Rendezvous Mechanisms in Host Identity Protocol

Diploma Thesis
by

Miriam Esteban Gómez

submitted to

Prof. Dr. Miquel Soriano Ibáñez

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Departament d'Enginyeria Telemàtica
Universitat Politècnica de Catalunya UPC

21st April 2006

carried out at

NEC

NEC Network Laboratories Heidelberg, Germany

Supervisors:

Dr. Marcus Brunner (NEC)
Dr. Lars Eggert (NEC)

Acknowledgements

This thesis is an outcome of eight exciting months working at NEC Network Laboratories in Heidelberg, Germany. First, I specially want to thank my supervisors at NEC, Marcus Brunner and Lars Eggert, for the chance they gave me to do my thesis at NEC, about such an interesting topic, and the helpful meetings. I can neither forget the useful advices from Simon Schütz, who also helped me during all my stay. I will never forget the weekly NGI coffee, the NGI beer events and all the parties with the NGI people in general.

I also really want to thank my professor in UPC, Miquel Soriano, for offering being my thesis professor during my internship at NEC.

I want to thank as well all the people I met in Heidelberg during these eight months, who made my stay there something I will never forget: Laurinha, Alain, Stefinho, Xavi, David, Albert, Jordi Palau, Miquel, Claudia, Antonio, Jordi Pujol, Alfredo, Justino, Marc Stöcklin, Patel, Alban, Roman, Luca Wullschleger, Luca Vollero, Joao, Zha Wenyi, Sebis, Vlado, Vlad, Paola, Eyner, Paulo, Oleg, Giorgio, Dario, Fabio, Matthias, Vivien, Olivier, Natali and all the rest of the people in and out of the lab who I shared a moment with. Pointing out Dani from all of them, for his constant support in everything I do, and the future stage 2 (K)(L).

I cannot forget all the people I met at the beginning of my university life, who made my lessons attendings much more easygoing: Albert, Inés, Alberto, María, Morata, Dani Vives and Berta among a lot more.

But also the people whom I spent my last years at university with and made me enjoy at the polimenu (and outside from there) many wonderful times: Juanky, Ari, Nando, Aven, Moski, Edgar, Bajito, Jeni, Costa, Ignasi, Dani Vázquez, Esteve, María, César, Ana, Willy, Rober, Bibi, Carmen, Francis, Silvia, Pili, Jaume and Nachete.

And I don't want to forget my parents, Albert and Marisol, and my sister, Tefaní, for their love and constant support, specially these last hard months. A new better life is coming.

Thank you very much.

Miriam Esteban Gómez

Contents

Abstract

The current trend in mobile networking is towards mobile hosts that have multiple network interfaces, e.g., WLAN and UMTS. However, when the current Internet architecture was originally designed, neither mobility nor multi-homing were considered. In the current architecture an IP address represents both the host's identity and the host's topological location. This overloading has led to several security problems, including the so called address ownership [1] problem, making IP mobility and multi-homing unnecessarily hard from the security point of view.

The Host Identity Protocol (HIP), being currently discussed at the Internet Engineering Task Force (IETF), can be used to simultaneously solve the security problems, and many of the practical problems, related to end-host multi-homing and end-host mobility. Basically, HIP introduces a new cryptographic name space and protocol layer between network and transport layers (Host Identity Layer), breaking the fixed binding between identities and locations. The approach is especially suitable for large open networks, where no pre-existing trust relationships can be assumed.

In this thesis we focus on HIP mobility, one of the most important issues within the *Ambient Networks* project (carried out by NEC and many other partners). Our main target is to study which resolution mechanism (mapping Host Identities and IP addresses) performs better: the Domain Name System (DNS), the Distributed Hash Tables (DHTs) or the Rendezvous Server (RVS). Our contributions include: i)implementation of two IETF drafts [4] [5] (developing a Rendezvous Server (RVS) and its Registration mechanism), detailing our early experiences in this issue and ii)a performance evaluation of the RVS comparing it to the other main resolution mechanism, DHT. All our work has been done within the Openhip project, an actively maintained HIP implementation developed in userspace.

[1] Within the current architecture, there is no way of checking that a node claiming to be a given address is actually the node that is indeed located at that address. It leads to several security problems we will address later.

Chapter 1

Introduction: Problem Description and Thesis Goals

1.1 Motivation

When the TCP/IP protocol was originally designed in the late 1970s and early 1980s, it was hardly imaginable that most of the world's computers would eventually be mobile and have several distinct network connections at the same time. Thus, the protocol suite was designed with singly-homed statically located hosts in mind. In that world, the location bound IP addresses served beautifully as identifiers for the hosts, since hosts rarely ever moved between locations.

Years ago, with the introduction of dynamic address assignment in PPP and DHCP, the assumption that an IP address would uniquely identify a host was broken, and the situation was further worsened by the introduction of private IP address spaces and Network Address Translator (NAT). Nowadays it looks like that the emergence of ubiquitous services and ad-hoc networks will soon lead to a situation where the majority of computing hosts are multi-homed[1] and mobile, and have no static addresses.

In addition to the nature of hosts, also the nature of users have changed during the years. For many years, the Internet was basically used by a fairly homogenous user community where everybody more or less trusted every one else. Not so any more. Trustworthiness must now be proved through explicit cryptographic mechanisms.

In a word, the environment has changed. Looking from the 1980s point

[1] A multi-homed end-point is simultaneously reachable at more than one location. Usually this is the result of having multiple interfaces, each separately connected to different location in the network.

of view, the requirements for mobility and multi-homing, together with the host-to-host signalling security, are new. Addressing these within the limitations of the current architecture has turned out to be hard. A wellknown solution for the mobility issue is MobileIP (MIP) but other are also being studied such as the Stream Control Transmission Protocol (SCTP). Regarding the security problem other solutions like IPSEC (IP security) and DNSSEC (Domain Name System security) have been proposed. About multi-homing there are some proposals such as Multi-homed TCP. Combining some of these mechanisms mobile nodes could maintain multiple data sessions, while moving from one IP realm to another, in a secure way.

To include all the requirements of the new Internet architecture in one only solution, it is necessary to do a radical re-engineering which differs from the previously introduced solutions. This is the objective the Host Identity Protocol (HIP), which is currently being developed by the Internet Engineering Task Force (IETF). HIP addresses the issues of mobility, security and multi-homing from a complete different perspective [2], it separates the end-point identifier and locator roles of the IP addresses creating the Host Identities (HI). In this way the IP address will be used as a locator with routing purposes and the HI will be used as a permanent end-point identifier. However, the transition between the current Internet architecture and the proposed by HIP may take a long time, therefore, the compatibility between HIP and the existing protocols is a critical point on the success of HIP.

HIP is still a solution under research, and so there are still several issues pending to be solved. Among the current open issues in HIP there are:

- NAT and Firewall traversal. It is not yet defined the procedure to use HIP between nodes with middleboxes. Some of the current solutions include a registration mechanism that will be explained in detail in Chapter 3.

- Resolution and Rendezvous[2] mechanisms. There is no consensus yet either about what type of mechanism use to map Host Identities and IP addresses nor about the procedure to discover another HIP aware node and stablish a communication with it.

- Registration mechanism requesting for a service (e.g. Rendezvous Server, middleboxes). With this mechanism a node will be able to ask for a service to a provider host. These services include, for instance, helping

[2]Rendezvous mechanism is introduced to help a HIP node to contact a frequently moving HIP node. This mechanism involves a third party, the Rendezvous Server (RVS), which serves as an initial contact point ("rendezvous point") for its clients.

discovering a HIP node and make possible to establish a HIP communication even if there is a firewall or NAT in between.

This thesis has been developed within the framework of the *Ambient Networks* project, carried out by NEC Network Laboratories in collaboration with other 40 partners. In this work we have focused on the mobility aspect of HIP. We have contributed by:

i Implementing and improving, based on our practical experiences, two different Internet drafts: *draft-ietf-hip-registration* ([4]) and *draft-ietf-hip-rvs* ([5]).

ii Developing a prototype, within the OpenHIP project, able to run a Rendezvous Server (RVS), make a HIP node register its Host Identity and IP address and establish a communication between two different nodes through the RVS [27].

iii Providing an insight on the implementation issues of a RVS.

iv Benchmarking the performance of a RVS (from a mobility point of view) comparing it to other possible resolution mechanisms, i.e. Distributed Hash Tables (DHTs).

1.2 Related Work

As mentioned before, some proposals to add mobility, security and multi-homing to the current Internet architecture are being discussed. Unlike HIP, most of them try to solve these problems reusing the existing architecture.

Regarding mobility, in Mobile IPv6 [21] a static address is assigned to each node. Mobile IP does not currently address end-host multi-homing, but there are informal proposals floating around how a single mobile node could use multiple home addresses and multiple care-of-addresses at the same time [19]. Until recently, the largest unsolved problem in Mobile IPv6 was achieving a scalable security solution. The currently proposed solution is based on the ideas of relying on the routing infrastructure to check that a mobile node is reachable both at its claimed home address and its claimed current address (care-of-address): Return Routability (RR). This approach is not very secure, even though it is claimed to be (almost) as secure as the current IPv4 internet. Thus, there are discussions going on about better proposals, e.g. hashing a public key and other information to the low order bits of an IPv6 address.

The mobility issue can also be treated at the Transport Layer like in SCTP [22]. Stream Control Transport Protocol is an IETF proposed standard transport protocol, which may eventually replace TCP and perhaps also UDP. In it, each communication process is associated with several IP addresses. While the SCTP approach is sound as such, the proposed mobility extensions [23] are bound to be plagued with the same security problems that Mobile IPv6 was recently hit. Since SCTP does not include explicit end-point identifiers, solving the security issues in a scalable way may be even harder than with Mobile IPv6.

Regarding multi-homing, the MULTI6 Working Group [24]from the IETF has proposed several approaches to multi-homing for IPv6 in its published RFCs. Apart from this, another group MONAMI6 [25] also investigates multi-homing from the end-point point of view, and not from a site point of view as the term "multihoming" is commonly understood so far.

Within the framework of HIP and this thesis, [1] argues that HIP will benefit from removing its current dependencies on the presence of a deployed DNS infrastructure, resulting in a simpler, more modular system. It also introduces a new resolution and rendezvous service which is further defined in [5]. Moreover, our work follows [4] guidelines, which defines the mechanism to register with a rendezvous service. Apart from this proposal, OpenDHT [15] provides another way to map identities and locations.

1.3 Structure of this thesis

This thesis is structured as follows:

Chapter 2 details the state of the art of the Host Identity Protocol and the several IETF drafts related to HIP. The different resolution problems when using this new protocol are also pointed out as well as some resolution mechanisms are explained here.

Chapter 3 presents our implementation of the Registration process with a Rendezvous Server (RVS) and the implementation of the RVS itself. Some contributions to drafts [4] [5] related to the Registration and Rendezvous mechanisms are also pointed out in this Chapter.

Chapter 4 studies the performance of different resolution mechanisms by testing some specific mobility scenarios in our own HIP network. These scenarios include the two resolution mechanisms identified as the most relevant ones (Distributed Hash Table and Rendezvous Server), which we refer to in Chapter 2.

To conclude, Chapter 5 summarizes the work done in this thesis and discusses possible future work.

Chapter 2

State of the Art

The aim of this chapter is to describe the state of the art of the Host Identity Protocol (HIP). All the technical background provided here is based on several Internet Engineering Task Force (IETF) drafts [6] [8] available during the preparation of this document.

The chapter is organized as follows: Section 2.1 gives a basic view of how HIP works studying the intrinsics of the protocol from three different perspectives: security, multi-homing and mobility. Afterwards Section 2.2 describes the resolution mechanisms designed to work with the current HIP implementations. This Section focuses on three specific resolution mechanisms: Domain Name System (DNS), Distributed Hash Tables (DHTs) and Rendezvous Server (RVS).

2.1 Host Identity Protocol (HIP)

The Host Identity Protocol comes from the need to communicate everywhere anytime securily. For this reason, it was necessary to distinguish between topological locators and identifiers (today both roles played by IP addresses). Thanks to this new approach, IP addresses act only as locators while host identities are the identifiers themselves. This solution, though, requires adding a new layer in the TCP/IP stack, the Identifiers layer, between the Transport layer and the IP layer (Figure 2.1).

One of the issues completely defined in HIP is that the Host Identity (HI) is the public key from a public/private key pair. This key can be represented by the Host Identity Tag (HIT), a 128-bit[1] hash of the HI, and has to be globally unique in the whole Internet universe. Another representation of

[1]Note that 128 bits are the same size as IPv6 addressese so the same number of hosts using this protocol could be allocated.

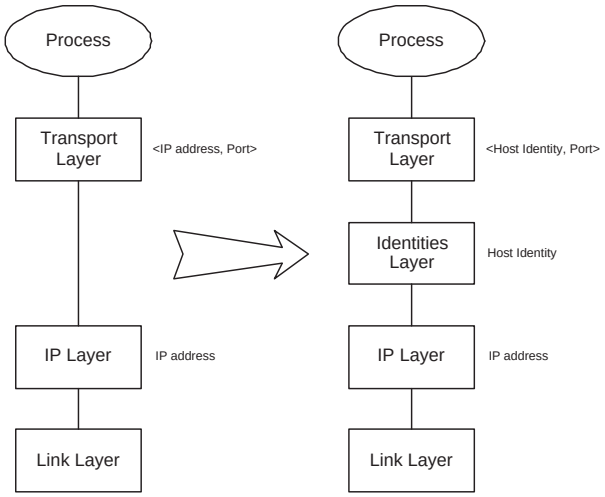


Figure 2.1: Host Identity Protocol architecture

the HI is the Local Scope Identity (LSI) which is 32-bits[2] size and can only be used for local purposes.

We are going to describe the protocol under three main principles: security, multi-homing and mobility, which are introduced in SubSections 2.1.1, 2.1.2 and 2.1.3.

As it may be a long time until HIP-aware applications are widely deployed (even with HIP-enabled[3] systems already upgraded), it is also interesting to know what will happen during this transition. This is the main aim of SubSection 2.1.4.

2.1.1 Security

In the original TCP/IP architecture, the host's identity is implicitly authenticated by the routing infrastructure. That is, since the hosts are identified with IP addresses, and since IP addresses are the fundamental piece of data used in routing, the very definition of the internetwork assures that the IP packets are indeed sent to the intended hosts. In the new architecture, there is no implicit binding between the host identifiers and the routing infrastructure. Thus, the implicit authentication does not exist any more, and must be replaced with an explicit one. Additionally, we must address the

[2]Note that 32 bits is the same size as IPv4 addresses so the same number of hosts using this protocol could be allocated.

[3]HIP-enabled systems support, for instance, a new address family AF_HIP used for creating a socket with a Host Identifier.

problems of address stealing and address flooding. The first attack is performed when a malicious node claims to own an address that some other node is currently using, with the intention of launching a man-in-the-middle or denial-of-service against the actual owner of the given address, while the second attack is referred to a malicious node which makes a large number of innocent peers nodes to believe that the attacker has become available at a target address, causing the peer nodes to flood the target address with unwanted traffic.

The address stealing and address flooding problems are not introduced with the end-point concept or the new host identifiers. Instead, they are originated from the Dynamic binding between the hosts themselves and their IP addresses. Thus, they also exist in environments that use Dynamic IP address assignment: the address stealing and flooding problems are present even in plain vanilla Mobile IP. Fortunately, introducing public key cryptography based on host identifiers that are public keys makes it easier to address these problems. In this section we look at the situation in more detail, starting from the nature of the new identifiers, and continuing with the properties of the new signalling protocols and functions.

Host Identifiers

The cryptographic nature of the Host Identifiers is the security cornerstone of the new architecture. Each end-point generates exactly one public key pair. The public key of the key pair acts as the Host Identifier. The end-point is supposed to keep the corresponding private key secret and not disclose it to anybody.

The use of the public key as the name allows a node to directly check, via an end-to-end authentication procedure, that a party is actually entitled to use its name. Compared to solutions where names and cryptographic keys are separate, the key-oriented naming does not require any external infrastructure to authenticate identity. In other words, no *explicit* Public Key Infrastructure (PKI) is needed. Since the identity is represented by the public key itself, and since any proper public key authentication protocol can be used to check that a party indeed possesses the private key corresponding to a public key, a proper authentication protocol suffices to verify that the peer indeed is entitled to the name. But how can we trust that the authenticated identity we are communicating with is the one we wanted to? If the mapping between a known domain name and an identity (performed in the Domain Name System - DNS) was hacked, then we could be actually contacting with an unknown site, although securily, after the authentication protocol. For this reason, when it is claimed that no *explicit* PKI is needed, it is as long as

a secure protocol is used for the requests to the DNS (e.g., DNSSEC [26]). However, this will be explained in more detail in Section 2.2.

This property of being able to verify the identity of any party without any explicit external infrastructure is the very cornerstone of our new architecture. It allows HIP to scale naturally, without requiring extra administrative overhead and it is accomplished by performing an initial four-way handshake known as HIP Base Exchange (HBE).

HIP Base Exchange (HBE)

The HIP Base Exchange (HBE) can be consider as a cryptographic key-exchange protocol performed at the beginning of the communication. The HIP Base Exchange is built around a classic authenticated Diffie-Hellman key exchange, described in Appendix C.2, used to create a session key and to establish a pair of IPSEC (also defined in Appendix C.1) Encapsulated Security Payload (ESP) Security Associations (SAs) between the nodes. It has to be mentioned that no certificates are required for the authentication because the HITs are self-certifying[4].

Thus, after exchanging the initial HIP Base Exchange (HBE) messages as shown in Figure 2.2, both communicating hosts will know that at the other end-point there indeed is an entity that possesses the private key that corresponds to its Host Identifier. Additionally, the exchange will create a pair of IPSEC Encapsulated Security Payload (ESP) Security Associations (SAs), one in each direction. The hosts are supposed to use the ESP SAs to protect the integrity of the packets flowing between them as well as secure the signalling messages exchanged between the end-points.

