administrator enables the library dynamically using `LD_PRELOAD` environment variable. This can be done with different granularities, such as per application, per user or per system, depending on security requirements. The different granularities can also be used to avoid the overhead of introduced by security. To reduce the configuration steps for users, the opportunistic mode should be provided at system level. Both client and server-based applications can use the library, although we believe that it is more beneficial to client applications.

The process of the opportunistic handshake is illustrated in Figure 4 from the viewpoint of an initiator. In step 1, the application calls a socket API function that sends data using IP addresses, for example connect() or sendto(). The opportunistic library then intercepts the function call. In step 2, the library (denoted as opplib in the figure) queries the HIP daemon for the corresponding HIT. The query blocks until the HIP daemon responds. The daemon triggers opportunistic base exchange with the peer in step 3. Upon receiving the R1 in step 4, the HIP daemon sends the responder's HIT to the library in step 5 and proceeds with the base exchange. Now, the opportunistic library can proceed with the translation and connect to the HIT of the responder in step 6. Finally, the control flow proceeds from transport to IPsec layer processing in step 7, which transmits the data over ESP.

The library consists of roughly 2000 physical lines (SLOC) of C code. It does not translate raw sockets or sockets that are already bound to HITs and it can translate both IPv4 and IPv6 addresses to HITs. It creates a completely new HIT-based socket for each IP-based socket because the sizes of IPv4 and IPv6 addresses are different.

The library processes datagram-oriented (UDP) socket calls differently from connection-oriented (TCP) socket calls. With datagrams, the local or peer address can change at any point of the communication and may require a new HIP base exchange. To avoid unnecessary base exchanges, the library caches the address-to-HIT translation for a specific socket until the application changes the peer address and only then triggers a new opportunistic base exchange to discover the HIT of the peer.

The opportunistic mode was mostly implemented within the library, but it also required some changes to the existing HIP daemon. The changes in daemon were minimal to implement the responder processing. Namely, the responder just selects a local HIT for R1 when the destination HIT of the I1 is NULL.

The initiator part of the daemon required to store state information related to opportunistic connections in the HIP association database. The database stores the state of HIP base exchanges and it is indexed using HITs of the initiator and responder. The I1 packet has an empty destination HIT in opportunistic mode and when the daemon accesses the database for such a packet, it does not use a blank destination HIT to index it. Instead, the daemon calculates a "pseudo HIT" based on the HIT prefix and the IP address of the responder. This avoids mixing up multiple simultaneous opportunistic base exchanges and protects against active attackers that try to send R1 packets to the initiator from other network addresses. It should be noticed that the daemon skips the pseudo HIT generation if it finds existing HIP association with the peer and returns the peer HIT immediately.

The implementation supports fallback to plain TCP/IP based on timeouts when an initiator detects that the peer does not support HIP. The daemon resumes the blocked library when it does not receive an R1 during a certain period of time. The library then discards the translation step and proceeds with unprotected communication using the original IP address. The implementation caches the addresses of the peers that did not support HIP to avoid triggering the base exchange unnecessarily again. The daemon flushes cache entries upon two events. First, this occurs when the HIP association closes or expires. Second, this occurs when local or peer host moves. Local host movement causes flushing because it might move between NATted realms that have overlapping private address spaces. Peer host movement address causes flushing because the previous IP address of the peer could be occupied by a new host and new connections to this IP should be directed to the host that presently occupies the address.

B. Performance

We evaluated the performance of the opportunistic mode on two 64-bit 2 GHz Intel Dual Core computers running Ubuntu 8.04. The machines were connected to each other using a direct 1 Gbit link. We used 1024-bit RSA asymmetric keys as HIs. We employed 128-bit AES keys for HIP and ESP encryption and 160-bit SHA1 keys for IPsec authentication. The linux kernel version of the computers were 2.6.25.8 and included BEET [8] patches for ESP that were adopted as part of the standard linux kernel in an upstream kernel version. The standard deviations were relatively small the measurements and therefore they are not included in this section.

We observed an average 47 ms delay for an application complete TCP connect() call to a HIT, which consists of the time to complete the base exchange and the TCP handshake. The delay was roughly the same with the opportunistic library. However, we experienced a minimum of three 3 seconds with each LSI measurement. The reason for this is that the LSI implementation does not yet cache data packets that the host sends during the base exchange. Instead, it just drops the packets. The Linux TCP stack includes fixed three second retransmission timeout by default when the first TCP packet is lost. The same behavior does not occur in the current library implementation because it blocks the socket calls and therefore no packets are actually lost.

The RTT between the two machines was 0.261 ms with plain ICMPv4, 0.319 ms using HITs and 0.658 ms with LSIs. We did not measure RTT with the opportunistic library because it does not support translation of raw sockets yet. Iperf version 2.0.2-4 showed a throughput of 942 Mbits/s for plain TCP, 300 Mbits/s for HIT-based TCP connection, 296 MBits/s for the opportunistic HIP library and 94 MBits/s for LSI-based connection. We assume that the LSI throughput was so low because the implementation is highly unoptimized.

We tested the opportunistic library in a mobility scenario where a TCP-based test application running on a local host created a connection to a peer application residing on a remote host. Then, we forced the local host to change its IP address to a new one while the TCP connection was still active. As a result, the TCP connection survived because of the HIP handover and local application was still able to send data to the peer application. The connection survived even though the local application was bound to the old IP address because the library was still translating the old address to the HIT.

V. Discussion

Our performance measurements indicate that the library offer the same performance as HIT based data transfers. The library exceeded the performance of the LSI implementation because it is still very unoptimized.

We observed that an opportunistic base exchange was typically invoked twice for the first connection of an application. The reason for this was that all of the library calls were wrapped, including also the DNS query. The DNS query triggered the first base exchange and the connection to the peer application triggered the second. This could be avoided by caching the IP addresses in the implementation, or by setting up by-pass policies for DNS ports.

The library requires further experimentation with varying kind of applications and environments. We have successfully tested it with the Firefox web browser, iperf and some other simple applications that create network connections in simple fashion. We are in the process of experimenting the library with a wider range of networking applications, such as real-time applications.

We are still improving the library implementation to meet some additional challenges. For example, we would like the library to support non-blocking operation. Also, the library does not yet cover all possible socket calls even though it handles the most common ones. We have also experienced that `LD_PRELOAD` was difficult to apply in some systems, such as CentOS 5.2 and Maemo tablets. For such systems, we suggest to apply the opportunistic mode as explained in [15].