By definition, the system initiating a HIP Base Exchange is the Initiator, and the peer is the Responder. This distinction is forgotten once the Base Exchange completes, and either party can become the Initiator in other future communications.

In this section, we focus on the different messages exchanged during the HBE as they are shown in Figure 2.2 and detailed in Figure 2.3 (not all the possible parameters sent in the HBE are represented there).

All HIP packets contain the Initiator and Responder HIT in the header (HIT_I, HIT_R) except from the first one which can be sent in opportunistic mode. That is, if a host receives a start of transport without a HIP nego-

[4]A Host Identity self-certifies as a regular public-private key pair. If someone claims to have a given public key, nobody can pretend to have the same public key unless he knows the corresponding private key; so by keeping your private key, you make your public key self-certifying. Furthermore, it is not computationally feasible to produce a matching HI given the HIT.

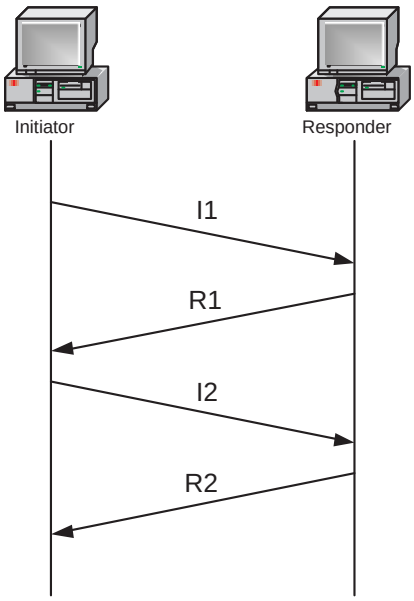


Figure 2.2: HIP Base Exchange messages

tiation, it can attempt to force a HIP Base Exchange before accepting the connection by sending an I1 but with NULL instead of HIT_R, cause it is not known a priori. Note that the full benefits of HIP in security are only gained in case of a known HIT_R, if not, it is almost impossible to defend the HIP exchange against some attacks, because the public keys exchanged cannot be authenticated. The only approach would be to require an R1, answering an opportunistic I1, to come from the same IP address that originally sent the I1. This procedure retains a level of security which is equivalent to what exists in the Internet today.

Coming back to the HIP Base Exchange, the first packet, I1, initiates the exchange, and the last three packets, R1, I2, and R2, constitute a standard authenticated Diffie-Hellman key exchange for session key generation. During the Diffie-Hellman key exchange, a piece of keying material is generated. The HIP association keys (e.g, ESP and HMAC[5] keys) are drawn from this keying material. If other cryptographic keys are needed, they are expected to be drawn from the same keying material.

The HBE is triggered by the Initiator by sending an empty I1 message to the Responder. Before the Responder receives the I1 packet, it has already computed a partial R1 message common for all possible communications with other hosts. This pre-computed R1 includes the HIT_R, the Responder's

[5]Described in Appendix C.4.

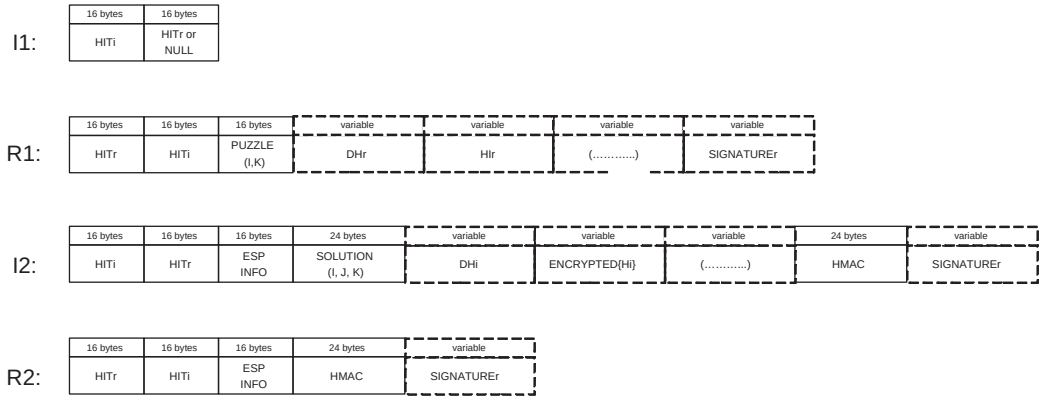


Figure 2.3: Structure of HIP Base Exchange messages

Diffie-Hellman key DH_R and the Responder's Host Identifier HI_R among other possible parameters not mentioned here. The Responder signs this pre-message with its private key. Once the I1 message is received, the Responder completes the R1 with the HIT_I received in the I1 and a $PUZZLE$ field whose level of difficulty will be adjusted based on level of trust on the Initiator.

The $PUZZLE$ parameter in R1 contains a cryptographic challenge which the Initiator has to solve before sending the following I2 message. With this puzzle, the Initiator is forced to perform some moderately brute-force computation before the Responder creates any protocol state. Thus, resource exhausting denial-of-service attacks[6] are prevented by allowing the Responder to increase the cost of the state start on the Initiator and reducing the cost to the Responder. This $PUZZLE$ parameter has three different components: the puzzle nonce I, the difficulty level K and the solution J. The Initiator has to solve the following equation[7]:

$$L_{trunc}(SHA1_{(I|HIT_I|HIT_R|J)}, K) = 0 \tag{2.1}$$

The Initiator must do a brute-force search for the value of J, which takes $O(2^K)$ trials, and this challenge has to be solved in a certain amount of time defined by the Responder also in the $PUZZLE$ field. Increasing K means increasing difficulty and computational cost on the Initiator. On the other hand, the Responder can verify the solution J sent by the Initiator by

[6]DoS attacks that take advantage of the cost of setting up a state for a protocol on the Responder (that is, storing some information) compared to the 'cheapness' on the Initiator.

[7]Find J such that the K lower bits of the hash of puzzle nonce I concatenated with HIT_I, HIT_R and J itself are zero.

computing a single hash.

When receiving R1, the Initiator checks first that it has already sent the corresponding I1 and verifies the signature using the public key HI_R (also included in the R1 or stored in a new Resource Record (RR) in the DNS as it will be explained in SubSection 2.2.1). If the signature is OK, then it solves the puzzle and creates the I2. This message includes the $PUZZLE$ parameter with its solution J, the Initiator Diffie-Hellman's key DH_I, a Security Parameter Index (SPI)[8] for the Responder-to-Initiator IPSEC Security Association (SA) and the Initiator public key HI_I, encrypted using a new session key. This encryption is made with the secret key exchanged during the Diffie-Hellman algorithm and it is done to make sure that the key exchange has been succesful. The message also includes a signature as well as an $HMAC$ parameter which will be checked for the integrity of the message. Key material for the session key (as well as HMAC integrity keys) is computed as a SHA1 hash (defined in Appendix C.3) of the Diffie-Hellman shared secret K_{IR}:

$$KEYMAT = K1|K2|K3|...$$

where [9]

K1 = SHA1($K_{IR}|sort(HIT_I|HIT_R)|I|J|0x01$)
K2 = SHA1($K_{IR}|K1|0x02$)
K3 = SHA1($K_{IR}|K2|0x03$)
...
K255 = SHA1($K_{IR}|K254|0xff$)
K256 = SHA1($K_{IR}|K255|0x00$)
... (2.2)

On receiving I2, the Responder verifies the puzzle solution J. If it is correct, it computes the session keys thanks to the Diffie-Hellman key exchange already performed, decrypts HI_I and verifies the signature on I2. The Responder sends the R2 containing the Security Parameter Index (SPI) for the Initiator-to-Responder IPSEC Security Association (SA), and $HMAC$ (defined in Appendix C.4) computed using the key material, and a signature.

[8] An SPI is an arbitrary value that uniquely identifies which Security Association (SA) to use at the receiving host. The sending host uses the SPI to identify and select which SA to use to secure every packet while the receiving one uses it to identify and select the encryption algorithm and key used to decrypt packets.

[9] Note that sort($HIT_I|HIT_R$) is defined as the network byte order concatenation of the two HITs, with the smaller HIT preceding the larger HIT, resulting from the numeric comparison of the two HITs interpreted as positive (unsigned) 128-bit integers in network byte order.

When the Initiator receives the R2, verifies the *HMAC* and the signature, the HBE is concluded. The *HMAC* confirms the establishment of a session key. For the Responder, the key confirmation is provided by the first inbound IPSEC packet that is protected by the new security association since it cannot be proved that the Initiator received R2 properly until the Responder receives some data.

2.1.2 Multi-homing

Multi-homing refers to a situation where an end-point has several concurrent communication paths that it can use. Usually multi-homing is the result of either the host having several network interfaces (end-host multi-homing) or the network between the host and the rest of the Internet having redundant paths (site multi-homing). From our theoretical point of view, a multi-homed end-host is a node that has two or more points-of-attachment with the rest of the network. This is illustrated in Figure 2.4.

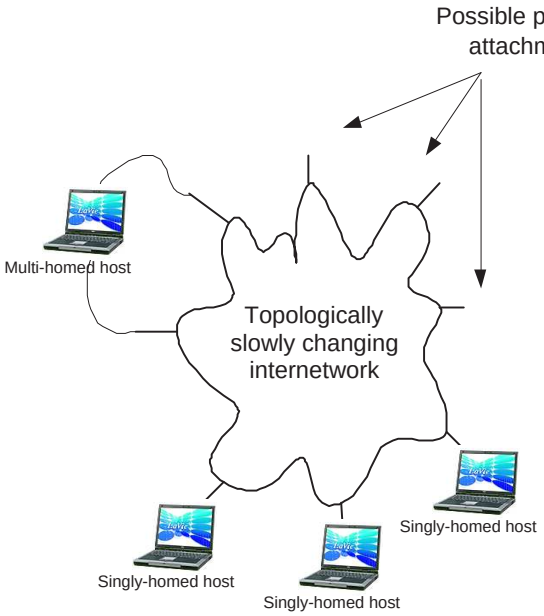


Figure 2.4: The multi-homing model

This situation can be characterized as the node being reachable through several topological paths. The node is simultaneously present at several topological locations. As a consequence, it also has several network layer addresses, each of which reflects one of the topological locations. In the general case, the addresses are completely independent of each other.

With Host Identity Protocol, the host may notify the peer host of the additional interface(s) by using the *LOCATOR* parameter. To avoid problems with the ESP anti-replay window[10], a host will use a different Security Association (SA) for each interface used to receive packets from the peer host.

When more than one locator is provided to the peer host, the host will indicate which locator is preferred. By default, the addresses used in the Base Exchange are preferred until indicated otherwise.

Although the protocol may allow for configurations in which there is an asymmetric number of SAs between the hosts (e.g., one host has two interfaces and two inbound SAs, while the peer has one interface and one inbound SA), it is recommended that inbound and outbound SAs be created pairwise between hosts.

To add both an additional interface and SA, the host sends a *LOCATOR* parameter with an *ESP_INFO*. The host uses the same (or new) Security Parameter Index (SPI) value provided in the *LOCATOR* and if both, the "Old SPI" and "New SPI" values in the *ESP_INFO*, are equal, this indicates to the peer that the SPI is not replacing an existing SPI. The multi-homed host waits then for a *ESP_INFO* from the peer and an ACK of its own UPDATE. Figure 2.5 illustrates the basic packet exchange.

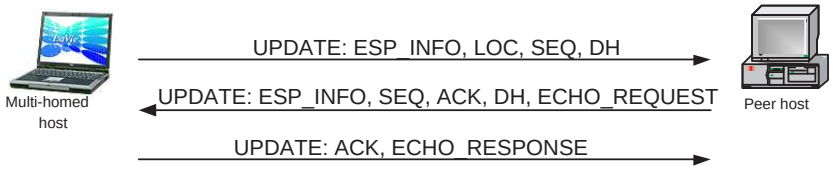


Figure 2.5: Multi-homed host announcing its locations

2.1.3 Mobility

As we said in Chapter 1, in this thesis we will mainly focus on the mobility aspect of HIP. We define mobility as the phenomenon where an entity moves while keeping its communication context active (see Figure 2.6). To move means that an end-host changes its topological point-of-attachment. At the same time, however, we want to make sure that all active communication

[10]IP Security (IPSEC) authentication provides anti-replay protection against an attacker by duplicating encrypted packets and assigning a unique sequence number to each encrypted packet. The decryptor keeps track of which packets it has seen on the basis of these numbers. Because the decryptor has limited memory, it can presently track only the last x packets: the window size.

contexts remain active, and the processes do not see mobility other than, possibly, in changes to the actually experienced quality of service. This is possible because a new address family (AF_HIP) is defined ,then applications can perform system calls to an specific HIT and sockets to this HIT can be created, independently from the IP address where the host is located. Notice that in order to continue to communicate, the host must still signal the changes in its addresses to its active peers. The signalling involved in this process is called Readdressing mechanism.

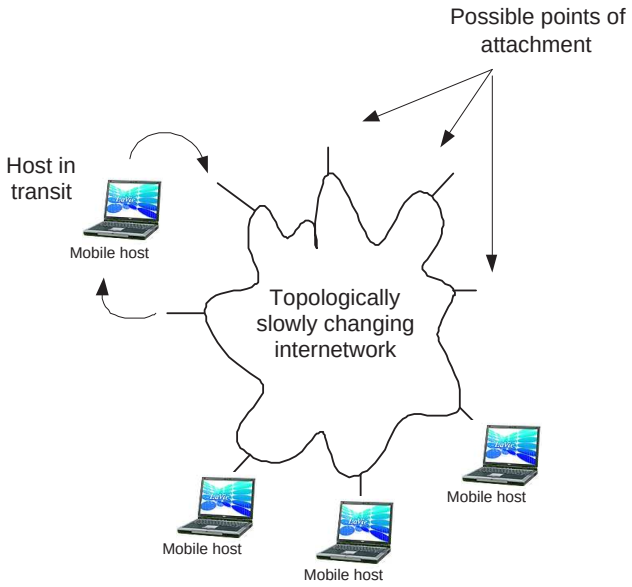


Figure 2.6: The mobility model

Readdressing mechanism

The Readdressing mechanism is triggered when a change of IP address bound to an interface is performed. This might be needed due to a change in the advertised IPv6 prefixes on the link, a reconnected PPP link, a new DHCP lease, or an actual movement to another subnet (changing its topological point-of-attachment in the network).

Currently, there are four different types of readdressing mechanisms and every HIP implementation can perform one, some or all of them taking into account its consequences. These readdressing mechanisms are all triggered the same way which we will follow to explain. Later on, we summarize the complete procedures.

The mobile host is disconnected from the peerhost for a brief period of time while it switches from one IP address to another. Upon obtaining a new IP address, the mobile host sends a *LOCATOR* parameter to the peer host in an UPDATE message. The *LOCATOR* indicates the new IP address and the Security Parameter Index (SPI) associated with the new IP address, the locator lifetime, and whether the new locator is a preferred locator.

The mobile host may optionally send an *ESP_INFO* parameter into the UPDATE packet to create a new inbound Security Association (SA). In this case, the *LOCATOR* contains the new SPI to use. Otherwise, the existing SPI is identified in the *LOCATOR* parameter, and the host waits for its UPDATE to be acknowledged. In the first case, also a Diffie-Hellman key exchange is performed, so then, new key material is created in both Initiator and Responder. That process is called rekeying.

Depending on whether the mobile host initiated a rekey (by sending an *ESP_INFO* parameter with the new SPI and its Diffie-Hellman key), and on whether the peer host itself wants to rekey, a number of responses are possible. Figure 2.7 illustrates an exchange for which neither side initiates a rekeying but for which the peer host does perform an address check by requering an aknowlegment. If the mobile host is rekeying, the peer will also rekey, as shown in Figure 2.8. If the mobile host did not decide to rekey but the peer desires to do so, then it initiates a rekey as illustrated in Figure 2.9 and both rekey again. The UPDATE messages sent from the peer back to the mobile are sent to the newly advertised address.

While the peer host is verifying the new address, the address is marked as unverified. Once it has received a correct reply to its UPDATE challenge, or optionally, data on the new SA, it marks the new address as active and removes the old address.

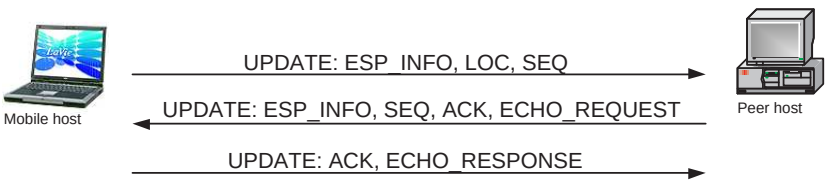


Figure 2.7: Readdress without rekeying, with address check

2.1.4 Compatibility with the current architecture

Fully deployed, the HIP architecture will permit applications to explicitly request the system to connect to another named host by expressing a location-independent name of the host when the system call to connect is performed.

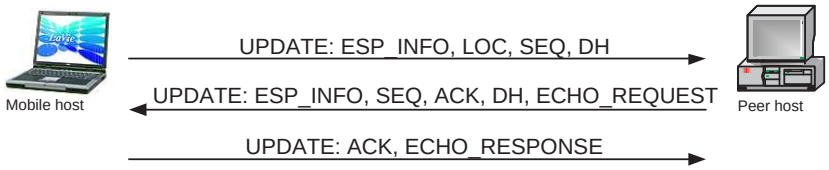


Figure 2.8: Readdress with mobile-initiated rekey

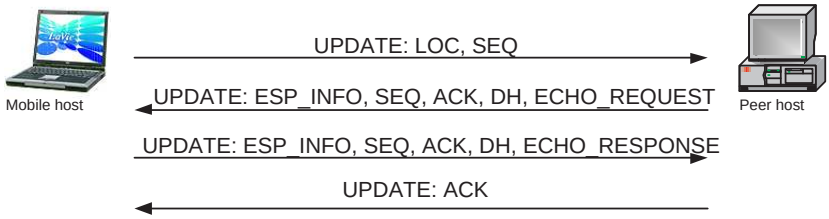


Figure 2.9: Readdress with peer-initiated rekey

However, there will be a transition period during which systems become HIP-enabled but applications are not.

When applications and systems are both HIP-aware [12], the coordination between the application and the system can be straightforward. For example, using the terminology of the widely used sockets API, the application can issue a system call to connect to another host by naming it explicitly, and the system can perform the necessary name-to-address mapping to assign appropriate routable addresses to the packets. To enable this, as it was previously mentioned, a new address family (AF_HIP) is defined as well as additional API extensions could also be defined (such as allowing IP addresses to be passed in the system call, along with the host name, as hints of where to initially try to reach the host).

To use HIP in the situation where the application is not HIP-aware and a traditional IP-address-based API is used instead, a legacy API is defined in [11].

The legacy API is based on the possibility of modifying the system call that returns an IP address to the application, given a domain name, in order to return a HIT instead. The "trick" is that the HIT is 128 bits long, the same length as an IPv6 address, therefore the application will not notice that is receiving a HIT and not an IP address from the resolver. The modified system call would also return the actual IP address to the HIP stack. In case of using IPv4, it is also possible to use a Local Scope Identifier (LSI) that is 32 bits long, the same than an IPv4 address

Thus, supporting the use of HITs and LSIs in place of IPv6 and IPv4 addresses, respectively, will lead to a transparent situation where Transport

Protocols can handle one or the other with no more issues.

2.2 Resolution Mechanisms

Due to the introduction of a new global namespace (the host identities space), HIP also affects the Internet's current resolution services. Thus, the rendezvous procedure and resolution mechanisms are becoming more complex. The various alternatives for performing name and identity resolutions lead to rendezvous procedures that offer significantly different characteristics. In this section, we focus on three different types of resolution procedures: Domain Name System (DNS), Distributed Hash Tables (DHTs) and Rendezvous Servers (RVS).

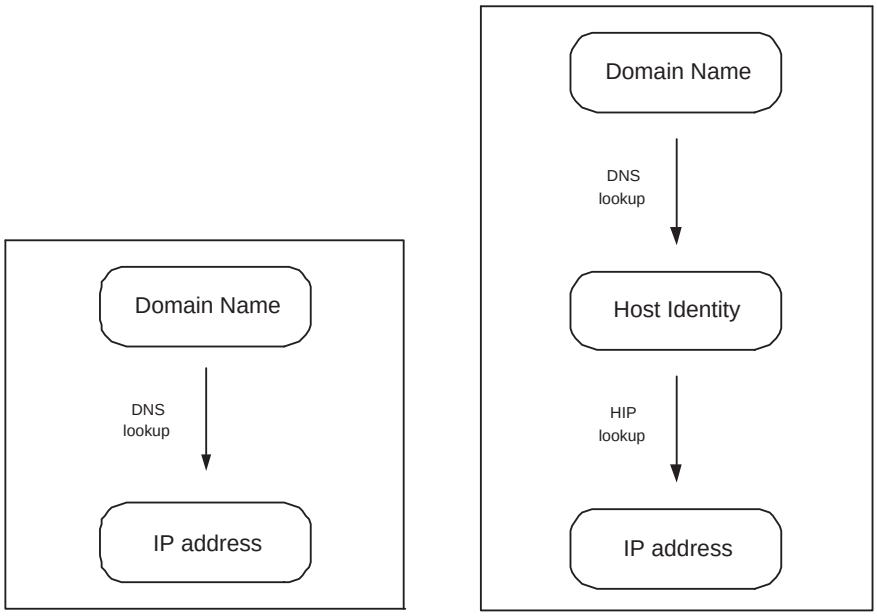
2.2.1 Domain Name System - DNS

The Domain Name System (DNS) is currently the Internet's single global resolution service. The DNS provides a two-way lookup service between domain names (Fully Qualified Domain Names - FQDN) and their set of corresponding IP addresses. However, HIP needs an additional resolution step: domain names (FQDNs) now map into sets of hosts identities (HIs and HITs) which in turn map into sets of IP addresses (Figure 2.10).

The additional HIP resolution step complicates the rendezvous procedure by which two nodes establish a communication channel. In the current Internet, the DNS maps the domain name (FQDN) of a remote node into its set of IP addresses, which the local node may then use to address packets. The address of each node's DNS server is preconfigured. In the absence of a preconfigured DNS server, nodes can still communicate HIP by using IP address directly in opportunistic mode (sending the I1 with a NULL Responder's HIT).

One of the solutions with HIP was creating a new Resource Record (RR) in the DNS [9]: HIPHI RR which stores the HI and the HIT of the registered host (as well as possible Rendezvous Server's domain name as it will be explained in SubSection 2.2.3). But with this new RR you would only get the identity bound to that domain name, that is, the Responder's Host Identifier (HI_R) and Host Identity Tag (HIT_R). The location of the identity (IP_R) is still unknown. Thus a second lookup is needed that resolves the identity to an IP address. This could be done by creating a new mapping in the DNS (HIT to IP address) as shown in Figure 2.11 but this may also mean overloading DNS functions.

Mobility is one of the main principles of Host Identity Protocol therefore



(a) Domain Name resolution without HIP

(b) Domain Name resolution with HIP

Figure 2.10: Domain Name resolution

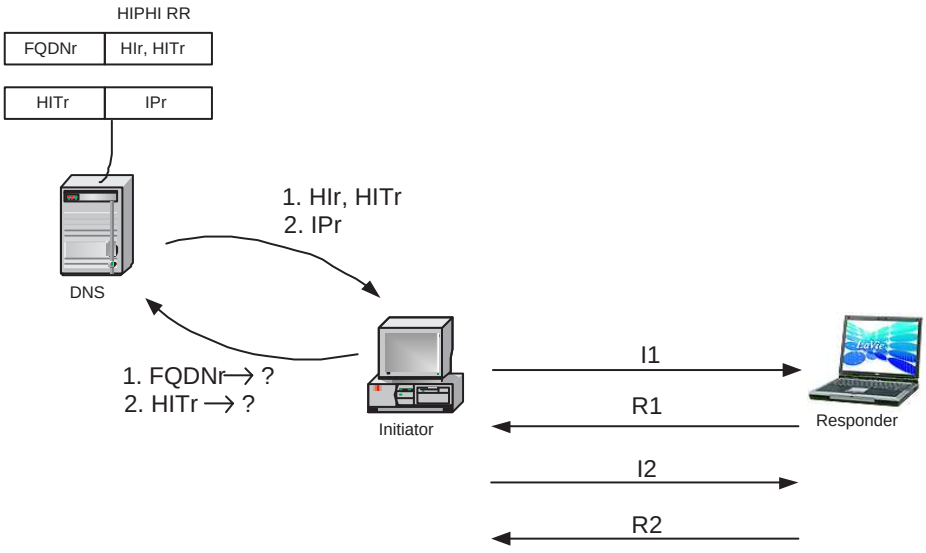


Figure 2.11: HIP Base Exchange using DNS as double resolver.

HIP should support mobile nodes that move rather frequently. This need to support mobility makes the DNS, even including the second mapping (HIT to IP address), a bad candidate to be a resolution mechanism. Although it was allowed Dynamic DNS where clients could update their location, a noteworthy consequence of the distributed and caching architecture of the DNS is that changes are not always immediately globally effective. This drawback essentially leads to an important logistical consideration when making changes to the DNS: not everyone is necessarily seeing the same thing you're seeing. This is because DNS uses cache servers to avoid overloading the primary servers, but the consequence is that when the primary servers are updated, the caches might still be serving the old IP addresses. Actually, when a modification in a store record of a DNS server is performed, there is no control over the amount of time the record is cached so the propagation delay could be too high for frequent mobile nodes.

As conclusion, we can say that DNS is a good choice for the domain name to identity resolution ($FQDN_R$ to HI_R and HIT_R), but not for mapping HIT to IP addresses. The importance of mobility lead us to study new technologies like Distributed Hash Tables (DHT) (which definitely are a better choice for the second mapping rather than DNS), although other new mechanism like the Rendezvous Server (RVS) will also be studied.

2.2.2 Distributed Hash Table - DHT

A better option to use with Host Identity Protocol as a HIP lookup service would be the use of Distributed Hash Tables (DHTs). The Host Identity namespace is flat, consisting of public keys, in contrast to the hierarchical Domain Name System. These keys are hashed to form Host Identity Tags (HITs) which appear as large random numbers.

DHTs are a class of decentralized distributed systems that partition ownership of a set of keys among participating nodes, and can efficiently route messages to the unique owner of any given key. Each node is analogous to a bucket in a hash table. Unlike existing master/slave database replication architectures, DHTs are typically designed to scale to large numbers of nodes and to handle continual node arrivals and failures[11]. This infrastructure can be used to build more complex services, such as distributed file systems, peer-to-peer file sharing systems, cooperative web caching, multicast, anycast, and domain name services.

Each node in a DHT maintains a set of links to other nodes (its neighbours or routing table). Together these form the overlay network, and are picked

[11]When one node leaves, it copies all of its stored information to its predecessor.

in a structured way, called the network's topology. All DHT topologies share some variant of the most essential property: for any key k, the node either owns k or has a link to a node that is closer to k in terms of the keyspace distance defined above. It is then easy to route a message to the owner of any key k using the following greedy algorithm: at each step, forward the message to the neighbour whose ID is closest to k. When there is no such neighbour, then this is the closest node, which must be the owner of k as defined above. This style of routing is sometimes called key based routing.

For instance in OpenDHT [15], an opensource DHT implementation that we have used in this thesis, when a mobile node wants to store a HIT with the corresponing itdata (IP address and a value for the lifetime we want to store the mapping in the DHT), the SHA-1 hash of the Host Identifier (HI) is found, producing a 128-bit key k (the Host Identity Tag (HIT)). Thereafter, a message *put(k,data)* is sent to any node participating in the DHT. The message is forwarded from node to node through the overlay network until it reaches the single node responsible for key k as specified by the keyspace partitioning, where the pair *(k,data)* is stored. One of the problems with the current available OpenDHT is that no HIP signatures are used to validate the *put()* requests, so then, the OpenDHT service does not currently prevent an attacker from polluting the DHT records for a known HIT.

When a client (Initiator) wants to contact with a node which HIT and IP address is stored in a DHT, it can then retrieve the contents of the Responder's HIT by asking any DHT node to find the data associated with the key k (Responder's HIT) with a message *get(k)*. The message will again be routed through the overlay to the node responsible for k, which will reply with the stored *data*.

To handle mobility, when a host which had already registered its HIT and IP address changes its location, it just have to send another *put(HIT, data)* to any node of the DHT. Thus, the updating process in the DHT is really straightforward, without the problems we found with DNS, where updating the caches may be quite slow.

Another issue when using DHTs is detected when two hosts which are communicating via HIP modify their location at the same time. While this change happened, some packets would be lost and the communication may finish.

The complete picture for the resolution would include a DNS lookup to find out the mapping between domain name and identity ($FQDN_R$ to HI_R and HIT_R mapping) and a DHT lookup to find the IP_R from the HIT_R, but then the Initiator must be preconfigured with the IP:port of one member of the DHTs to perform the HIP lookup (Figure 2.12).

The lack of security when no HIP signatures are used to validate the *put()*

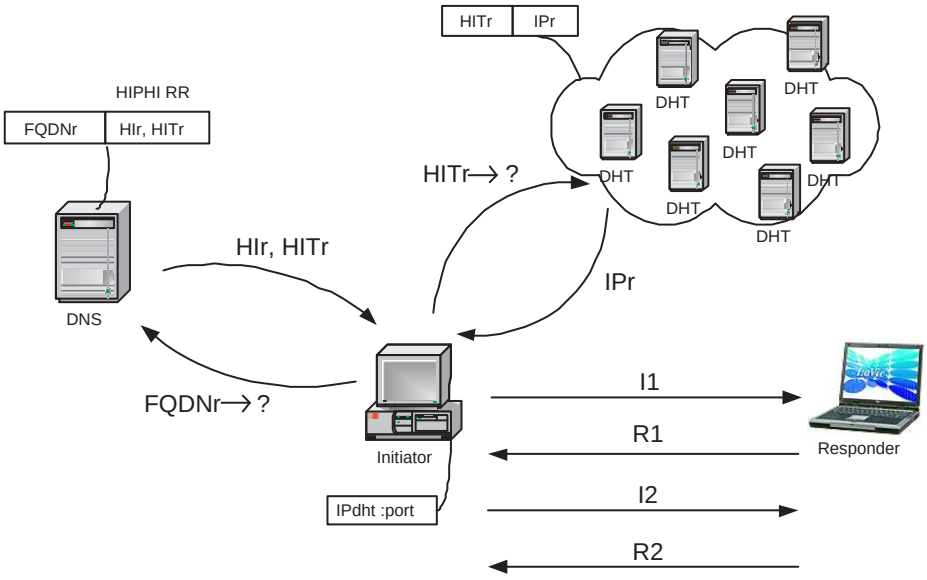


Figure 2.12: HIP Base Exchange using DHT and DNS.

requests and the packet losses due to a double-jump[12] are two problems when using DHTs. Both issues could be solved by Rendezvous Servers acting as resolvers and packet forwarders. But, how does really function a Rendezvous Server?

2.2.3 Rendezvous Server - RVS

In order to start the HIP Base Exchange, the Initiator node has to know how to reach the mobile node. A new possibility is using a piece of new static infrastructure to facilitate rendezvous ("contact point") between HIP nodes: the Rendezvous Server.

The mobile node keeps the rendezvous infrastructure continuously updated with its current IP address(es). The rendezvous mechanism is also needed if both of the nodes happen to change their address at the same time. In such a case, the HIP readdress packets will cross each other in the network and never reach the peer node.

Thus, Rendezvous server (RVS) is designed to support double-jump scenarios - simultaneous host-mobility. Both the initial HIP Base Exchange and the location updates during a HIP session work only, if one of the communicating hosts is stationary and has an unchanging IP address. A RVS functions as a fix point in a network and it keeps track of host mobile nodes.

[12]Two mobile nodes change their location at the same time.

For this to work, the host mobile nodes have to register with their HIs as Rendezvous Clients (RVClient) on a RVS for a Rendezvous Service (RVServ). This means that if an Initiator does not know the IP address of a Responder (mobile node) and the Responder has registered RVServ on a RVS, the Initiator can send the I1 message containing the Responder's HI to a RVS with known IP address. The RVS then relays the I1 message to the Responder as shown in Figure 3.5.

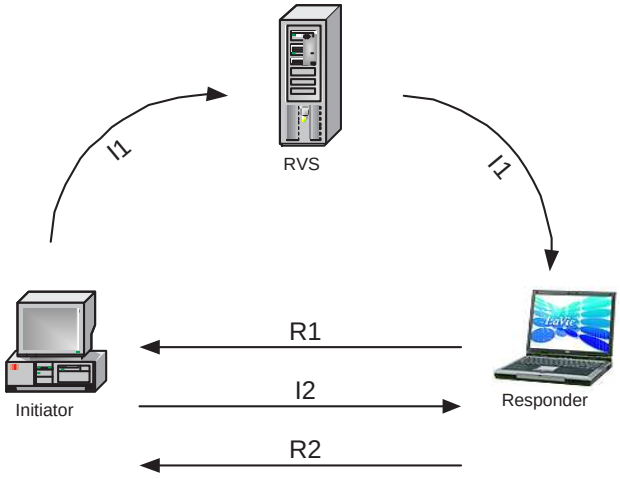


Figure 2.13: HIP Base Exchange with RVS

But how do we know the IP of the RVS where the peer is registered? For this reason it is necessary the addition of a new Resource Record (RR) in the DNS called IPRVS[13]. Thus, the complete procedure includes: i) the registration of the mobile peer's IP address in the RVS (HIT_R to IP_R mapping) and ii) another registration of the mobile peer's domain name and its RVS IP address in the DNS ($FQDN_R$ to HI_R, HIT_R and $FQDN_{RVS}$ mapping). Then, the host requests whatever information from the DNS sending the peer's domain name ($FQDN_R$) and receives the peer's RVS domain name ($FQDN_{RVS}$) and the HI/HIT of the peer itself (HI_R, HIT_R). With this information, it performs another lookup requesting the IP address of the RVS (IP_{RVS}), given the RVS domain name ($FQDN_{RVS}$) provided in the first lookup. Done this, a complete HIP Base Exchange between the Initiator and the Responder can be performed as seen in Figure 2.14, but we will give the detailed information in Chapter 3.

Note, moreover, that the Rendezvous Server is considered a static third party so then the IP address stored in the DNS doesn't have to be modified.

[13]Remember that with DHTs, the IP:port of one member must be preconfigured. There is no extension in the DNS about DHTs.

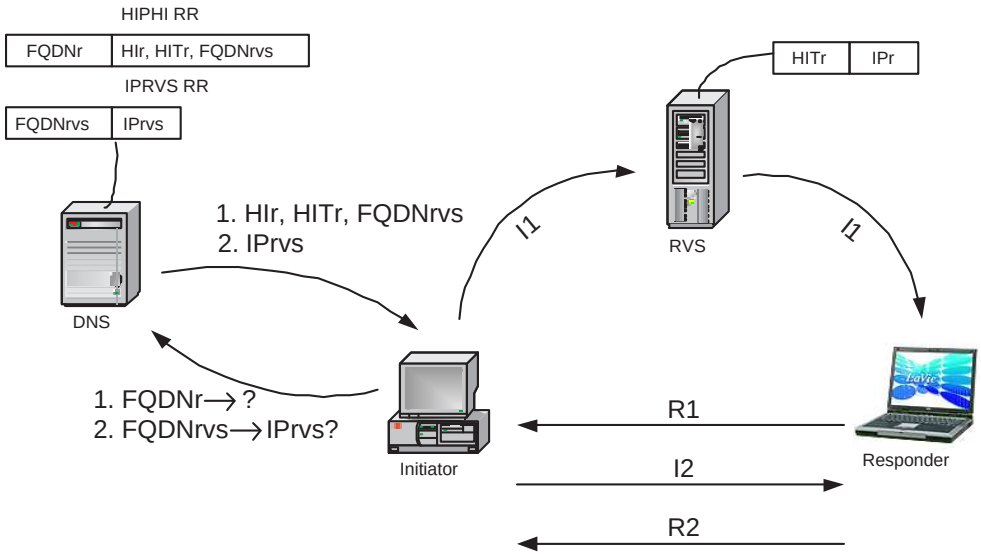


Figure 2.14: HIP Base Exchange with RVS and DNS

However, may be possible that, in the future, new uses for the RVS could be deployed and it would become a not frequent mobile party. In this case, the use of Dynamic DNS should be useful for these infrequent modifications of the RVS's location.