Opportunistic HIP requires a relaxed security model in terms of leap of faith or time. It makes two security-related assumptions. The first assumption is that a remote host cannot guess the time when the initiator sends the I1 and reply using its own R1. The second assumption is that an attacker can snoop the trigger but cannot reply using its own R1. Typically, packet snooping is quite easy in e.g. wireless networks which makes the opportunistic mode especially vulnerable in such kind of broadcast environment. However, this quite difficult in practice because the attacker has to forge its address as the destination address of the I1 trigger and to be able to respond from this address. The opportunistic mode is a "better than nothing" security guarantee until sufficiently amount of HIP infrastructure is deployed on the Internet. We believe that heterogeneous wired and wireless networks cannot support a global PKI in any case.

The fallback approach, however, could introduce the possibility to attack the base exchange by down-negotiation. The attacker (middleman) can just drop the I1 packet. The initiator now sends e.g. a TCP SYN to the Recipient and the middleman catches it, and establishes connections to the initiator and the responder. This way, the middleman does not need to use any cryptographic operations to catch the traffic. However, this attack can be mitigated by security policies in the end-hosts, for example, by not allowing unencrypted connections.

In addition to attackers, packet loss could also be a problem with the fallback approach. The problem with the fallback is that it is based on timeouts. The timeout has to be long enough to provide reasonable guarantee against packet losses, but, on the other hand, long timeouts frustrate the user. Long timeouts could be avoided using explicit detection of HIP capability of the peer in a backwards compatible way as described in more detail in [16].

There are different pros and cons in using different kind of identifiers at the application layer, we do not see HITs, LSIs and opportunistic IP identifiers as rivalling or conflicting approaches. On the contrary, they complement each other to support different kinds of applications. However, a system should provide some kind of default policy for selection among different kinds of identifiers. For the default policy, we suggest that the application first try to resolve HIs of the peer. This is required for forwards compatibility with PKI. When the application supports IPv6 through the system resolver, the resolver returns the HIT of the peer to the application. When the application supports only IPv4, the resolver returns an LSI instead. This way, the application can detect the presence of HIP. It then tries the opportunistic mode as the last option, i.e., when the resolver returns just an IP address to the application in the absence of a HI. For backwards compatibility, the opportunistic library should fall back to non-HIP communication when the peer does not respond with an R1 during certain period of time.

VI. Conclusions

We have presented the design and implementation of a leap-of-faith end-to-end security architecture for host mobility. Our implementation allows legacy applications to establish IPsec security associations and change their points of network attachment without losing connectivity. One of the major advantages of our architecture is the backward compatible implementation of leap-of-faith security as the hosts can fall back to plain TCP or UDP connections when a peer does not support HIP. Hosts benefit from secure mobility, assuming that an active attacker cannot mount a man-in-the-middle attack in the beginning of the communication. We showed that our approach is sufficient to secure communication in heterogeneous network environments in the lack of a global PKI or DNSSEC deployment.

References

[1] V. Devaparalli and F. Dupont, "RFC 4877: Mobile IPv6 Operation with IKEv2 and the revised IPsec Architecture," Apr. 2007.

[2] R. Moskowitz and P. Nikander, "Host Identity Protocol Architecture," IETF, RFC 4423, May 2006.

[3] T. Aura and M. Roe, "Designing the Mobile IPv6 Security Protocol," *Annales des télécommunications / Annals of telecommunications, special issue on Network and information systems security*, no. 3-4, March-April 2006.

[4] J. Arkko and P. Nikander, "How to authenticate unknown principals without trusted parties," in *LNCS 2845: Security Protocols, 10th International Workshop*, Apr. 2003.

[5] M. Komu, T. Henderson, P. Matthews, H. Tschofenig, and A. Keränen, "Basic HIP extensions for traversal of network address translators draft-ietf-hip-nat-traversal-04," July 2008, work in progress, expires in Jan 2009.

[6] M. Komu, S. Tarkoma, J. Kangasharju, and A. Gurtov, "Applying a Cryptographic Namespace to Applications," in *Proc. of the first ACM workshop on Dynamic Interconnection of Networks (DIN 2005)*. Cologne, Germany: ACM Press, Sept. 2005.

[7] T. Henderson, P. Nikander, and M. Komu, *RFC 5338: Using the Host Identity Protocol with Legacy Applications*, Sept. 2008.

[8] P. Jokela, R. Moskowitz, and P. Nikander, "Rfc5202: Using the encapsulating security payload (ESP) transport format with the host identity protocol (HIP)," IETF, RFC 5202, Apr. 2008.

[9] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "RFC5201: Host identity protocol," Apr. 2008.

[10] P. Nikander, J. Arkko, and B. Ohlman, "Host identity indirection infrastructure (Hi3)," in *Proc. of The Second Swedish National Computer Networking Workshop 2004 (SNCNW2004)*, Karlstad, Sweden, Nov. 2004.

[11] T. Koponen, P. Eronen, and M. Särelä, "Resilient Connections for SSH and TLS," in *USENIX Annual Technical Conference*, May 2006.

[12] M. C. Richardson and D. H. Redelmeier, "RFC 4322: Opportunistic Encryption using the Internet Key Exchange (IKE)," Dec. 2005.

[13] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "RFC 4033: DNS Security Introduction and Requirements," IETF," RFC, Mar. 2005.

[14] C. Candolin, J. Lundberg, and P. Nikander, "Experimenting with early opportunistic key agreement," in *Proceedings of Workshop SEcurity of Communication on Internet, Internet Communication Security*, Sept. 2002.

[15] T. Finéz, "Backwards compatibility experimentation with host identity protocol and legacy software and networks," Master's thesis, Helsinki University of Technology, Department of Computer Science, 2008.

[16] B. Bishaj, "Efficient leap of faith security with host identity protocol," Master's thesis, Helsinki University of Technology, Department of Computer Science, June 2008.

Publication VI

Kristiina Karvonen, Miika Komu and Andrei Gurtov. Usable Security Management with Host Identity Protocol. In ***Computer Systems and Applications (AICCSA'09)***, Proceedings of the Seventh ACS/IEEE International Conference on Computer Systems and Applications, Rabat, pp. 279 - 286, ISBN 978-1-4244-3807-5, May 2009.