At this point, we claim that the use of Rendezvous Servers is a complete solution for the HIP resolution procedure. However, some performance is needed to assess its suitability, which will be seen in Chapter 4.

Chapter 3

RVS and Registration Implementations

In this chapter we present our implementation of a Rendezvous Server (RVS) and its Registration mechanism following the instructions of the following IETF drafts: *draft-ietf-hip-registration* ([4]) and *draft-ietf-hip-rvs* ([5]).

The Chapter is organized in the following manner: Section 3.1 defines the main concepts which will be used throughout this chapter. Section 3.2 describes in a detailed way how the Registration mechanism works, following the guidelines of draft [4]. In the same manner, Section 3.3 details the mechanism used with a Rendezvous Server following draft [5]. To conclude, Section 3.4 specifies different functionality problems detected when implementing the mechanisms mentioned above, following the guidelines indicated in the former drafts.

3.1 Terminology

In this section we describe some terminology that will be used in this chapter.

- Registrar. A HIP node offering registration for one or more services.
- Requester. A HIP node registering with a HIP registrar to request registration for a service.
- Service. A facility that provides requesters with new capabilities or functionalities operating at the HIP layer. In our implementation we refer to a HIP Rendezvous server, but a service includes firewalls and other middleboxes that support HIP traversal.

- Registration. Shared state stored by a requester and a registrar, allowing the requester to benefit from one or more HIP services offered by the registrar. Each registration has an associated finite lifetime. Requesters can extend established registrations through re-registration[1].
- Registration Type. An identifier for a given service in the registration protocol. For example, the rendezvous service is identified by the specific registration type 1.
- Rendezvous Service(S). A HIP service provided by a Rendezvous Server to its rendezvous clients. The Rendezvous Server offers to relay some of the arriving base exchange packets between the Initiator and Responder.
- Rendezvous Server (RVS). A HIP registrar providing rendezvous service.
- Rendezvous Client (RVClient). A HIP requester that has registered for rendezvous service at a Rendezvous Server.
- Rendezvous Registration. A HIP registration for rendezvous service, established between a Rendezvous Server and a rendezvous client.

3.2 Registration Mechanism

The registration mechanism for the Host Identity Protocol (HIP) allows hosts to register with services, such as HIP rendezvous servers or middleboxes. In our case, it was necessary to implement this mechanism due to the fact that a mobile node willing to be reachable for any other node has to perform a specific procedure with a registrar offering rendezvous service(s) (Rendezvous server).

In this thesis we don't specify the means by which a requester discovers the availability of a service, or how a requester locates a registrar. Once a requester has discovered a registrar, it either initiates HIP base exchange or uses an existing HIP association with the registrar. In both cases, registrars use additional parameters which announce their quality and grant or refuse registration. In the same way, requesters use corresponding parameters to register with the service.

The following subsections will describe the differences between this registration handshake and the standard HIP base exchange [6]. In short, the

[1]We will refer re-registration as an update procedure.

registration process will vary depending on wether a previous HIP association with the RVS exists or not. Figures 3.1 and 3.2 illustrate this situation.

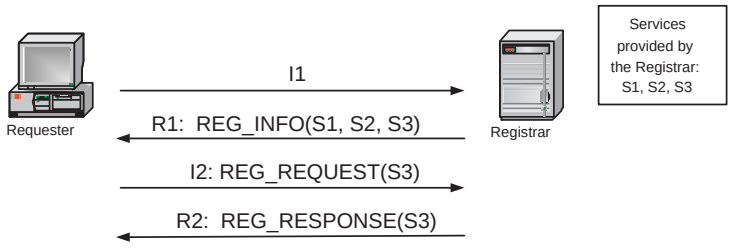


Figure 3.1: Registration with no previous HIP association created.

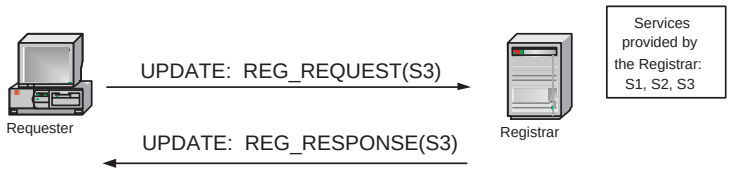


Figure 3.2: Registration with previous HIP association created. Also called re-registration or update process.

The first hipothesis (not previous HIP association created) will be used for subsections: i)Registrar announcing its ability, ii)Requester requesting registration and iii)Registrar granting or refusing service(s) registration. For the last subsection (Establishing and maintaining registrations), we will take for granted that a HIP association is already created (second hipothesis).

Note that messages I1, R1, I2 and R2 shown in Figures 3.1 and 3.3 include the same parameters than in the HIP Base Exchange(HIT_I, HIT_R, the puzzle ...) and some other only necessary for the registration process (*REG_INFO*, *REG_REQUEST* ...) which are explicitly defined here.

3.2.1 Registrar announcing its ability

A host that is capable and willing to act as a registrar (R) includes a *REG_INFO* parameter in the R1 packets it sends during all HIP base exchanges in order to announce its registration capabilities. The registrar can also include the parameter in UPDATE packets when its service offering has changed.

The *REG_INFO* parameter includes the minimum and maximum lifetime permitted by the registrar to provide the different services it has, apart from the services themselves (S1, S2, S3 ...).

3.2.2 Requester requesting registration

To request registration with a service, a requester (RQ) constructs and includes a corresponding *REG_REQUEST* parameter in the I2 it sends to the registrar. This minimizes the number of packets that need to be exchanged with the registrar.

A registrar can finish a HIP Base Exchange that does not carry a *REG_REQUEST* by including a *NOTIFY* parameter with the type *REG_REQUIRED* in the R2. In this case, no HIP association is created between the hosts (Figure 3.3).

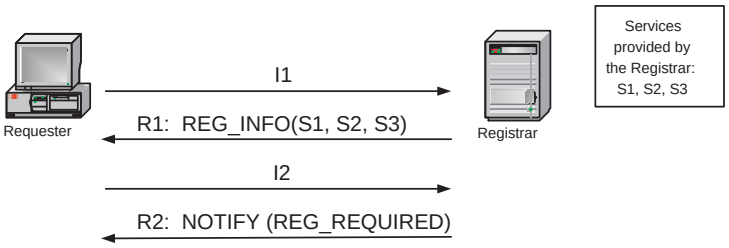


Figure 3.3: Error case during the registration process with no successful HIP association (NOTIFY).

The requester must not include more than one *REG_REQUEST* parameter in its I2 since this parameter specifies the different services a requester wants to join and the lifetime registration it needs for all of them.

When the registrar is requested a registration which lifetime is either smaller or greater than the minimum or maximum lifetime offered by the registrar (and shown in *REG_INFO*), respectively, then it will grant the registration for the minimum or maximum lifetime, respectively. Further explanation on lifetimes and maintaining registered services are in Subsection 3.2.4.

3.2.3 Registrar granting or refusing service(s) registration

Once registration has been requested, the registrar is able to authenticate the requester based on the host identity included in I2. It then verifies the host identity is authorized to register with the requested service(s), based on local policies. The details of this authorization procedure depend on the type of requested service(s) and on the local policies of the registrar, and are therefore out of the scope of this thesis.

After authorization, the registrar includes in its R2 a *REG_RESPONSE* parameter containing the service(s) type(s) for which it has authorized registration, and zero or more *REG_FAILED* parameter containing the service(s) type(s) for which it has not authorized registration or registration has failed for other reasons. In particular, *REG_FAILED* with a failure type of zero indicates the service(s) type(s) that require further credentials for registration. Type 1 means that the requester asked for a registration type which is not offered by the registrar.

If the registrar requires further authorization and the requester has additional credentials available, the requester should try again to register with the service after the HIP association has been established. Note that if HBE has already been performed, this four handshake will not be used to perform another try as in Figure 3.1 since there is already established a HIP association. This is performed under the second hipothesis (HIP association already created) made in subsection Establishing and maintaining registrations as shown in Figure 3.2. Furthermore, this will happen not only in case additional credentials have to be shown to the registrar but in other failure cases and the willingness of the requester of a second attempt.

Successful processing of a *REG_RESPONSE* parameter creates registration state at the requester. In a similar manner, successful processing of a *REG_REQUEST* parameter creates registration state at the registrar and possibly at the service. Both the requester and registrar can cancel a registration before it expires, if the services afforded by a registration are no longer needed by the requester, or cannot be provided any longer by the registrar (for instance, because its configuration has changed).

3.2.4 Establishing and maintaining registrations

Establishing a registration after a failure case (not the NOTIFY one) or maintaining it after the registration lifetime has ended will not perform a HIP Base Exchange again, since there is already an existing HIP association created between the requester and the registrar (Figure 3.2).

Under this hipothesis, the requester will send an UPDATE packet with a *REG_REQUEST* parameter (and the possible correct values for the service) in order to success in the registration of the wanted service. On the other hand, once the lifetime ends, the registrar will inmediately cancel all the services corresponding to that requester unless this last triggers a re-registration, before the services finish, by sending the UPDATE packet with the *REG_REQUEST* parameter as we previously mentioned.

Thus, when the *REG_REQUEST* parameter is inside an UPDATE packet, the registrar must not modify the registrations of registration types which

are not listed in this parameter. Moreover, the requester must not include the parameter unless the registrar's R1 or UPDATE packets have contained a *REG_INFO* parameter with the requested registration types. Thus, a *REG_REQUEST* parameter will only be included in an UPDATE packet if i)the registrar already showed the services it is providing in a *REG_INFO* parameter in the R1 during the HIP Base Exchange (as a second attempt due to a failure or the first update of the service registration) or ii)the registrar already showed the services it is providing in a *REG_INFO* parameter in an UPDATE because some changes happened in its configured offered services.

The requester must not include more than one *REG_REQUEST* parameter in its UPDATE packets and the registrar should include a *REG_RESP ONSE* parameter in its UPDATE packet only if a registration has successfully completed.

The minimum lifetime both registrars and requesters must support is 10 seconds, while they should support a maximum lifetime of, at least, 120 seconds. These values define a baseline for the specification of services based on the registration system. They were chosen to be neither to short nor too long, and to accommodate for existing timeouts of state established in middleboxes (e.g. NATs and firewalls). Specifically, in our RVS implementation, we used one week as a default lifetime.

A zero lifetime is reserved for cancelling purposes. Requesting a zero lifetime for a registration type equals to cancelling the registration of that type. Furthermore, a registrar (and an attached service) may cancel a registration before it expires, at its own discretion by sending a *REG_RESPONSE* with a zero lifetime to all registered requesters.

3.3 Rendezvous Server

This section defines a rendezvous extension for the Host Identity Protocol (HIP) which is used to initiate a communication between HIP nodes. Rendezvous servers improve reachability and operation when HIP nodes are multi-homed or mobile as explained in Chapter 2.

The Host Identity Protocol architecture defined in [7] introduces the rendezvous mechanism to help a HIP node to contact a frequently moving HIP node. The rendezvous mechanism involves a third party, the Rendezvous Server (RVS), which serves as an initial contact point ("rendezvous point") for its clients. The clients of an RVS are HIP nodes that use the HIP Registration Protocol [4] to register their HIT to IP address mappings with the RVS. After this registration, other HIP nodes can initiate a base exchange using the IP address of the RVS instead of the current IP address of the

node they attempt to contact as it was mentioned in Chapter 2. Essentially, the clients of an RVS become reachable at the RVS' IP addresses. Peers can initiate a HIP base exchange with the IP address of the RVS, which will relay this initial communication such that the base exchange may successfully complete.

Figure 3.4 shows a simple HIP base exchange without a Rendezvous Server, in which the Initiator initiates the exchange directly with the Responder by sending an I1 packet to the Responder's IP address, as defined in the HIP base specification [6].

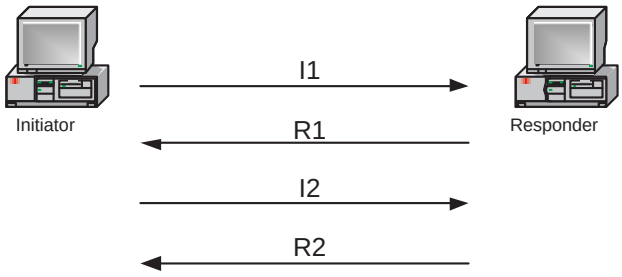


Figure 3.4: HIP base exchange without Rendezvous Server.

Proposed extensions for mobility and multi-homing [8] allow a HIP node to notify its peers about changes in its set of IP addresses. These extensions presumes initial reachability of the two nodes with respect to each other.

However, such a HIP node may also want to be reachable to other future correspondent peers that are unaware of its location change. The HIP architecture [7] introduces rendezvous servers with whom a HIP node registers its host identity tags (HITs) and current IP addresses. An RVS relays HIP packets arriving for these HITs to the node's registered IP addresses. When a HIP node has registered with an RVS, it records the IP address of its RVS in its DNS record, using the HIPRVS DNS record type as we commented in Chapter 2.

Figure 3.5 shows a HIP base exchange involving a Rendezvous Server. It is assumed that HIP node R previously registered its HITs and current IP addresses with the RVS, using the HIP registration protocol [4]. When the Initiator I tries to establish contact with the Responder R, it must send the I1 of the base exchange either to one of R's IP addresses (if known via DNS or other means) or to one of R's rendezvous servers instead. Here, I obtains the IP address of R's Rendezvous Server from R's DNS record and then sends the I1 packet of the HIP base exchange to RVS. RVS, noticing that the HIT contained in the arriving I1 packet is not one of its own, checks its current registrations to determine if it needs to relay the packets. In

case the HIT sent in I1 is not registered in the RVS, it will silently drop the packet. However, if it determines that the HIT belongs to R, it will relay the I1 packet to the registered IP address. R then completes the base exchange without further assistance from RVS by sending an R1 directly to the I's IP address, as obtained from the I1 packet. In this specification the client of the RVS is always the Responder. However, there might be reasons to allow a client to initiate a base exchange through its own RVS, like NAT and firewall traversal but this thesis does not address such scenarios.

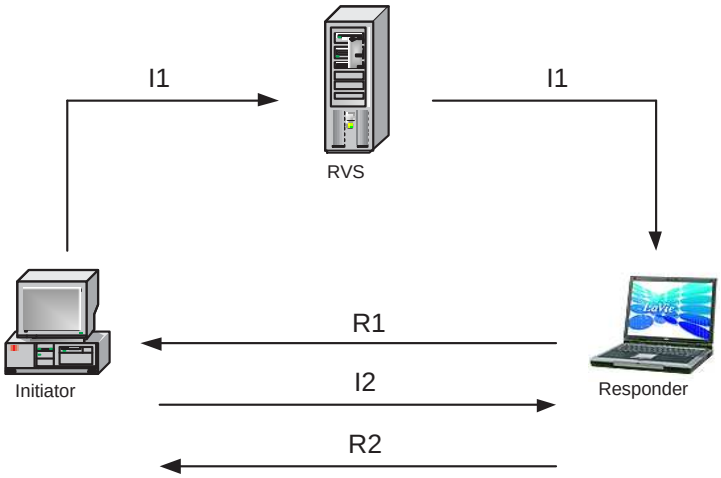


Figure 3.5: HIP base exchange with a Rendezvous Server.

The functions of a Rendezvous Server can be further divided in three main parts: i)Rendezvous client registration, ii)Relaying the Base Exchange and iii)Updating a rendezvous service.

3.3.1 Rendezvous client registration

Before a Rendezvous Server starts to relay HIP packets to a rendezvous client, the rendezvous client needs to register with it to receive the rendezvous service by using the HIP registration extension [4] explained in section 3.2 and illustrated in Figure 3.6. The registration type used in the *REG_INFO* and *REG_REQUEST* is defined as 1 when related to rendezvous service.

3.3.2 Relaying the Base Exchange

If a HIP node and one of its rendezvous servers have a rendezvous registration, the rendezvous servers relay inbound I1 packets that contain one of the client's HITs by rewriting the IP header. They replace the destination IP

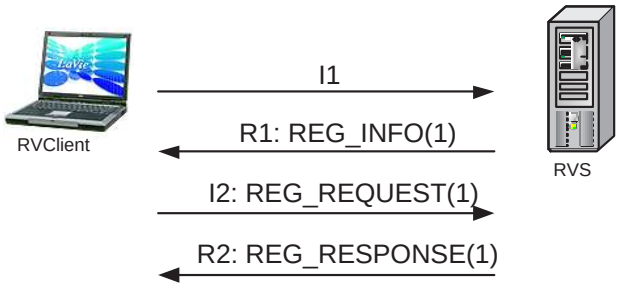


Figure 3.6: Registration mechanism with a Rendezvous server.

address of the I1 packet with one of the IP addresses of the owner of the HIT, i.e., the rendezvous client. They must also recompute the IP checksum accordingly. Because this replacement conceals the Initiator's IP address, the RVS must append a *FROM* parameter containing the original source IP address of the packet.