Usable Security Management with Host Identity Protocol

Kristiina Karvonen Miika Komu Andrei Gurtov
Helsinki Institute for Information Technology
firstname.lastname@hiit.fi

ABSTRACT

Host Identity Protocol (HIP) proposes a change to the Internet architecture by introducing cryptographically-secured names, called Host Identities (HIs), for hosts. Applications use HIs instead of IP addresses in transport layer connections, which allows applications to tolerate host-based mobility better. HIP provides IPsec-based, lower-layer security, but the problem is that this type of security is invisible for most applications and users. Our main contribution is the implementation and user evaluation of several security indicators which inform the user when HIP and IPsec are securing the connections of the user. We experimented with application and system level security indicators at the client-side, as well as with server-side indicators. In this paper, we present implementation experience on integrating the identity management Graphical User Interface (GUI) to HIP and results of usability tests with actual users.

I. INTRODUCTION

A host and its location are identified using Internet Protocol (IP) addresses in the current Internet architecture. However, IP addresses can serve only as short-term identifiers as a considerable amount of hosts are portable devices and they change their IP addresses when moved from one network to another. Short-term identifiers disrupt long-term transport layer connections, such as Internet phone calls, and make locating the peer host more difficult. Impersonation attacks are possible because IP addresses are relatively easy to forge.
The Host Identity Protocol (HIP) architecture [26, 23] leverages a so-called *identity/locator split* to address these challenges in an integrated approach. It separates the identity of a host from its location as illustrated in Figure 1. The identity is called the *Host Identity (HI)* and it is used as a long-term identifier on the upper layers of the network stack. The location of host is bound to IP addresses and used for routing packets to the host in the same way as in the current Internet architecture.

The HI namespace consists of *Host Identifiers*, each of which consists of the public key component of a private-public key pair. Each host is responsible for creating one or more public/private key pairs to provide identities for itself. As the HIs are based on public-key cryptography, they are computationally difficult to forge. HIs are location-independent identifiers which allow a mobile host to preserve its transport layer connections upon changes in the network. On the other hand, the HI can be used for looking up the current location of a host because the HI is a long-term identifier. A client host obtains the HI of a server typically from the DNS. However, the infrastructure may not support this in certain scenarios, such as in peer-to-peer and ad-hoc environments. In such cases, *opportunistic* HIP can be used for contacting a peer without prior information of the peer's identity. Opportunistic HIP is based on a “leap-of-faith”, which means that it is prone to man-in-the-middle attacks for the initial connection. It is similar to SSH, where the client caches the public key of the server after the first successful connection.

There are many challenges in making HIP understandable to the end users. As an example, users have developed an automatic response to press “OK” to SSH software prompts (meant to verify and accept the key) without consulting the prompts properly, if at all [3]. This is due to many prompts being uninformative to most users that do not increase the user's security awareness even when read [11]. As we implemented a prompting mechanism for HIP-related connections using our publicly available HIP Firefox2 add-on (available also as a firefox3 extension), we witnessed the same phenomenon.

Most Internet applications can run unmodified over HIP [16], although only HIP-aware (new) applications utilizing the extended socket interface [20] can take better advantage of the new features provided by HIP. As HIP secures application data traffic with IPsec that is located logically "deep" within the networking stack, the challenge is to provide proper and understandable security indicators to the user to convince her that the connection, e.g., to a banking web site, is secured. Such indicators can be developed as extensions to applications (e.g., a security add-on to Firefox browser) or within a host-wide HIP management utility that controls all applications.

When designing the security indicators, it is important to decide how and when to apply users’ existing security habits, and when to break them. For HTTPS, a browser typically illustrates the access to a secure site by a padlock icon or by changing the color of the address bar. However, recent research has shown that these indicators might be ineffective as they go unnoticed by most users [31]. We experimented with the usability of the security indicators with volunteers, who accessed and judged security of web pages. Our first implementation prototype and GUI (Fig.2) was targeted to and usability tested with technical users who are assumed to be the first adopters of HIP. Usability needs to be double-checked in later phases of the development with non-technical users [2].

The rest of the paper is organized as follows. Section 2 gives background on HIP and usable security. Section 3 describes the implementation of our security model for HIP. Usability evaluations are presented in Section 4, followed by results and usability improvements in Section 5. Section 6 concludes the paper with a summary and plans for our future work.

II. BACKGROUND AND RELATED WORK

In this section, we compare the security mode provided by Host Identity Protocol to the familiar model of Transport Layer Security. A short overview of related work on usability of network security completes this section.

A. Host Identity Protocol

In HIP [26], IP addresses are used to route packets, but in the upper parts of the stack the addresses are replaced with Host Identifiers. These Host Identifiers form a new Internet-wide name space for hosts. In HIP, each host is directly identified with one or more public keys that each corresponds to a private key possessed by the host. Each host generates one or more public/private key pairs to provide identities for itself.

For backwards compatibility with networking APIs, applications use shorter representation of the HI. IPv4 applications use 32-bit *Local Scope Identifiers* (LSIs), and IPv6 applications use 128-bit *Host Identity Tag*s (HITs). A HIT is constructed by calculating a digest over the public key. A HIT binds the application to the public keys used for the communications, which is referred as *channel binding*.

The introduction of new end-point identifiers changes the role of IP addresses. When HIP is used, IP addresses become pure topological labels, naming locations in the Internet. One benefit of this identity/locator separation is that hosts in private address realms (behind NATs) can name each other in a unique way with HITs [21]. A second benefit is that the hosts can change their IP address without breaking transport layer connections of applications and rely on HIP to manage host mobility. Thus, the relationship between location names and identifiers becomes dynamic.

The problem of certifying the keys in Public-Key Infrastructure (PKI) or otherwise creating trust relationships between hosts has explicitly been left out from the HIP architecture, as it is expected that each system using HIP may want to take care of it in a different manner. For mere mobility and multi-homing, the systems can work without any explicit trust management, in an opportunistic manner.

HIP uses IPsec as described in [6] to provide data encryption and integrity protection for network applications. Before two network applications can communicate with each other using IPsec-protected traffic, the underlying hosts authenticate each other and negotiate encryption keys for IPsec using HIP [27].

B. Transport Layer Security

Transport Layer Security (TLS) provides security for applications at the application layer. TLS is usually supported by user-space libraries that provide an API for the application to communicate securely with a peer application.

When TLS is applied to existing legacy applications, it requires always rewriting both client and server applications. In addition, it requires the allocation of a new transport layer port at the server side because plain TCP and TLS-based TCP connections cannot use the same port. With HIP, no changes to the application are required and the same transport layer port can be used at the server side. This makes HIP easier to deploy even to binary-only legacy applications for which there is no source code available. As such, HIP can be considered as a means for extending the lifetime of legacy network applications which require security or mobility support.

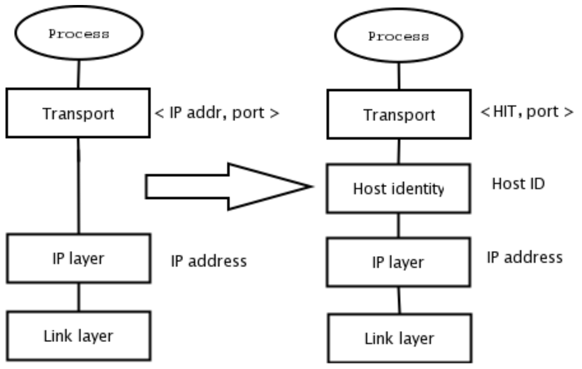


Figure 1. HIP introduces a new layer to the stack

Although there is some on-going effort to make UDP to support TLS [25], currently TLS cannot be used directly with UDP because it assumes a reliable transport protocol. This is problematic especially when VoIP calls need to be protected end-to-end., Fortunately, HIP is applicable also to UDP. HIP supports end-host mobility which is currently missing from the current TLS specification.

C. Managing Security in a Usable Way

The success of any application in managing security depends on its usability; this is also the case with HIP. However, usability and security are often seen as contradicting goals: what is usable cannot be secure and vice versa [9]. A typical example is an easy-to-remember password that tends to be trivial to break, whereas a strong password is hard to recall but security gets breached when users write such passwords down or share them with others [10, 29].

Overall, users can be considered as the "weakest link" in security [1, 29]. If users do not understand the security model behind the user interface, security is at risk. Furthermore, a gap between the mental model of the security experts and non-experts can lead to ineffective and poor communication of how the security works, and what the risks are [5]. Also, in computer security, user errors are in general not acceptable [12]. This leads to the fact that usable security can be described as "usability times two": in security, a single error may be too much, so the generic "trial-and-error" approach will not do. Situation where "the average man" is trying to maintain his security compares to the metaphor of an elephant visiting a store selling objects made of glass - with the lights turned off.

Further, security is usually not the users' primary but secondary goal – an enabler for trustworthy communications of money, private information of personal relationships [11]. Users are not interested or motivated in security *per se*, but rather as means to an end. Because of this, users should be burdened as little as possible with the security features [12], and usage of the security should become a natural part of the actual usage, not an unnatural, add-on extension of the security that introduces an interruption to their primary task, as only too often is the case [11]. For example, unnecessary prompts should be avoided, safe default settings provided, automating as much as possible of the security taking place.

It should also be noted that users do not often realize that they are at risk in a given situation, or what the actual risks are [31] [7]. Users may perceive the risks to be different from what they actually are. However, for the security UI to be successful, it needs to take into account the *perceived* risks or the users do not feel secure. It is a general saying in the field of usability that if the user cannot find functionality, it does not exist. For the UI this means that if the security is not *perceived* to be there, there *is* no security from the user's point-of-view.

A further problem in creating usable security is that users have learned to ignore security indicators. These include usage of padlock icons in the browser address bar and in the lower right corner, the coloring of the address bar, and an extra "s" in the protocol name to show that TLS protocol is being used. There is new work on the browser development side to create standards for web security interfaces [32][30][15]. Unfortunately, they do not work because users do not understand their significance, or the information given is too hard to follow and digest.

Users tend not to know what valid trust marks look like and how to interpret declarations of privacy and how much trust should be induced from their presence [4]. In our previous studies, it became evident that users felt trusting when an image of a visa sign was visible on a given site, falsely inhering that Visa would guarantee their transactions with the site in question. Further, if a user wants the bargain badly enough, he will give up the security [3] [22]. Current security indicators and privacy policy declarations, then, are neither sufficient nor the most usable solution to provide users with information about security [32].

The Extended Validation Certificates [14] approach, intended to overcome some of these obstacles, introduces a new user interface for handling security. However, from usability point-of-view, this scheme seems somewhat problematic, since it includes usage of multiple colors for indicating security. Not only color coding is likely to diminish the overall accessibility, but interpretation and even perception of colors may differ according to cultural variation [8]. It is also difficult for users to identify individual colors in isolation in a reliable way, and the surrounding color scheme of the browser frame and the web page may affect how the color is perceived and how noticeable it is. Furthermore, this type of security indicators may not be noticeable enough either, as e.g. [32] have shown.

III. IMPLEMENTATION DESIGN AND SECURITY MODEL

The user interacts with Firefox web browser and GTK-based HIP GUI. The browser contains an add-on that displays indicators when a connection is based on HITs. The GUI receives notifications on all HIP network communications from the HIP software module. The GUI prompts the user when it is needed: the user can accept or reject new HIP related network connections. Hence, the GUI acts as an end-host firewall for HIP. The user can also use the GUI to sort the server fingerprints to groups as illustrated in Figure 2. The main purpose of the groups is to distinguish between trusted and untrusted peers. Groups can also be used to apply common attributes to all of the fingerprints within the group.

The HIP module translates host names to HITs and provides the browser HIP-based connectivity by intercepting some of the networking related function calls. The module allows varying degrees of authentication for the browser by first trying the strongest authentication method available and then falling back towards weaker authentication methods if the stronger method is not available. At best, the client has obtained the public key of the server already before establishing the connection. This is visible to use both by the browser add-on and the prompt of the HIP GUI. As the second best option, the client tries to establish opportunistic security and learns the public key during the connection set up. This step is visible to the user only by the prompt. Finally, the client uses regular TCP/IP if the server is not HIP capable, which is detected through a timeout. In such a case, no HIP-based security indicators are visible to the user.

It should be noticed that TLS can be used to improve the overall level of security. Thus, the strongest level of security occurs when client uses both HIP and TLS.