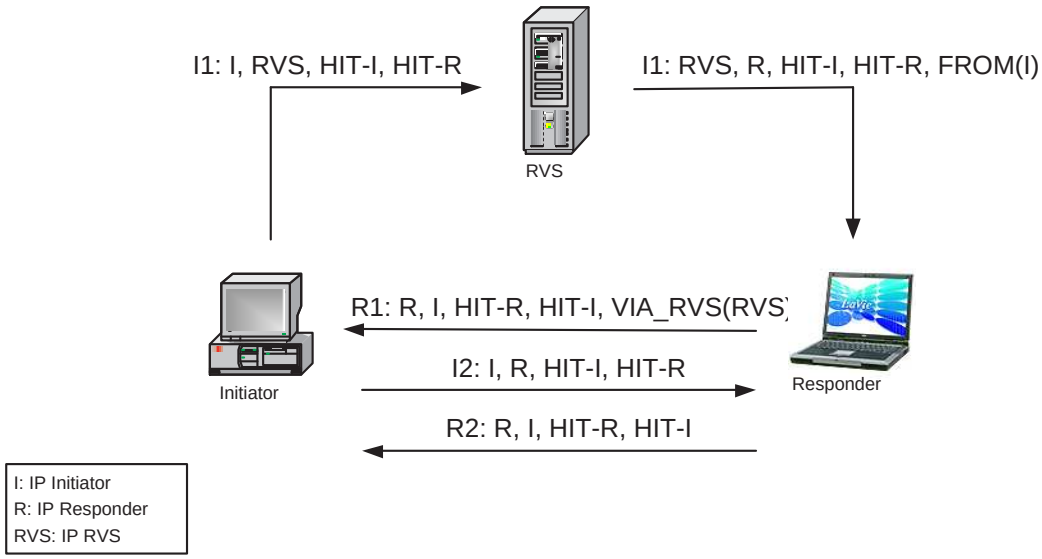


Figure 3.7: Relaying the Base Exchange via a Rendezvous server.

The RVS will verify the checksum field of an I1 packet before doing any modifications. After modification, it recomputes the checksum field using the updated HIP header, which included new *FROM* parameter, and a pseudo-header containing the updated source and destination IP addresses. This enables the Responder to validate the checksum of the I1 packet "as is".

Because the I1 does not include any *SIGNATURE* parameter, these two end-to-end integrity checks are unaffected by the operation of rendezvous

servers.

The following subsections describe the differences of processing of I1 and R1 while a Rendezvous Server is involved in the base exchange.

Processing outgoing I1 packets

An Initiator should not send an opportunistic I1 with a NULL destination HIT to an IP address which is known to be a Rendezvous Server address, unless it wants to establish a HIP association with the Rendezvous Server itself and does not know its HIT.

When an RVS rewrites the source IP address of an I1 packet due to egress filtering[2], it must add a *FROM* parameter to the I1 that contains the Initiator's source IP address.

Processing incoming I1 packets

When a Rendezvous Server receives an I1 whose destination HIT is not its own, it consults its registration database to find a registration for the rendezvous service established by the HIT owner. If it finds an appropriate registration, it relays the packet to the registered IP address. If it does not find an appropriate registration, it drops the packet.

A Rendezvous Server interprets any incoming opportunistic I1 (i.e., an I1 with a NULL destination HIT) as an I1 addressed to itself and will not attempt to relay it to one of its clients.

Processing outgoing R1 packets

When a Responder replies to an I1 relayed via an RVS, it will append to the regular R1 header a *VIA_RVS* parameter containing the IP addresses of the traversed RVS's.

Processing Incoming R1 packets

The HIP base specification [6] mandates that a system receiving an R1 must first check to see if it has sent an I1 to the originator of the R1. When the

[2]Egress filtering allows you to control the traffic that is headed out from your network and restrict activity to legitimate purposes. The addition of a simple rule to your border router and/or firewall allows you to provide a good deal of protection against many categories of malicious activity: "No outbound traffic bears a source IP address not assigned to your network". Adding this rule could mean the prevention of a major headache in the event a malicious individual attempts to use your site as a launching point for a DoS attack.

R1 is replying to a relayed I1, this check will be based on HITs only and in case the IP addresses are also checked, then the source IP address must be checked against the IP address included in the *VIA_RVS* parameter. The main goal of using the *VIA_RVS* parameter is to allow operators to diagnose possible issues encountered while establishing a HIP association via an RVS.

After this happens both the Initiator and the Responder can finish the HIP Base Exchange without further assistance from an RVS as well as communicating sending other HIP packets between them.

3.3.3 Updating and maintaining a rendezvous service

A mobile node which changes its point-of-attachment in the network (IP address) and has several HIP associations with other nodes (including a Rendezvous Server) must make these nodes aware of the modification. This is performed by the readdressing process we already explained in Chapter 2.

Updating the change to a non-RVS node means changing the mapping between HIT and IP in the HIP association between them. Apart from this, updating a RVS node will also modify the IP address in the mapping HIT to IP in the HIP registration service so then the mobile node will be reachable to possible new HIP associations with other nodes via its RVS. This is shown in Figure3.8.

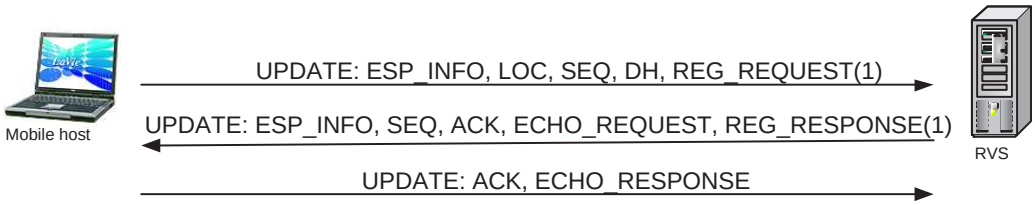


Figure 3.8: Updating a change of point-of-attachment to a Rendezvous server.

To maintain registration after the lifetime assigned by the RVS has ended, it will not be performed a HIP Base Exchange again, since there is already an existing HIP association created between the RVClient and the RVS. As explained in Section3.2, the RVClient will send an UPDATE packet with a *REG_REQUEST* parameter with an appropiate lifetime and registration type 1 to the RVS (before the lifetime of the registration has completely expired). The RVS will respond with another UPDATE packet including a *REG_RESPONSE* parameter if the registration completed successfully or a *REG_FAILED* if there was any error. This is shown in Figure 3.9.

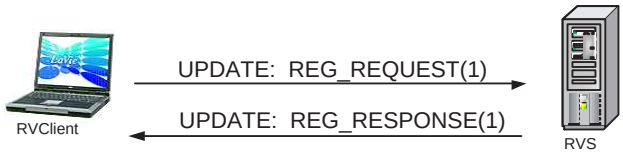


Figure 3.9: Maintaining a registration in a Rendezvous server.

3.4 Contributions

While implementing the code related to the registration mechanism and the Rendezvous Server, several issues had to be solved. Some of them only implied modifications of the HIP implementation we were using for our experiments (Openhip), but some other also became modifications in some definitions of the drafts [5], [4] and [6]. These changes didn´t modify the HIP essence; actually, they referred to details found while implementing an extension of the HIP protocol never implemented before.

The first problem we found was related to the HIP state machine detailed in the Appendix B. Imagine that a registration process between a host and a Rendezvous Server ends with an error: the RVClient receives an R2 message with a *REG_FAILED* parameter. As it was explained in Section 3.2, the RVClient may want to do a second attempt to register with the RVS. In this case, since the HIP association is already established (although not the HIP registration), no HIP Base Exchange will be used again. Instead, a two-way handshake UPDATE process will be performed in which the RVClient sends first an UPDATE with a *REG_REQUEST* parameter and the appropiate values to be able to register this time.

Following the definition of the HIP state machine, a node can only receive an UPDATE message in case it is in the ESTABLISHED state. As anyone could suppose wrongly, once the HIP Base Exchange has successfully completed, only the Initiator (in our experiment the RVClient) has transitioned to this state, while the Responder (RVS) is still in the R2-SENT state. This is due to the fact that the Responder at this moment has no way to know that the Initiator has received correctly its R2 packet. Because of this, the Responder will only transit to state ESTABLISHED in case it receives data from the Initiator (via the Security Association (SA) created between them) or a timeout preconfigured has passed by.

Under the hipothesis of our experiment, the Initiator (RVClient) will send an UPDATE packet to the Responder (RVS) while this one is still in the R2-SENT state. For this reason, and upon agreement with the IETF Working Group, it was decided to create a new transition between R2-SENT and

ESTABLISHED states whenever an UPDATE packet is received. Doing it this way, it was not necessary hardcoding our implementation with values only acceptable for it. The complete process is shown in Figure 3.10.

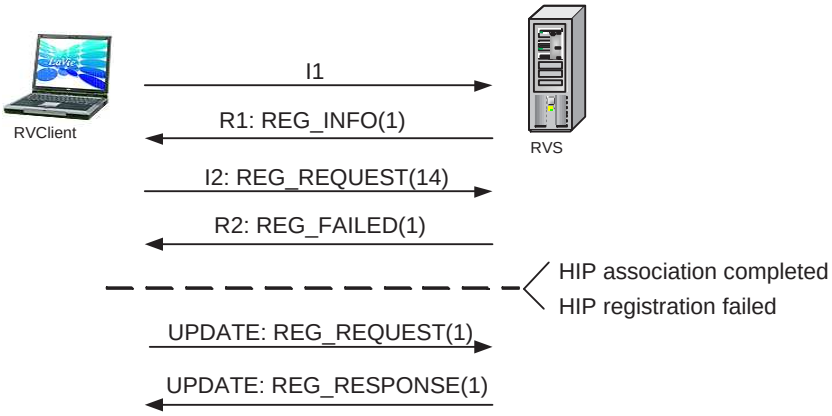


Figure 3.10: Establishing a registration with a Rendezvous Server after a failure in the first attempt.

Some other issues, less significant than the above mentioned, were defining new error cases while our implementation was being tested.

It was quite clear that if the Requester asks for a service which is not provided by the Registrar, this last one will answer with an error included in the parameter *REG_FAILED*. In the current draft, there was only one error defined which was the one generated when requiring further credentials for registration (error type 0). Thus, it was added another error type (now, error type 1) to include all the processes which include the situation explained before.

Upon agreement with the IETF Working Group, it was decided not to create a new error type in case the Registrar is requested a registration which lifetime is out of the range provided by it. Instead, it was agreed that if the requested lifetime is smaller than the minimum one offered by the registrar, it will grant the registration for the minimum lifetime. Respectively, if requested lifetime is greater than the maximum one offered by the registrar, it will grant the registration for the maximum lifetime.

Chapter 4

Performance Evaluation

The aim of this chapter is to evaluate the efficiency and signalling costs introduced by the Host Identity Protocol (HIP) and the different rendezvous mechanisms studied so far. We will mainly focus on mobility scenarios: which resolution mechanism is able to perform a readdressing in less time? That is a critical parameter in the performance of handovers, and therefore has a direct impact on the quality of the communications, e.g. VoIP, that will run over HIP.

This chapter will show the pros and cons, and the performance mainly in terms of delays introduced, when we have two hosts communicating without HIP, when these two hosts communicate directly via HIP, or when these two hosts communicate via HIP and use a resolution mechanism. Only the mapping HIT to IP will be considered as a part of the resolution mechanisms and the study presented here will focus on DHTs and RVSs as possible candidates for HIP resolution mechanisms. The solution based on DNS is discarded because it clearly fails as HIP resolution mechanism as it was explained in Chapter 2.

In the previous chapters we have been seeing the different benefits that HIP introduces but, of course, there is a price to pay in terms of signalling. For instance, HIP introduces a new handshake at the beginning of the communication, the HIP Base Exchange, or new procedures to register with a regitrar or to perform a readdressing. This Chapter will evaluate the delay introduced by these new handshakes. In order to do so, we define four basic scenarios and we test the performance of the handshakes allowed in each of them, see Figure 4.1:

1 Scenario A: No Host Identity Protocol (HIP) used. Experiments performed: RTT measurement.

2 Scenario B: HIP with no resolution mechanism for the mapping HIT to

IP address. Experiments performed: HIP Base Exchange, Readdressing and RTT measurement.

3 Scenario C: HIP with Distributed Hash Table (DHT). Experiments performed: HIP Base Exchange, Readdressing, Registration and RTT measurement.

4 Scenario D: HIP with Rendezvous Server (RVS). Experiments performed: HIP Base Exchange, Readdressing, Registration and RTT measurement.

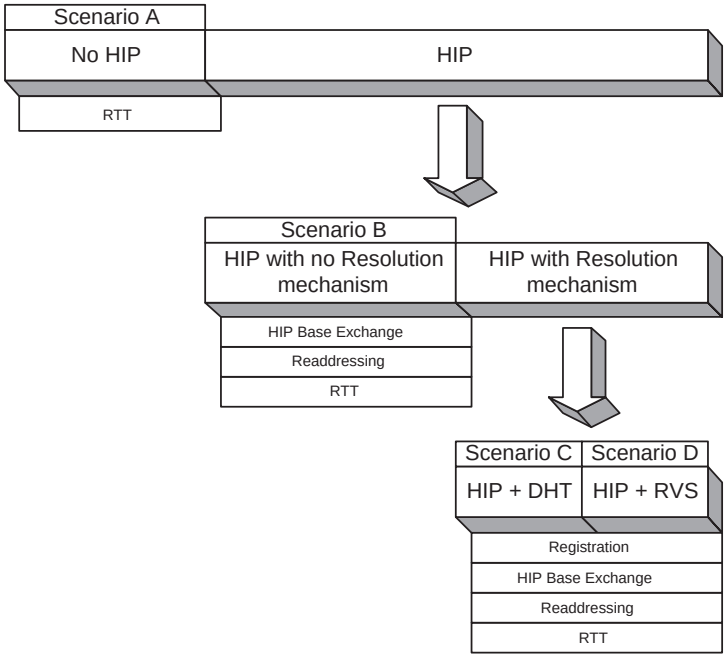


Figure 4.1: Experiments performed in each Scenario evaluated.

This Chapter is organized as follows: first we mention the software used in this performance evaluation[1]. Then, the general setup and the description of the different experiments are introduced. Finally, the performance of each one of the experiments is described and we discuss the results by looking first at every escenario and later discussing later all of them, from a global point of view.

[1]Notice that even though the concrete values shown in this chapter are completely dependent on the software we have used, the general tendences that we show are valid because they depend on the structure of the protocol and not on our implementation.

4.1 Software

This section shortly introduces the software that has been used to perform the tests described in this chapter. This includes the HIP implementation itself and some other tools.

OpenHIP

All tests are based on the OpenHIP project [14], which was the chosen implementation to develop the Rendezvous Server and Registration extensions. This thesis used the 0.3 release for its experimental evaluation since it was the one used in the development process. It was used the kernel patched mode of OpenHIP, which is based on a modified Linux 2.6.11 kernel with the HIP-layer implemented and a HIP daemon running over it. The extensions developed as part of this thesis can be found in the release 0.3.1 of the OpenHIP software.

OpenDHT

OpenDHT [15] is a publicly accessible Distributed Hash Table (DHT) service. However, we used it as a private DHT in our own HIP network. This was possible thanks to an extension in the OpenHIP project done in the 0.3.1 release. OpenDHT's simple put-get interface is accessible with HIP over XML RPC[2].

Click-router

This is part of the Click Modular Software Project [16]. Click routers are flexible and configurable routers. Actually, a Click-router is an interconnected collection of modules which control every aspect of the router's behaviour, from communicating with devices to packet modification to queueing, dropping policies and packet scheduling. We used this tool by patching a Linux 2.4 kernel and, basically, to simulate the delays in the links between the different hosts used in the experiments.

Ethereal

To examine the traffic generated by the OpenHIP implementation, a modified version of Ethereal [17] has been build. Ethereal is a protocol analyser and

[2]XML-RPC is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism. It is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.

traffic recorder. It can be used to record traffic which can be analysed later on. The OpenHIP implementation developers have published a patch for Ethereal 0.10.10, so it is possible to identify the different HIP packets and parameters included.

MatLab

MatLab [18] is a numerical computing environment and programming language. Created by The MathWorks, MATLAB allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. It was used to parse and analyse all the data from the tests results.

4.2 General Configuration and Setup

This section introduces specific configuration details of the different scenarios and experiments. These general characteristics are common for all the tests. If any experiment has a specific setup, it will be detailed in the section referred to that experiment. This section is further divided into: network, machines, HIP configuration and resolvers.

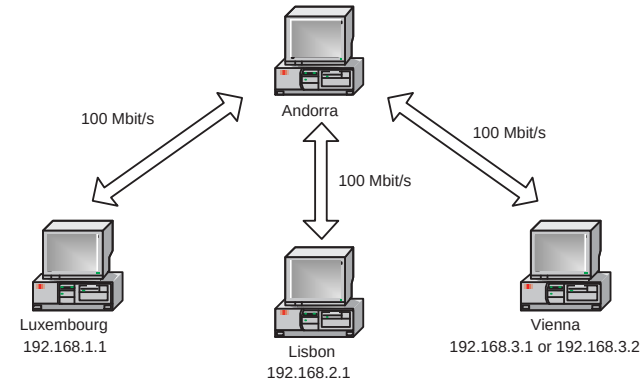
Network

The HIP network that we designed was also connected to our lab network, so we could access from anywhere via ssh, for instance. All the computers in our HIP network had static IPv4 addresses and all the interfaces used were 100 Mbit/s ethernet cards. The node acting as mobile node, which had two different interfaces (eth1 and eth2), had also static IPv4 addresses in each interface. To simulate mobility, we switched the connection from one interface to the other.