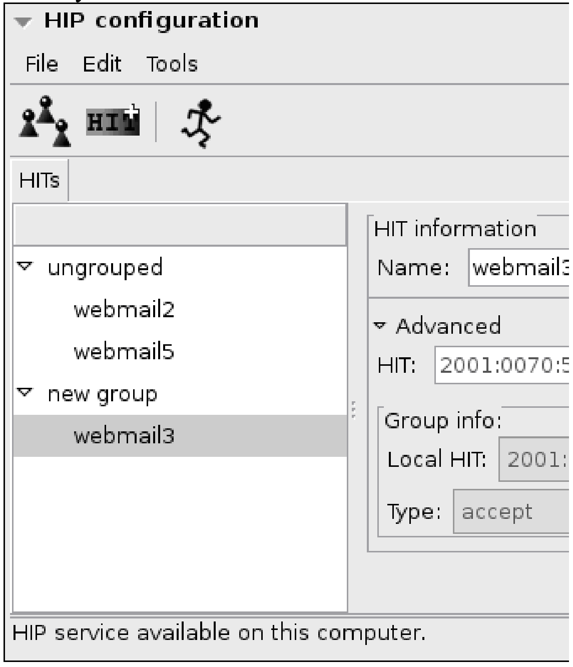
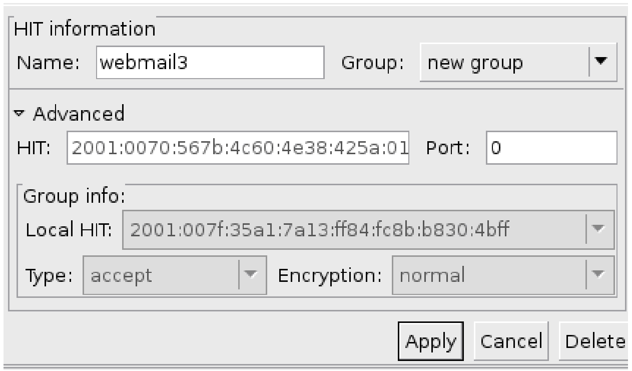


Figure 2a. HIP management GUI navigation tree



HIT information

Name: webmail3 Group: new group

Advanced

HIT: 2001:0070:567b:4c60:4e38:425a:01 Port: 0

Group info:

Local HIT: 2001:007f:35a1:7a13:ff84:fc8b:b830:4bff

Type: accept Encryption: normal

Figure 2b. A close-up on HIP management GUI

IV. USABILITY TESTING OF HIP

The test design was based on refinements made on a round of pilot tests with 10 users with a mock-up of a Finnish online auction site huuto.net, where users sell and buy personal items to peers. Based on user reactions, we changed the application to a mock-up Webmail, in order to narrow down the complexity of the choices and interactions available to better control. We also considered using an online bank as in [32] but gave up this idea since we were not able to create a mock-up design for a bank that would look convincing enough. On basis of existing literature, Webmail account design is less demanding in order to be experienced trustworthy enough to provide valid data about real-enough user reactions as described in. [13], [18], [8]. We were interested in the following questions: Would users notice the security indicators? Which security indicators would users use for judging webmail security? Would users be able to use HITs? How understandable is the concept of HIT?

According to [22], if users know the test is about security, they tend to become more caring and thoughtful about their actions, acting more responsible they normally would and even then, [3] have shown that privacy policy information tends to be noticed. Users have learned to ignore security in real life as it is often incomprehensibly expressed and tediously presented. These test effects were taken into account when designing and analyzing the test behavior and test results.

A. Test Setting

We used HIPL software branch “gui” with patch level 226 in the usability tests. The OS was Ubuntu 6.10 Linux with Linux 2.6.17.14 kernel. The user operated an IBM R51 laptop which was connected directly to another laptop hosting a number of virtual webmail servers (webmail1-5.) The servers were HIP enabled except for webmail1 and webmail4. All of the servers were running apache2 web server.

The tests were conducted in a lab-type environment: a closed, silent meeting room with no outside disturbances. The usability tests with the first group were conducted at the company premises of the participants. The usability tests with the second group were conducted at the premises of the university. In the test, a moderator observed, and if necessary, guided the user through the test tasks. The test ended with an interview. Another person was taking notes. Users took the test one at a time. The test took approximately 30-45 minutes depending on the user’s eagerness to give feedback, talkativeness in the interview section, and speed of interaction in conducting the HIP management GUI test tasks.

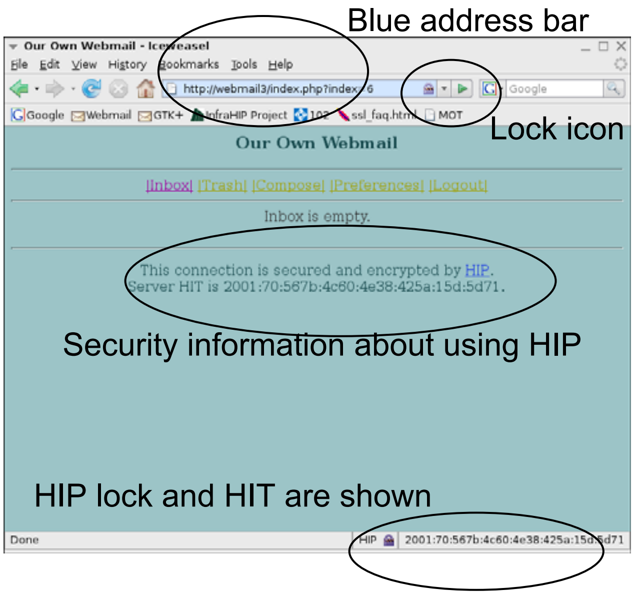
B. Test Users

We had two types of users. **Group 1** consisted of 9 users already familiar with HIP. The users were working for an international network and telecommunications vendor and their work included work on HIP. **Group 2** consisted of 6 users not familiar with HIP, but also this group was technically adept. All users were students or graduates of a technical university, aged between 18-39 years, and familiar with some type of encryption technologies other than HIP. All users were male. No user was color blind.

C. Test Procedure

Users first filled in a background questionnaire (gender, age, average computer usage). Before starting any of the tasks, the users were told shortly about the test setting: HIP was explained to be a new way to provide security in the Internet, providing kind of "fingerprints" of the services used online. Users were also told that they would be logging into email services, and then test out a new user interface to manage the fingerprints HIP created for them during logging into the email services. A talk-aloud protocol was employed: users were asked to tell what they were thinking as they proceeded through the test tasks.

The users would first log into the five Webmail accounts, one by one. The test proceeded from the least secure account login procedure to the most secure. The security indicators were introduced gradually in an incremental fashion. The reason for such a set-up was our hypothesis that users would realize that the security indicators were missing only after their gradual introduction.