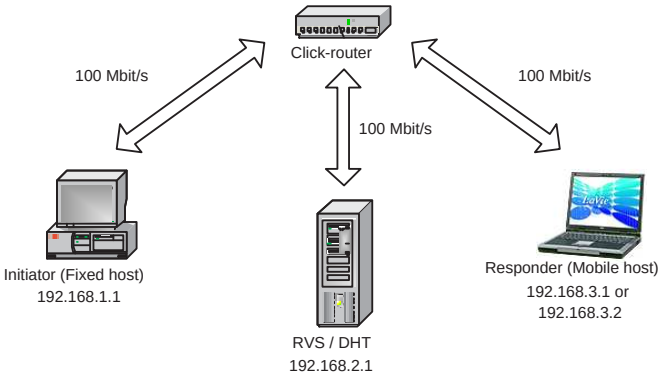
Machines

Depending on the scenario performed, we used a different number of computers. Thus, three of them always acted as HIP nodes while another one was the click-router. Below there is a brief description of the characteristics of each machine depending on the function they performed (also illustrated in Figure 4.2).

- HIP nodes:



(a) Physical tests setup



(b) Machine functions

Figure 4.2: Correspondance between machines and functions.

These three machines were clones (in hardware). They all had a Suse Linux 2.6.11 kernel modified with the Openhip 0.3.1 patch, and Ethereal 0.10.10 also patched. Each machine played a fixed role in all the scenarios as explained below:

  - Initiator (Fixed node): Luxembourg.
  - Responder (Mobile node): Vienna.
  - RVS/DHT (depending on the scenario): Lisbon. This machine also included the OpenDHT implementation for the DHT role in Scenario C.

- Click-router:

  Performed by Andorra. It had a vanilla Linux 2.4 kernel installed in a Suse Linux 2.4 kernel. The Click-router implementation was installed as a module. Basically, we used the click-router to modify the delays between the different hosts to simulate different situations as it will be explained in the next Section.

HIP configuration

Prior to the tests, computers acting as HIP nodes computed their own HI and HIT that were used throughout the tests. This is done by executing the hitgen script. First *hitgen* generates a HI and its respective HIT and stores them in */etc/hosts/my_hosts_identities(.xml)* file. Then, *hitgen -publish* copies these HIs and HITs in the file */etc/hosts/known_hosts_identities(.xml)*, which will be exchanged with other HIP nodes so that each node is aware of every node's identity. Also *hitgen -conf* creates a default configuration file for the HIP communication.

After this, we had to configure IPSEC policies between hosts. For instance, we configured IPSEC ESP between all the HIP nodes (that is, all except from the one acting as click-router) in scenarios A, B and D. In Scenario C, taking into account that the DHT configured does not communicate via HIP with the rest of nodes, no IPSEC policies have to cover its communications with the rest.

Resolvers

Regarding to the machine whose function is Rendezvous Server (RVS) or Distributed Hash Table (DHT), we must mention that both had a preconfigured

load in all the experiments of 1000 entries[3]. In order to fill the registration table in both type of resolvers we used different methods which are explained below.

- In order to fill the registration table in the RVS, we used the file */etc/hosts/registered_hosts(.xml)*, which contained random HITs and IPs with their corresponding random lifetime. This file was configured in the server before starting the experiments and read at the beginning of the RVS run. Note that the RVS comes from a HIP node with special funcionalities and requirements.

- About the DHT, we programmed a bash script to automatically send *put* requests with HITs, IPs and lifetimes associated before the tests started.

4.3 Experiments

In this evaluation we focus on basic HIP properties. We investigate the establishment of a HIP connection (looking at the HIP Base Exchange), the registration of a service in a Rendezvous Server (looking at the Registration mechanism), the maintenance of a HIP connection when there is a change of location of one host (Readdressing mechanism) and the overhead introduced by HIP by looking at the RTT[4].

These tests will be used to give a basic idea of the delays that HIP (Scenario B) adds to the current architecture (Scenario A). Also, these measurements will be the basis on which we will focus to discern, later on, which resolution mechanism performs better (Scenario C and D), in terms of added delays.

Some bash scripts had to be programed to run the different experiments and also to automate the configuration of all the scenarios. It was agreed to perform 100 repetitions of each experiment to reduce the 95% confident intervals.

The variables we used to perform these experiments were the delays configured in each link of our HIP network. Thus, depending on the value of the

[3]Note that we didn't use the loaded entries in the RVS and DHT as a variable in the tests because the processing time when searching, inserting and removing was processor time, insignificant in relation to the global times measured.

[4]Round Trip Time refers to the elapsed time for a message to get to a remote place and go back again. Note that all the delays considered in this thesis will be treated as RTT throughout the experiments.

configured delay between the hosts[5], we divided every test in seven different cases (case0, case1, ... case6). This means that every experiment (HIP Base Exchange, Readdressing, Registration and RTT measurement) in every scenario (A, B, C or D) will be repeated seven times depending on the case tested. Figure 4.3 and Table 4.1 illustrate this.

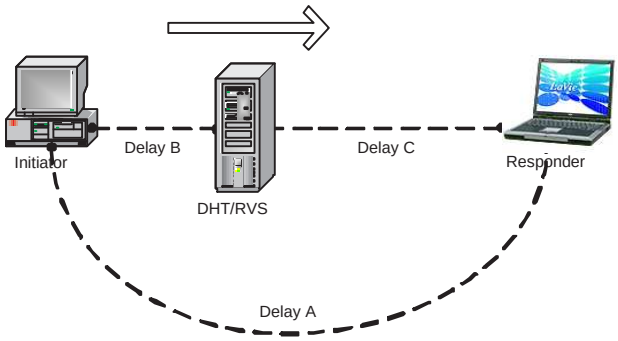


Figure 4.3: Delays configured in the scenarios A to D.

	Case0	**Case1**	**Case2**	**Case3**	**Case4**	**Case5**	**Case6**
Delay A	100 ms	220 ms	220 ms	220 ms	220 ms	220 ms	220 ms
Delay B	100 ms	20 ms	60 ms	100 ms	140 ms	180 ms	220 ms
Delay C	100 ms	220 ms	180 ms	140 ms	100 ms	60 ms	20 ms

Table 4.1: Delays configured in each link in scenarios A to D.

In case0, a static delay of 100 ms RTT was configured between all hosts to simulate a *real* communication through the Internet (cause few milliseconds don't seem to be a good approach in nowadays Internet connections). In this situation, Initiator, Responder and RVS/DHT are always equidistant (Delay A = Delay B = Delay C) to be able to easily compare the measured delays in the different experiments.

The other six cases try to simulate the transition of the RVS/DHT moving from the Initiator to the Responder. That is, we modify the delays such that, in case1, the RVS/DHT is closer to the Initiator than to the Responder (Delay B < Delay C) and then it moves until in case6 RVS/DHT is closer to the Responder than to the Initiator (Delay B > Delay C). Note that, the delay between Initiator and Responder (Delay A) will always be constant (220 ms RTT) through the different cases. These six cases are only significant in Scenario C and Scenario D, where a DHT or a RVS are used (that means that

[5]Delay A between Initiator and Responder, Delay B between Initiator and RVS/DHT and Delay C between RVS/DHT and Responder.

Delay B and Delay C are actually used). The purpose of these last six cases is to know the contributions of the packets exchanged in the different links in mechanisms such as HIP Base Exchange, Readdressing or Registration when DHT and RVS are used. Note that, moreover, the values chosen for the delay in these cases represent a range which covers from a local communication to an international one [28].

Coming back to the tests performed, four experiments are taken into account in our scenarios: HIP Base Exchange, Readdressing, Registration and RTT measurement. These tests are listed below with a short accounting for why they have been selected.

4.3.1 HIP base exchange

To establish a HIP connection the initial HIP Base Exchange (HBE) has to be performed. As described in the HIP Base Exchange section (see SubSection 2.1.1), the HIP base exchange comprises a 4-way handshake. This mandatory handshake introduces a delay every time a HIP connection is established. This test is included to determine how long the delay at the beginning of a HIP connection is.

In order to measure the HBE time, we made some modifications in the code to add some timers. Thus, we measured the values in the HIP layer, counting not only the transition of the packets between Initiator and Responder, but also the processing time at both hosts. Specifically, the measurement comprises from the moment when the Initiator sends the I1 packet until the Initiator again receives the R2 packet.

4.3.2 Readdressing

While a HIP connection is active it may have to rekey due to a change of IP address as seen in SubSection 2.1.3. Other reasons can cause this rekeying such as the ESP sequence numbers, but we will focus only on the mobility case. Readdressing means that a HIP UPDATE packet is sent by the mobile node and has to be, at least, acknowledged by the fixed peer. While this is done, no data packets can be transmitted. This test is included to see how the rekeying affects the performance of a HIP connection.

Like in the HBE timing, here we also took the measurements in the HIP layer. The times given comprise since the Mobile node detects a location change until it receives the last UPDATE packet from the Fixed node. It must be mentioned that the readdressing type performed by the OpenHIP implemention is the one with the rekeying initiated by the Mobile node (see Subsection 2.1.3).

In this experiment, it has to be kept in mind that under HIP, when a node changes its IP address, a readdressing has to be performed with all the HIP nodes with whom the moving node has currently an association, including of course the RVS (also an update of the mapping HIT to IP is included in the readdressing mechanism in this case). Figure 4.4 illustrates this experiment.

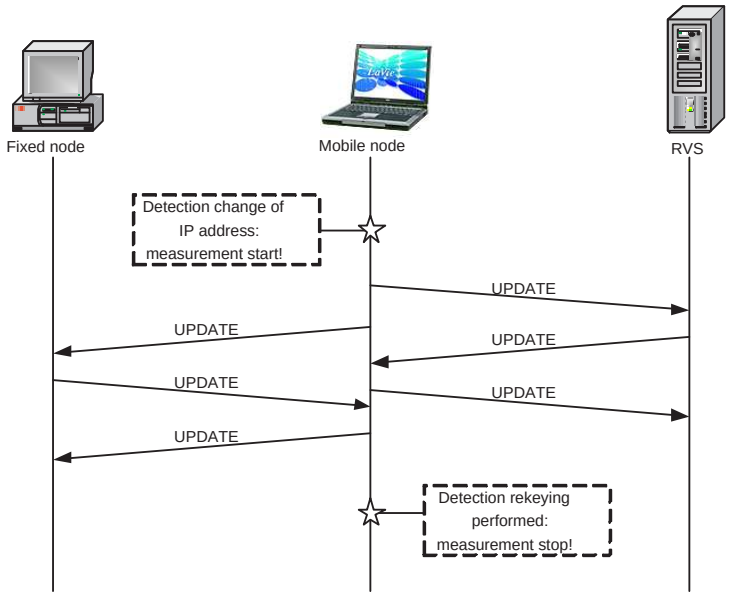


Figure 4.4: Readdressing time measured in Scenarios D.

In case of the DHT, no readdressing, with the DHT itself, is performed (defined as a readdressing mechanism in HIP) but an update of the HIT to IP is done instead, by sending a *put* request as shown in Figure 4.5.

4.3.3 Registration

To register a HIT and an IP address in a Rendezvous Server an initial HIP base exchange with new parameters related to registration is performed. As described in the Registration section (see Sections 3.2 and 3.3), the HIP registration mechanism comprises a 4-way handshake. This mandatory handshake introduces a delay every time a HIP connection is established. Nevertheless, with Distributed Hash Tables (DHTs) a different mechanism is used to register HITs and IP addresses which we would also like to study here. This test is included to determine how long the delay at the beginning of a HIP connection is, due to the RVS or the DHT.

To carry out the measurement of the Registration time with a RVS, we used similar timers than in the HIP Base Exchange. Thus, the timing com-

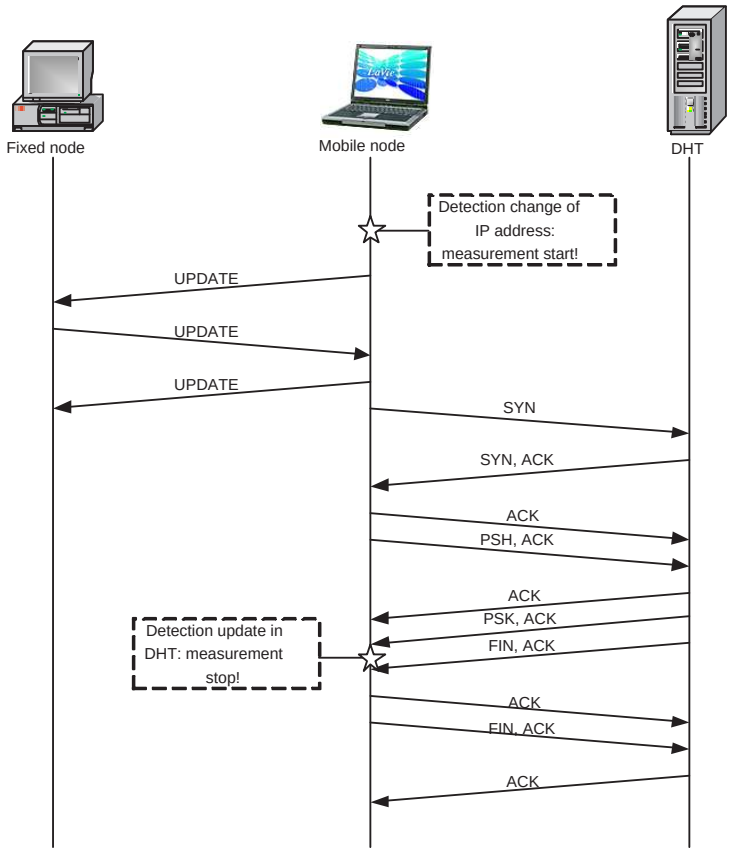


Figure 4.5: Readdressing time measured in Scenarios C.

prises from the moment in which the Initiator (here the Requester Mobile node) sends the I1 packet until the Initiator again receives the R2 packet from the RVS with a *REG_RESPONSE* parameter.

Regarding to the DHT, the Registration consists of a *put* request with the necessary parameters, sent in a TCP[6] connection. Thus, the Registration time includes a three-way handshake which opens the TCP connection but not the termination because since the Mobile node receives a *PSK_ACK* from the DHT, the Initiator can consider that the new mapping HIT to IP mapping is operative. Figure 4.6 illustrates the Registration mechanism with a DHT.

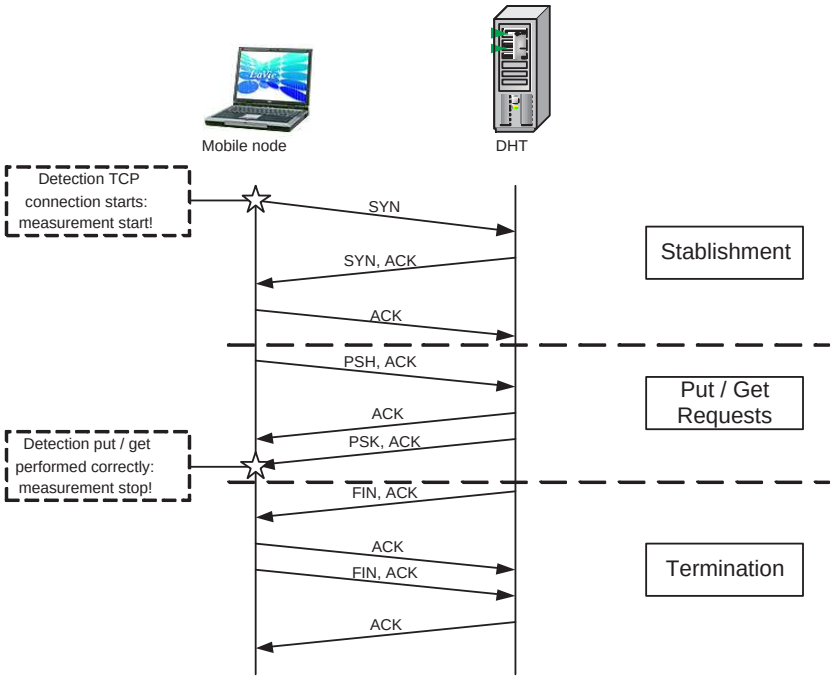


Figure 4.6: Registration with a DHT.

4.3.4 Round Trip Time (RTT)

When two hosts are communicating through a network connection a very important measurement is the Round Trip Time (RTT). The RTT shows how long it takes for a piece of data to travel from one host to the other and back. It is an indicator of the delay introduced by the network connection and the layers below the one being conseidered. It should be noted that the RTT

[6]More detail about Transmission Control Protocol in Appendix D.

can be measured at different layers. Measured at layer two it will encompass only the data link connection whereas a measurement at the application layer encompasses both the network connection and the IP-stack operations at both hosts. This test is included to see which effects HIP has on the RTT of a connection in the current architecture, considering that all data packets are protected with IPSEC ESP in HIP and the RTT must increase (since the data packet size also increases).

To perform this test, a ping is triggered from the Initiator to the Responder once the HIP connection is established.

4.4 Scenarios

As we mentioned before, four are the scenarios tested in our experiments. These scenarios are listed below with a short accounting for why they have been selected, also including the experiments performed in every one. This section describes each scenario without pointing out any result. This will be done in Section 4.5, where we will have all the measurements we need to compare and contrast the performance between all scenarios, cases and experiments.

4.4.1 Scenario A: No HIP

The first scenario will be used as the simple basis where the rest of the experiments have to focus to perform a comparison between them. It will include tests and measurements of the RTT between two hosts without any extra protocol (non HIP-aware nodes).

Thus, for this scenario it was only necessary the use of two hosts (Luxembourg and Vienna) and the click-router (Andorra) to configure Delay A with 100 ms RTT (case0) or 220 ms RTT (case1...case6), as shown in Figure 4.7. To trigger the communication, the *ping* command was used and the desired packets were captured in Ethereal. Hundred repetitions were taken from this experiment and the mean value and standard deviation for the delay RTT were calculated.