Blue address bar
Our Own Webmail - Iceweasel
File Edit View History Bookmarks Tools Help
http://webmail3/index.php?index=6
Google
Google Webmail GTK+ InfraHIP Project 102 ssl_faq.html MOT
Lock icon
Our Own Webmail
|Inbox| |Trash| |Compose| |Preferences| |Logout|
Security information about using HIP
HIP lock and HIT are shown
Done
HIP 2001:70:567b:4c60:4e38:425a:15d:5d71

Inbox is empty.

This connection is secured and encrypted by HIP.
Server HIT is 2001:70:567b:4c60:4e38:425a:15d:5d71.

Figure 3. Webmail 3 site with blue address bar,icons and text that all indicate a HIP secured connection.

This also proved to be the case. The five webmail accounts showed the following security indicators:
Webmail 1 & 4: Insecure connection. There are no security indicators visible, except for the statement in plaintext in the middle of the page saying "This connection is insecure. Please enable HIP".
Webmail 2: The connection is secured with opportunistic HIP. Traditional security indicators are, however, still missing, and the security is stated only in plaintext in the middle of the page saying "This connection is secured and encrypted by HIP". Below the statement, the HIT for the connection is shown.
Webmail 3: HIP module finds the HIT of the server before contacting it and now there are more security indicators visible in addition to the text on the web page: the address bar has turned blue, and there is a picture of padlock both in the address bar and in the lower right corner, with text "HIP" and also the HIT for the site is shown as visualized in Figure 3.
Webmail 5: Both TLS and HIP are used, the address bar has turned yellow, and the same security indicators as in previous case are present. Web server forwarded traffic from HTTP port to HTTPS.

We created random usernames and passwords to be used during the test, instead of asking users to use their own username and password because users have shown reluctance in using this type of personal, private information in test settings in previous tests by us [19] and others [32]. The Webmail addresses and the username and password, of type "username" and "passwd" were presented to the users on paper slips one by one. After logging in, the users were asked to rate the experienced security of the Webmail in question on a scale from 1 to 5, where 1 was considered "insecure" and 5 "secure" We were unsure if users would be willing to use the scale and if they would only use some ratings, but, in fact, it turned out that they used the full scale from 1 to 5, also stating the reasons behind their judgments.

After the Webmail log-ins, the users were asked to complete several tasks with the HIP GUI. The test tasks can be found in **Table 1.** The tasks were described in natural language, e.g. for Task 8 "Can you change the name the group you created earlier?" The word "security" was not mentioned in task description to users in order to avoid bias. Care was taken to see that each user at least tried to accomplish all tasks at some point of the test. With the prototype, not all possible functionalities were available but, they were shown on the GUI to give users some idea of all properties of the security management that GUI could allow for.

Table 1. The usability test tasks (from moderator's perspective). Tasks 1-6 describe the security level and the order in which users logged into the Webmail accounts. Tasks 7-9 are HIP GUI related test tasks.

Task no	**Task content**
1	Log into insecure webmail1
2	Log into webmail2 with opp. HIP
3	Log into webmail3 with normal HIP
4	Repeat task no 1
5	Repeat task 3, no prompt this time
6	Log into webmail5 with TLS and HIP
7a	Find new fingerprint
7b	Create a new group and rename it
7c	Move a fingerprint
8	Rename a group
9	Delete a fingerprint

All of the test sessions ended with a brief interview, where the user could provide feedback in a free fashion. Users were also asked about their real life usage of security after testing the Webmail accounts and the HIP GUI. The questions in the interview part included self-report on:

- what kind of encounters they had had with security;
- what kind of encounters their friends had had with security;
- if were they conducting online transactions, and if so, what were the payment methods they were using;
- if they had had any problems with security before; and if so, what kind of problems;
- if they were interested in security in general, and if so, how would this relative interest/disinterest manifest itself in their behavior.

D. Analysis of the Tests

Figures 4 and 5 show how users evaluated the security of the Webmail accounts and how they succeeded in the tasks related to the HIP GUI management. Figure 4 shows mean and standard deviation of security grades from users in the test: Overall, users evaluated the security level of the HIP protected communications roughly twice as secure as unprotected communications. The case with TLS (and HIP) was rated more secure than HIP communications but the difference was insignificant. Group 1 doubted the security indicators on the web page more than Group 2.

Most users reported awareness of security indicators on websites and claimed they were actively following them. They claimed to be actively following and were familiar with 1) pictures of locks in the browser, 2) changing color of address bar and 3) the 'S' in the HTTPS string, and 4) certificate announcements, all associated with the SSL protocol usage which was trusted by all users in our study. However, in our study, it became evident that in practice this was not really so, as many users did not report that the security indicators were missing from the first Webmail accounts they were shown during the test.

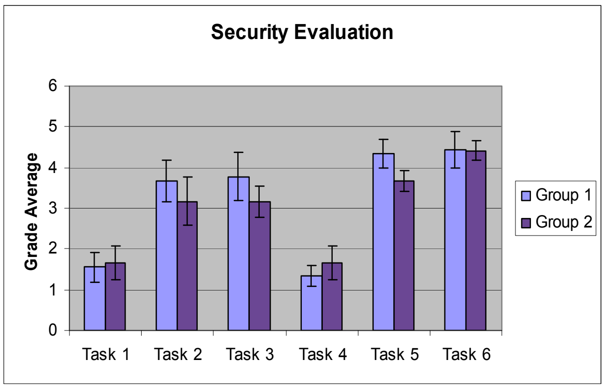


Figure 4. Perceived level of security with the Webmail sites according to user evaluations.

Most users were indulging in online transactions in real life on a regular basis. However, users were not very trusting towards the online service providers they were unfamiliar with. Users reported using several means to protect their assets online, such as a) using only sites they knew well, or b) only making payments via their bank's online services. Some reported also c) having multiple credit cards: one for offline and one for online purchases, with very limited credit limit on the latter in order to minimize the risk. For some of the youngest users in our tests, d) using someone else's card (parents') was one additional way to overcome personal security risks in online situations

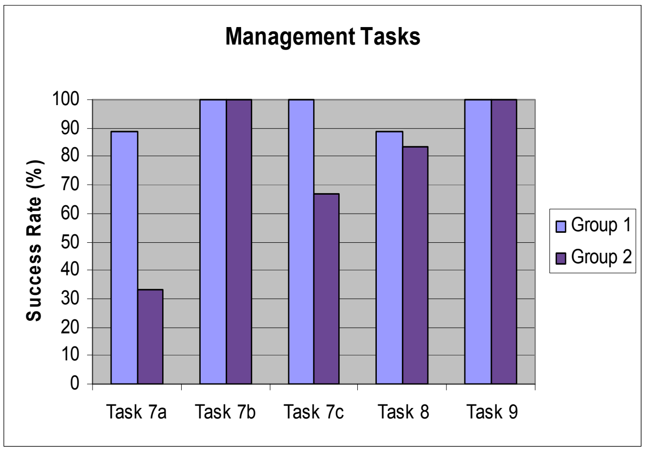


Figure 5. The success rate for management tasks with HIP GUI.

.

Familiarity with the security indicators came up as one of the key ingredients in promoting willingness to feel secure. Users wanted to see something similar to the current SSL implementation also with HIP. This is a natural outcome, as routine ways to deal with new interactions are very often preferred by users - 'old habits die hard'. Yet, even on basis of habitual use patterns, users failed to notice there was a color difference in the address bar; for HIP it was blue, whereas for SSL and HIP-SSL combination it was yellow. This is a remarkable result from the perspective of the EV, since it is to a great extent based on changing the coloring of the address bar to inform the user about the security of the situation. Without educational efforts the users may not notice such indicators, or will misinterpret them. Our users were happy as long as the address bar was colored, *regardless of the color*.

There was a clear difference between the two user groups. The group already experienced with HIP was more actively searching for the security indicators, whereas the other group only rarely noticed these indicators at all. Further, even group experienced with HIP sometimes failed to notice missing security indicators until logging in a webmail which had more security indicators. Only then would they realize that these indicators were not present in the previous webmail accounts they had logged into and considered secure.

There was also a clear gap between what users were actually noticing during the test about security indicators, and what they claimed to be searching for in online situations in real life. So, once again, there was a clear difference in what users claim to do with what they actually do which is typical in usability studies. This is why it is so important to observe users in action and not only rely on their reports of their own actions [28].

Overall, even if users seemed to have learned to look for indicators of security at least to some extent and were involved in online transactions, their attitudes, sources of information and amount of interest in security were surprisingly underdeveloped. The users claimed they would not mind if someone would gain access to their personal e- mails, since "they didn't have anything to hide" – a claim users often abandon once the privacy gets breached. Security was seen as a burden, and users were not really interested in it – they did not follow security news, and did not express worry about bad things happening to them.

V. DISCUSSION

The test setting was not very realistic: a laboratory environment with no disturbance; users were not using their own usernames and passwords. They also knew the test was about security. However, the usability tests revealed a number of areas of improvement in the UI (Figure 6).

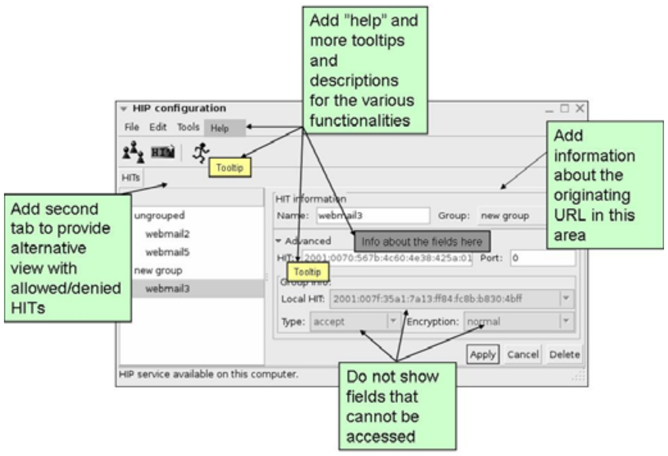

Add "help" and more tooltips and descriptions for the various functionalities
HIP configuration
File Edit Tools Help
Tooltip
Add information about the originating URL in this area
HITs
Add second tab to provide alternative view with allowed/denied HITs
ungrouped
webmail2
webmail5
new group
webmail3
HIT information
Name: webmail3
Group: new group
Advanced
Info about the fields here
HIT: 2001:0070:567b:4c60:4e38:425a:01
Port: 0
Tooltip
Group info
Local HIT: 2001:007f:35a1:7a13:ff84:fc8b:b830:4bff
Type: accept
Encryption: normal
Apply
Cancel
Delete
HIP service available on this computer.
Do not show fields that cannot be accessed

Figure 6. HIP GUI improvement items.

The UI was clearly too technical: users experienced it difficult to understand, "aimed for technical administrators", and most non-HIP users reported they wished they would prefer not manually handling the HITs at all. Further, the traditional security indicators were not efficient: users didn't notice the changing color of the address bar (blue for HIP, yellow for SSL). The *absence* of the security indicators went unnoticed, too. Only the Firefox add-on displayed security indicators when HIP was used in "normal" mode. For the opportunistic mode, the add-on displayed IP addresses instead of HITs. This was the only way for the user to distinguish between the opportunistic and normal modes. However, users didn't notice the change, which means that the users did not notice that they were using leap-of-faith security. A user could have compared the HIT of the server displayed on the web page and noticed that it does not match to IP address displayed by the add-on, but he didn't. The lesson learned is that the lowered security used in opportunistic mode should be informed to the user at least in the prompt.

Overall, the UI concepts were difficult. Users were confused with the concepts of HIT and HIP and explicitly expressed a craving for more information. Even when they were able to learn that fingerprint and HIT were synonyms, the concept of a fingerprint or HIT itself was experienced to be difficult. Especially, differentiating between HITs of the local and peer host was very hard. Further, usage of grouping needs retouching: Users could imagine, when prompted, some possible uses for groups, such as grouping the HITs according to the service or context with which they would be used. However, they didn't at this point at least realize that the groups could be used for indicating which HITs were allowed and which were not. Better visualization of the allowed/denied dimension is probably needed for enhanced usability.

Users were looking for a help menu, and also wanted to have more tooltips and explanatory texts present in the UI. This is indicated that the UI was somewhat immature and technical. For the same reason, UI was seen as "administrative GUI". Further, users were frustrated of being shown fields that they could not access. Some users reported it made them realize how little they actually knew of the technology behind the GUI. In the next version, such fields must be either enabled or not shown to the users.

The familiarity aspect was important: Users liked the HIT announcement to the extent that it reminded them of other types of certificates they were familiar with. Users also expressed an explicit wish for the procedure to be similar to SSL.

Currently, in the navigation panel, there is only one view available for the HITs. However, there is probably need for more, alternating views to the same data. Users may want to organize the HITs according to contents and/or services they are related to, or according to when the HITs are in fact allowed or denied, to enhance personalization.