4.4.2 Scenario B: HIP with no resolution mechanism

The second scenario will perform tests and measurements of the HIP connection establishment (HIP Base Exchange), re-establishments of HIP connections due to a change of location (Readdressing mechanism) and RTT. These results will be used, aside from gaining base of knowledge about HIP,

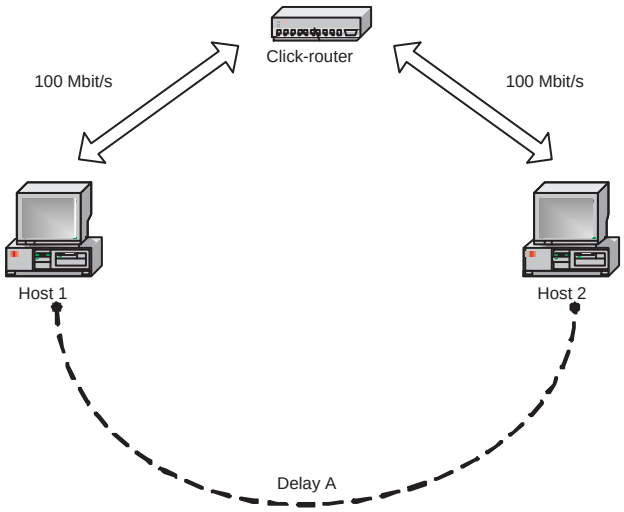


Figure 4.7: Scenario A.

to compare this scenario where no HIP resolution is used to the following ones, where the RVS and the DHT are introduced.

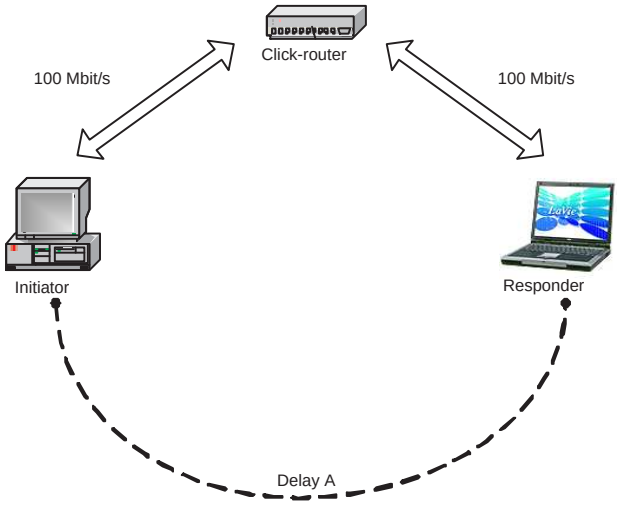


Figure 4.8: Scenario B.

For this scenario it was only necessary the use of two hosts (Luxembourg and Vienna) and the click-router (Andorra) to configure Delay A with 100 ms RTT (case0) or 220 ms RTT (case1...case6), as illustrated in Figure 4.8.

4.4.3 Scenario C: HIP with DHT

The third scenario will perform tests and measurements for the same experiments as in the second scenario but it will also include some measurements for the Registration mechanism performed at the beginning of the communication with a DHT. The results from these test scenarios will, like the results from the other one, be compared to Scenario B and Scenario D to figure out which mechanism performs better under whichever situations.

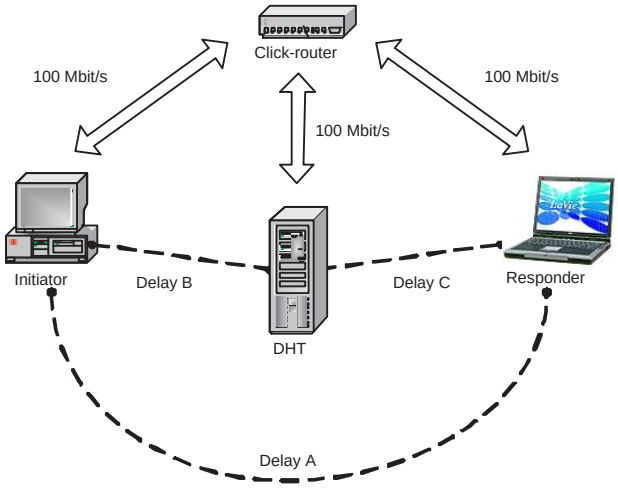


Figure 4.9: Scenario C.

For this scenario, it was necessary the use of three machines: Initiator (Luxembourg), Responder (Vienna) and DHT (Lisbon) as indicated in Figure 4.9. Note that the DHT configured in our network is not a *real* Distributed Hash Table, strictly speaking. The reason is that we only configured one node of the DHT (it could be known as HT in our scenario), so it becomes the simpliest, easiest and fastest way to request a DHT: the first node we are asking for some info is the one which has it. In a *real* DHT, we should have various nodes, store the info in the node which has the key inserted (through a routing algorithm used by the DHT nodes) and, when requesting the info to any node, be routed until the node which has the key [29].

4.4.4 Scenario D: HIP with RVS

The fourth scenario, similar to the one previously described, will perform tests and measurements for the same experiments defined in Scenario B, but it will also include some measurements for the Registration mechanism performed at the beginning of the communication with a RVS (not a DHT). The

results from these test scenarios will, like the results from the previous one, be compared to Scenario B and Scenario C to figure out which mechanism performs better under whichever situations.

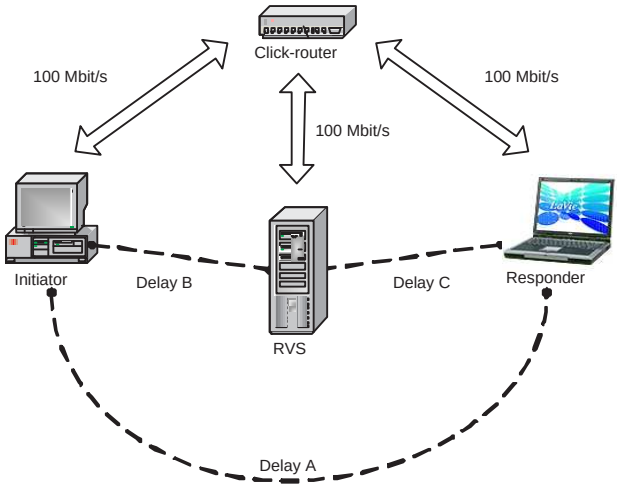


Figure 4.10: Scenario D.

For this scenario, it was necessary the use of three machines: Initiator (Luxembourg), Responder (Vienna) and RVS (Lisbon) as seen in Figure 4.10.

4.5 Tests Results Overview

In this section, we will give an overview of the results obtained during the experiments done in the different scenarios. This section is divided in two parts: Case 0 and rest of cases. This division aims to identify two different situations: the first one is related to the case of having a static delay, equal for each link, while the second one shows the effect of moving the RVS/DHT from the Initiator to the Responder.

Figure 4.11 shows the different experiments performed each scenario. It roughly illustrates the handshakes and packets exchanged during these experiments and will be useful to discern the reasons why we obtained the results we did.

4.5.1 Case 0

The purpose of this case is to study the delays introduced in the several handshakes that have been explained throughout this thesis, using the different alternatives proposed. In order to compare the different mechanisms

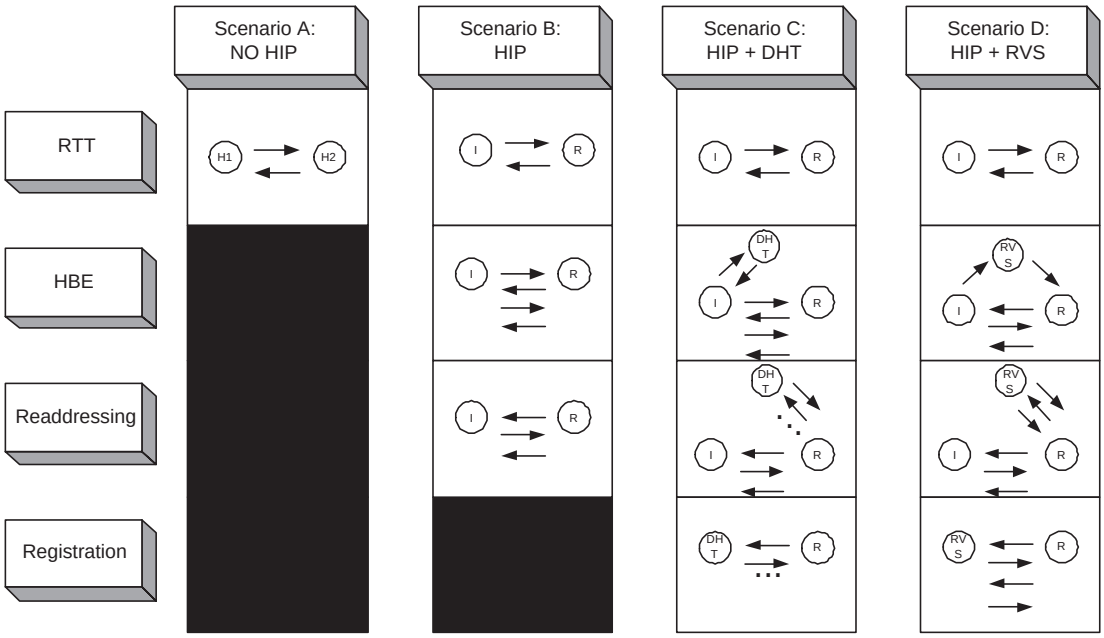


Figure 4.11: Scenarios and experiments overview.

in the same conditions, the network delays that we consider are fixed and are the same in all the experiments.

As we mentioned previously, this case uses a static delay in each link of 100 ms RTT. The results obtained (mean and standard deviation) are shown in Table 4.2. A more visual overview is given in Figure 4.12.

RTT

In the first experiment, measuring the RTT, we can see that there is only a 0.3% increase in the RTT when using HIP. This increase is due to the IPSEC ESP encapsulation, which adds some overhead to the packets transmited. Looking at the result obtained, we conclude that the overhead introduced by the data packets exchanged between two HIP nodes is not important, given the capacity of the current Internet transmission lines. Of course a bigger increase would be observed in case of considering a slower access like a modem line, but the current trends seem to indicate that the broadband access will become a reality by when HIP should start to be deployed, therefore we consider these results as representative enough.

Notice, moreover, that no differences will exist between a ping triggered in a plain HIP scenario (Scenario B) and HIP with any rendezvous/resolution

	Scenario A NO HIP	Scenario B HIP	Scenario C HIP + DHT	Scenario D HIP + RVS
RTT	100,34 ms ± 0,08 ms	100,65 ms ± 0,07 ms	100,65 ms ± 0,07 ms	100,65 ms ± 0,07 ms
HBE		362,58 ms ± 10,19 ms	595,23 ms ± 13,87 ms	413,43 ms ± 7,55 ms
Readdr.		1283,57 ms ± 38,80 ms	1500,14 ms ± 50,48 ms	2359,81 ms ± 39,54 ms
Registr.			211,97 ms ± 3,25 ms	362,29 ms ± 15,89 ms

Table 4.2: Results from experiments tested in case 0.

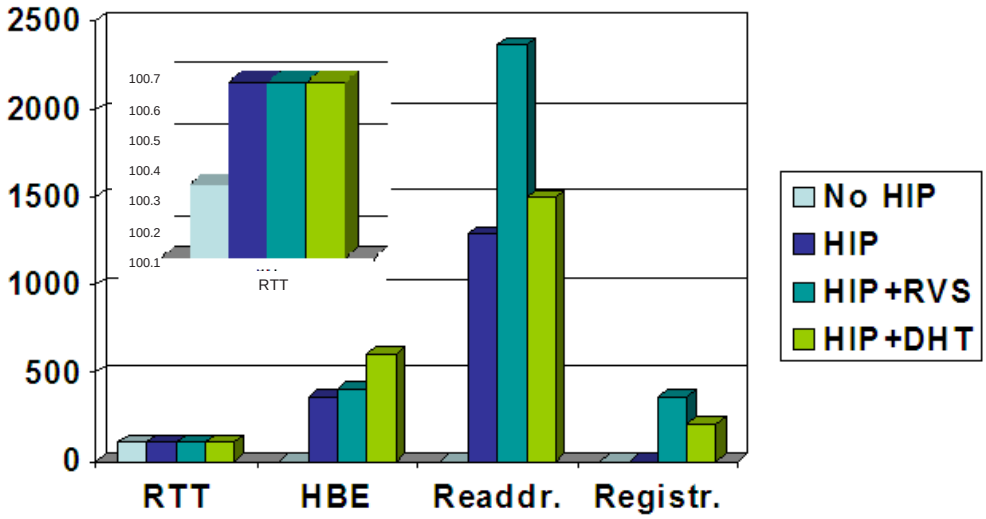


Figure 4.12: Graphic comparison between the results in Table 4.2.

mechanism (Scenarios C and D). Therefore we ran this experiment only once for all the HIP scenarios.

HBE

Regarding the HIP Base Exchange, in Scenario B we can see the delay added to the current architecture by this initial handshake. A big part of the mean value obtained is not due to the RTT but because of the processing time spent in solving the puzzle and creating a state in the Responder.

The values obtained in Scenario C and D seem correct with respect to the ones obtained in Scenario B, since the scenarios with resolution mechanisms include more packets exchanges. Furthermore, comparing both Scenarios C and D, a HBE with a DHT is longer than one with a RVS (with the DHT the HBE is increased by a 64.2% while in the case of RVS only by a 14.0%). The reason is that with the RVS, the Initiator does not perform a lookup for the Responder's IP address, but sends directly the I1 to the RVS which will relay it to the Responder (see Figure 4.11). With the DHT, instead, a TCP connection is opened between the Initiator and the DHT to send a *get* request in order to know the Responder's IP address, and once it is known, a normal HBE is performed between the Initiator and the Responder, without further help from the DHT.

Readdressing

In the Readdressing tests, we observe that with the DHT, the time measured increases by a 16.8%, while with the RVS it increases by 83.8%. This is because with the RVS, the mobility case also includes a Readdressing with the RVS itself as explained in SubSection 4.3.2. This is the reason why the Readdressing time in Scenario D (HIP + RVS) is nearly twice the one in Scenario B (plain HIP), two nodes are updated (Fixed node and RVS) instead of one. Moreover, with the RVS an update in the registration table is also needed (performed also in the Readdressing handshake).

Regarding Scenario C, the Readdressing is performed between the Mobile node and its HIP peer, and a *put* request is performed between the Mobile node and the DHT in order to update the IP address stored there, which takes shorter than a HIP Readdressing.

Being this delay critical for highly mobile HIP nodes, an specific study was done to know the different individual contributions on the overall delay time, which are illustrated in Figure 4.15. The different processes involved in the Readdressing mechanism are represented in Figure 4.13 and Figure 4.14. Thus, our global measurement was further divided in:

- Readdress. Here, the readdress time is considered from the moment when the Mobile node detects a mofication in its location (its IP address has changed) until the moment before it sends the first UPDATE packet to the Fixed node. During this time, changes in the IPSEC Security Associations (SAs) are done to enable the new address under the ESP encapsulation.

- RTT. This measurement includes the time since the first UPDATE packet is sent by the Mobile node until this is acknowledged with another UPDATE.

- UPDATE process. This measurement comprises the time of processing the acknowledged UPDATE packet from the Fixed node.

- Rekeying. Here, a recomputation of the session keys are performed so that a new Security Parameter Index (SPI) is included in the IPSEC SAs already created.

- Update DHT. This time is only measured in Scenario C, where a DHT is used. It comprises from the first packet sent to open the TCP session between the Mobile node and the DHT until the answer of the DHT acknowledging it, before the end on the TCP session.

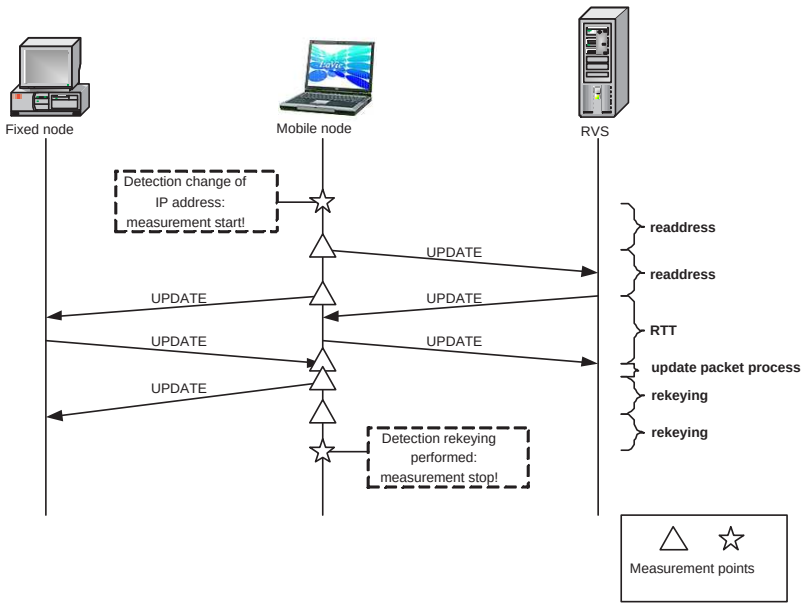


Figure 4.13: Processes involved in the Readdressing time measured in Scenario D.