Further usability improvements include creating suitable icons for the HITs and adding keyboard shortcuts for advanced users in order to support multiple interaction methods.

VI. CONCLUSIONS

On basis of tests it is obvious that a lot of work still needs to be done for the HIP GUI to be truly usable. However, users were able to manage the created HITs with the prototype to the extent that they were able to create and remove groups, and have some idea how they could be used in practice. The identified usability improvements are straightforward to implement and would probably enhance the user-friendliness of the GUI to a great extent – something to be evaluated with another round of usability tests. The differences with the two user groups were relatively small, which may be indication that it is possible to please most users with just one GUI, instead of having several versions for various users.

HIP is based on low-layer IPsec mechanisms which may not be always visible especially to legacy network applications. In such a case, as complete automation may not be the best way to go as users tend to crave for visual confirmation and feedback for security taking place, *prompting* can be used to assure the user that the underlying communications are in fact secured. Alternatively, the client or server software can be modified to show security indicators to the user in a way that is likely to get noticed. We experimented with both of these approaches in this paper.

Our work has further corroborated that the current security indicators do not work. Existing research has shown that users are interested in security only as a secondary goal, as means to an end [11] and do not understand security information when it is provided for them [32]. Still, users may want to know more about security if it is easily available and provided in a way that is understandable [1].

References

[1] Adams, A., Sasse, M.A., Users are not the Enemy, CACM 1999.

[2] Cao, X. and Iverson, L. 2006. Intentional access management: making access control usable for end-users. Proc. of SOUPS '06, vol. 149. ACM Press, New York, NY, 20-31

[3] Grossklags, J., Thaw, D., Perzanowski, A., Mulligan, D., and Konstan, J. (2006), User Choices and Regret: Understanding Users' Decision Process about Consensually acquired Spyware, I/S: A Journal of Law and Policy for the Information Society, Vol. 2, No 2.

[4] Tsai, J., Egelman, S., Shipman, R., Pu, K-C.D., Cranor, L., Acquisti, A (2006), Symbols of Privacy. Poster Abstract.

[5] Liu, D., Asgharpour, F., Camp, L.J, Risk Communication in Computer Security Using Mental Models. Proc. of USEC07 & Financial Cryptography 2007, LNCS.

[6] P. Jokela, R. Moscowitz, Nikander, P. Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP), RFC5202, April 2008.

[7] Camp, L.J, Trust & Risk in Internet Commerce, MIT Press, (Cambridge, MA) (2000).

[8] Cyr, D., Head, M. and Ivanov, A. (2006). Design Aesthetics Leading to M-loyalty in Mobile Commerce. Information and Management.

[9] DeWitt, A. J. and Kuljis, J. 2006. Is usable security an oxymoron? Interactions 13, 3 (May. 2006), 41-44.

[10] Dhamija, R., Perrig, A., Deja Vu: A User Study. Using Images for Authentication. Proc. of the 9th USENIX Security Symposium, August 2000.

[11] Yee, K-P., Guidelines and Strategies for Secure Interaction Design, in Cranor, L.F & Garfinkel, S (Eds.): Security and Usability: Designing secure systems that people can use. O'Reilly Books (2005) 247-274.

[12] Whitten, A, Tygar, J.D (1999), Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. Proc. of the 8th USENIX Security Symposium, August 1999.

[13] Ecommerce Trust Study, Cheskin Research and Studio Archetype/Sapient. 31. http://www.cheskin.com (1999)

[14] Franco, R., Better Website Identification and Extended Validation Certificates in IE7 and Other Browsers. http://blogs.msdn.com/ie/archive/2005/11/21/495507.aspx, Nov. 21, 2005.

[15] Hartman, S., IETF Internet-Draft: Requirements for Web Authentication Resistant to Phishing. http://www.ietf.org/internet-drafts/draft-hartman-webauth-phishing-09.txt, August 2008.

[16] Henderson, T., Nikander, P., Komu M, RFC5338: Using HIP with Legacy Applications, Sep 2008.

[17] Touch, J., Black, D., Wang, Y-S., RFC5378: Problem and Applicability Statement or Better Than Nothing Security (BTNS), Nov 2008

[18] Karvonen, K., The Beauty of Simplicity. Proc. of the ACM Conference on Universal Usability (CUU 2000), November 16-17, 2000, Washington DC, USA

[19] Karvonen, K: Creating Trust, Proc. of the Fourth Nordic Workshop on Secure IT Systems (NordSec'99), November 1-2, 1999.

[20] Komu, M, Tarkoma, S, Kangasharju, J., Gurtov, A., Applying a Cryptographic Namespace to Applications, Proc. of the first ACM workshop on Dynamic Interconnection of Networks (DIN 2005)

[21] Komu et al, Basic HIP Extensions for the Traversal of Network Address Translators, Oct, 2008, Internet draft, work in progress

[22] Kuo, C., Perrig, A., Walker, J., Designing an evaluation method for security user interfaces: lessons from studying secure wireless network configuration. Interactions, 13(3):28-31, ACM Press, 2006.

[23] A. Gurtov, Host Identity Protocol (HIP): Towards the Secure Mobile Internet, ISBN 978-0-470-99790-1, Wiley and Sons, June 2008.

[24] Modadugu, N., Rescorla, E., The Design and Implementation of Datagram TLS, Proc. of NDSS 2004, February 2004

[25] Moskowitz, R., Nikander, P., Host Identity Protocol (HIP) Architecture, RFC 4423, May 2006.

[26] Moskowitz, R., Nikander, P. Jokela, P., Henderson, T., Host Identity Protocol, RFC 5201, April 2008.

[27] Nielsen, J, Usability Engineering, Academic Press, Boston 1993,

[28] Riegelsberger, J., Sasse, M.A., & McCarthy, J., The Researcher's Dilemma: Evaluating Trust in Computer Mediated Communications. International Journal of Human Computer Studies, Vol. 58, (2003) 759-781

[29] Staikos, G., Web Browser Developers Work Together on Security. http://dot.kde.org/1132619164/, Nov. 2005.

[30] Salam, A.F, Rao, H.R., and Pegels, C.C.: Consumer-Perceived Risk in E-Commerce Transactions. Comm. of The ACM, December 2003/Vol. 46, No. 12, (2003) 325-331

[31] Schechter, S., Dhamija, R., Ozment, A., Fischer, I., The Emperor's New Security Indicators, Proc. of the IEEE Symposium on Security and Privacy, May 2000.7