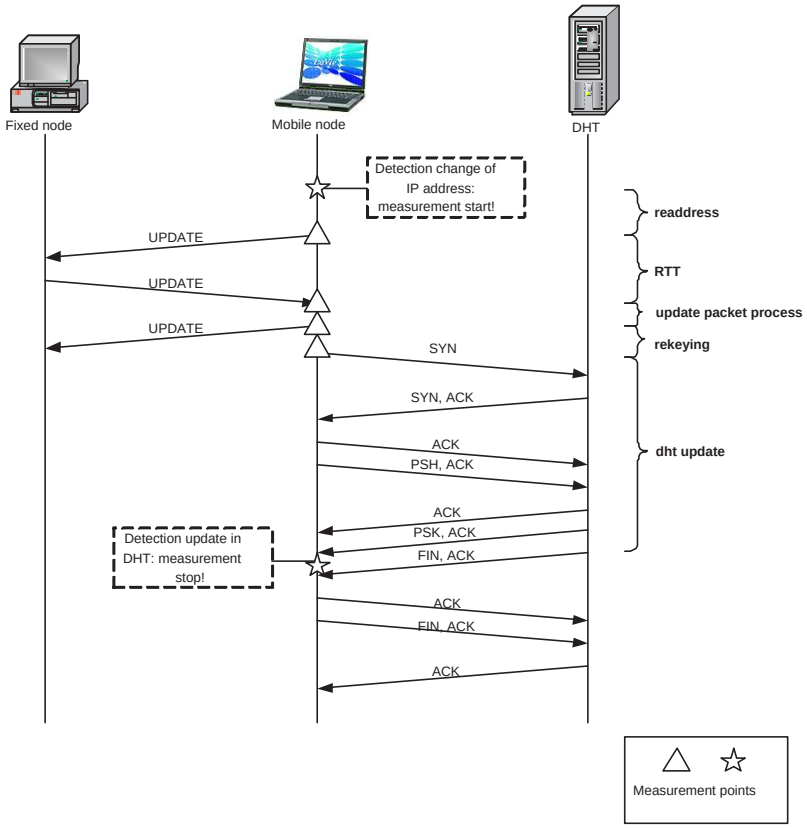


Figure 4.14: Processes involved in the Readdressing time measured in Scenario C.

After the measurements, we realised that the rekeying time was about one second. This surprised us because this mechanism is supposed to be fast enough to support mobile nodes without noticing any changes, apart from the possible change to the actually experienced quality of service. Focusing on this process, we discovered a preconfigured timeout of one second which could be the one added to the HIP ESP specification [10] to prevent being stuck in the rekeying state in case there is an error[7]. After revising all the packets exchanged and the parameters included, we concluded that no error was happening. Nevertheless, our short time left did not let us find out the actual reason for such timeout.

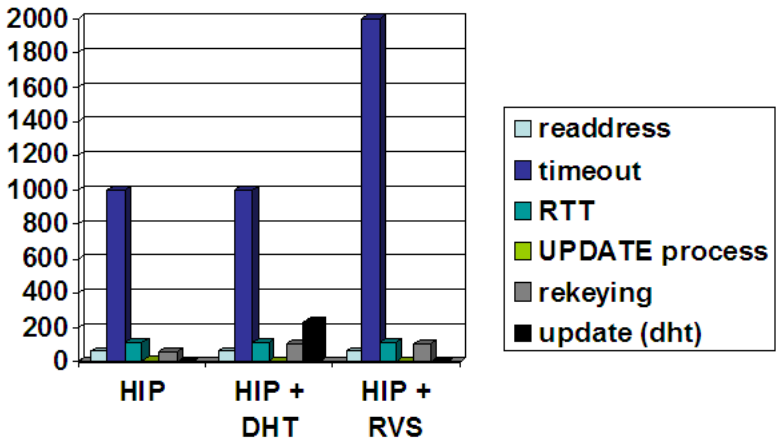
At the time of writing this thesis, we consider this timeout as a mechanism belonging to the IPSEC implementation, not to the HIP one, even though we are not completely sure about that. Therefore we decide to isolate the contribution of this timeout, as it is shown in the following graphics:

On the upper graphic of Figure 4.15, the processes involved in the Readdressing include the timeout of the rekeying. Thus, we observe, as previously explained, that Scenario C (DHT) performs much better than Scenario D (RVS) in terms of delays introduced (a 33.8% delay less with the DHT). However, as illustrated in Figure 4.15, in the lower graph, if the timeout was corrected, the RVS scenario would present a much better performance (46.0% less delay) than the one with the DHT. Is for this reason that it is important to find out what causes this increased rekeying (it might also be a bug in the current OpenHIP implementation).

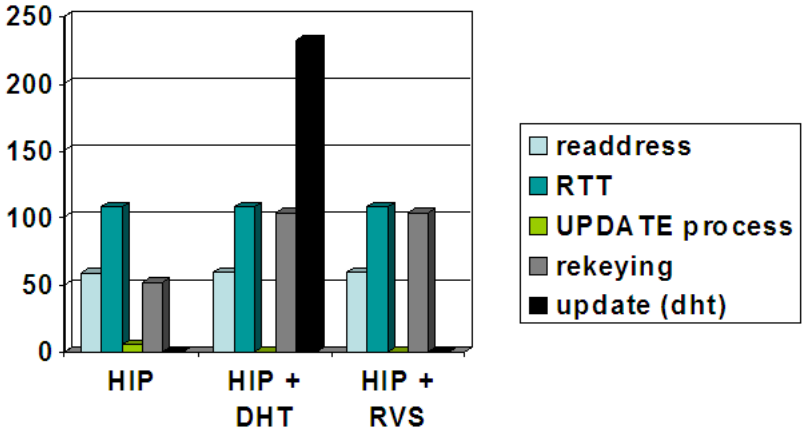
Registration

Regarding the Registration experiments, we observe that, as it was expected, in Scenario D the mean value is close to the one of the HBE performed in Scenario B (plain HIP). This is because the Registration mechanism is a normal HBE (but with new parameters included) performed between a Registrar ofering a Rendezvous Service and a Requester asking for it. Instead, in Scenario C, the mean value obtained is quite smaller (41.5% of the RVS Registration time), since with the DHT a complete different mechanism is performed. Notice that the time measured in this Scenario does not include the TCP session termination since once the Mobile node receives the

[7]In case a protocol error occurs and the peer system acknowledges the UPDATE but does not itself send an *ESP_INFO*, the system may not finalize the outstanding ESP SA update request. To guard against this, a system may re-initiate the ESP SA update procedure after some time waiting for the peer to respond, or it may decide to abort the ESP SA after waiting for an implementation-dependent time. The system must not keep an oustanding ESP SA update request for an indefinite time.



(a) Timeout configured



(b) No timeout configured

Figure 4.15: Processes involved in the Readdressing with/out the timeout configured.

acknowlegment of the *put* request, the DHT is storing the new HIT→IP sent.

Thus, the main reason why the registration with the RVS is bigger than the one with the DHT is that, as mentioned before, the first one is a complete HBE, where the Initiator (here the Requester asking for Rendezvous Service) has to solve the puzzle mentioned in Chapter 2. These extra operations mean an increase in the delay with respect to the DHT, but of course the benefit is a gain in security. Remember that the lack of security is one of the main drawbacks of using DHTs as a resolution mechanism, and that we are also considering the most optimistic case for the DHT.

4.5.2 Cases 1 to 6

The purpose of these cases is to study how the changes on the network topology affect the resuls that we obtained in case 0. Only scenarios C and D (DHT and RVS) are considered.

In this set of cases, we try to simulate the movement of the RVS and DHT from the Initiator to the Responder. The reason for this is that we consider a situation where the resolver (DHT or RVS) might be placed anywhere between the Initiator and the Responder with a uniform probability. Thus, in case 1 we suppose a delay between the Initiator and the DHT/RVS much smaller than the one between the DHT/RVS and the Responder. As long as the case number increases, the DHT/RVS approaches the Responder's vecinity by decreasing the delay between them and increasing the one between the Initiator and the DHT/RVS.

Note that the delay between the Initiator and the Responder is in all cases constant (220 ms RTT), so Scenario A and B (the ones where no RVS or DHT is included) will not see any difference in the measurements through the different cases. The values obtained will only be helpful to compare to the other Scenarios, with no mention about the distances between RVS/DHT and the rest of hosts.

All the results (mean and standard deviation) are included in Table 4.3, but some other graphics give a more visual overview of those values (Figures 4.16, 4.17 and 4.18).

RTT

This value, as expected, is the same for all the HIP scenarios, since in this experiment there are only two hosts involved and the data is routed directly from the Initiator to the Responder (there is no relay of the I1 packet). This is the reason why we run the experiment only once for all the cases. Thus, there is no difference between the value obtained in Scenarios C and

		Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Scenario A NO HIP	**RTT**	220,36 ms ± 0,08 ms					
Scenario B,C,D	**RTT**	220,82 ms ± 0,09 ms					
Scenario B HIP	**HBE**	602,50 ms ± 6,38 ms					
	Readdr.	1410,97 ms ± 30,10 ms					
Scenario C HIP + DHT	**HBE**	736,94 ms ± 8,79 ms	771,42 ms ± 7,49 ms	847,94 ms ± 6,79 ms	932,91 ms ± 6,66 ms	973,06 ms ± 6,70 ms	1053,7 ms ± 7,28 ms
	Readdr.	2015,78 ms ± 100,29 ms	1904,68 ms ± 83,25 ms	1784,77 ms ± 69,85 ms	1681,77 ms ± 81,47 ms	1560,38 ms ± 44,14 ms	1464,89 ms ± 78,85 ms
	Registr.	450,80 ms ± 5,68 ms	370,95 ms ± 6,03 ms	290,96 ms ± 5,98 ms	210,11 ms ± 2,98 ms	130,14 ms ± 3,05 ms	55,64 ± 3,96 ms
Scenario D HIP + RVS	**HBE**	614,94 ms ± 18,40 ms	614,00 ms ± 7,29 ms	612,72 ms ± 5,99 ms	613,93 ms ± 5,99 ms	614,67 ms ± 7,11 ms	612,79 ms ± 6,20 ms
	Readdr.	2499,95 ms ± 47,30 ms	2500,16 ms ± 47,29 ms	2497,38 ms ± 46,50 ms	2498,41 ms ± 47,15 ms	2498,80 ms ± 46,45 ms	2497,84 ms ± 46,76 ms
	Registr.	601,49 ms ± 5,46 ms	522,94 ms ± 7,06 ms	442,44 ms ± 6,55 ms	362,97 ms ± 5,86 ms	282,18 ms ± 6,88 ms	203,12 ms ± 8,15 ms

Table 4.3: Results from experiments tested in cases 1 to 6.

D througout the different cases since the RTT is fixed like the delay between them.

HBE

Regarding Scenario C, where a DHT is used, we observe how the HBE time increases as the DHT moves from the Initiator to the Responder. The reason for this is that the Initiator has to request for a lookup in the DHT, therefore the further it is, the longer it takes. The mean value for the measurements seem to be fair since the whole HBE includes roughly the time of the plain HBE plus a lookup in the DHT.

Concerning Scenario D, with a RVS as a resolver, as it is logical, the mean value for the HBE time is bigger than in Scenario B (plain HIP) because it includes one more packet exchanged (the I1 relayed from the RVS to the Responder). Note that this mean value is constant not depending on the distance between the Initiator and Responder to the RVS because in our experiments the total delay time between them is assumed to be constant (Delay B + Delay C = constant, Delay A = constant).

We can also get to these conclusions by simply watching Figure 4.16. In this hypothesis, we can see that HIP + RVS performs better than HIP + DHT since this last Scenario has a high dependency on the distance of the resolver, which does not affect Scenario D with a RVS.

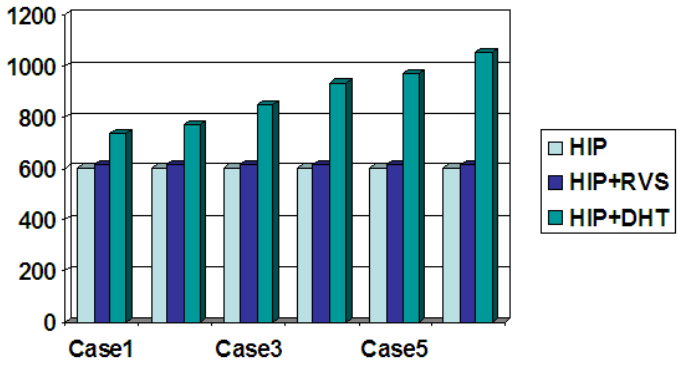


Figure 4.16: Graphic comparison of HBE time obtained through the different cases in each scenario.

Readdressing

In Scenario C, we can see how the Readdressing time decreases when the DHT is closer to the Mobile node (Responder) which is updating its HIT to IP address mapping. This is due to the fact that this Readdressing includes not only the rekeying with the HIP peer node but the update with the DHT. Note that even the delay is getting bigger when the DHT is far from the Responder, it is still always below the RVS case.

In Scenario D, it is observed how the Readdressing time is again constant regardless of the distance of the RVS. This can be explained as this time includes two different rekeyings (Initiator and RVS), which are performed concurrently so then the global time will always be the one of the further node.

In Figure 4.17, these variations are clearly illustrated. From the added delay point of view, in these experiments HIP + DHT performs better than HIP + RVS. However, note that even this tries to be a *real* simulation of the Internet architecture, it is not since having only one DHT node means testing always the best case (which it is not realistic). Resource limitations refrain us from having a whole DHT network.

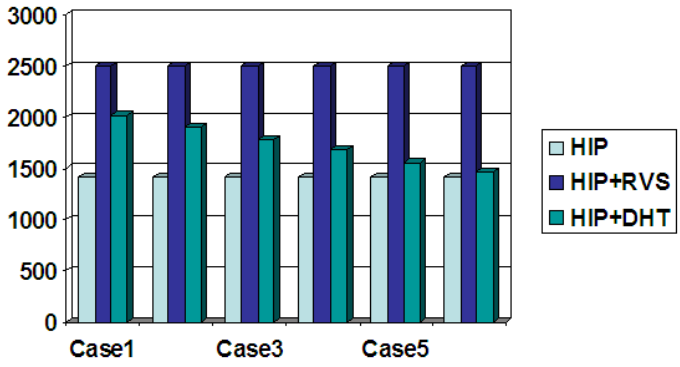


Figure 4.17: Graphic comparison of Readdressing time obtained through the different cases in each scenario.

Registration

Regarding the Registration, in Scenarios C and D the time for the Registration is shorter as the DHT/RVS is closer to the Mobile node (the one which actually registers its HIT and IP address). However, we can observe how the time measured is always bigger in Scenario D (HIP + RVS) due to the initial

HBE with specific parameters performed to register, so then from the delays added point of view the performance of the DHT would be better than with RVS. Again, we have to emphasize the fact that only one node of DHT is configured in our experiments (so then we are counting only the best case). Furthermore, as in the other experiments we have to choose between a fast performance or a not as fast but secure one.

In Figure 4.18 we can clearly see the variations explained above.

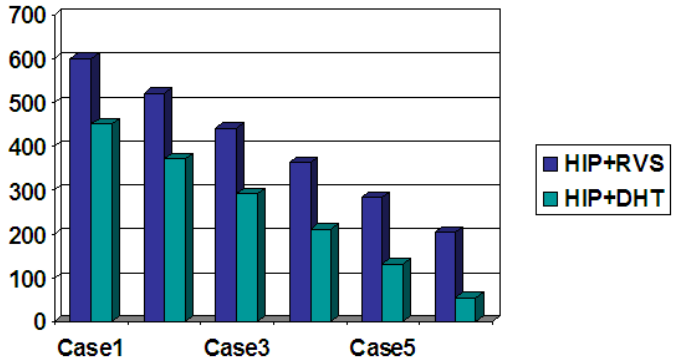


Figure 4.18: Graphic comparison of Registration time obtained through the different cases in each scenario.

4.6 Conclusions

This Section is included to summarize the conclusions arisen during the tests results overview.

Case 0

- The data overhead introduced by HIP is not significant as observed in the ping test (RTT).
- The HBE is time consuming because in our experiments was between 3.5 and 6 times a RTT depending on the resolution mechanism used. Note that it is an exchange only needed at the beginning of the communication between twho HIP nodes, though.
- Comparing DHTs and RVSs, our experiments show a better performance for the DHT in both Registration and Readdressing measurements. From the RVS point of view, having a bigger time for Reg-

istration is not critical since, as HBE, this mechanism is only needed at the beginning of the communication between a RVS and a Mobile node. Regarding Readdressing, a big time here will be critical for hihgly moving nodes.

Nevertheless, it has to be noticed that about DHT, on one hand, we are showing the most optimistic result for the DHT case (as only one DHT node is considered) and, on the other hand, the DHT, unlike RVS, is a resolution mechanism which does not provide security.

Cases 1 to 6

- Assuming the resolver node placed between the Initiator and the Responder, the RVS shows a constant performance that does not depend on its specific position, as seen in the HBE and Readdressing experiments. Instead, the DHT performance depends on its position with respect to the node that has to access the DHT.

Another important issure related to our performance evaluation is the fact that we discovered a timeout of about one second that substantially increases the Readdressing time in all HIP scenarios (in HIP+RVS two seconds as two HIP nodes are rekeyed). It has to be noticed that if this timeout could be somehow solved, the results above mentioned would change inmediately since HIP + RVS would work much better in this critical experiment and hihgly mobile nodes could be even better supported than in the HIP + DHT case.

Chapter 5

Summary and Conclusions

New needs arising in the current Internet architecture have lead to some issues still pending to be solved. Unsecured, fixed and single-homed nodes now search for a global solution. Several proposals have been defined in the last decade but they do not cover all the problems mentioned above. Thus, the Host Identity Protocol, created focusing in security, mobility and multi-homing, seems to be a good approach for the Next Generation Internet.

In this thesis we have started from the three main principles that define HIP and followed with the problems that such a new architecture has to deal with, specially when being introduced in the current main name resolution infrastructure: Domain Name System (DNS). After some discussion, it was agreed that a new rendezvous mechanism is needed to map identities and locators and connect HIP nodes since DNS is not designed to support high mobile nodes due to its caching architecture. Among several proposals, Distributed Hash Tables (DHTs) and Rendezvous Servers (RVSs) seem to be the most suitable ones.

After studying the main characteristics of DHTs, some problems were found such as the lack of security when accessing them (*put* requests can be performed by any node just knowing the key stored, the HIT) and the impossibility to support double-jump moving nodes (two nodes changing their location at the same time) with the important drawback of having packet losses. At this point, RVSs, HIP nodes with special functionalities, solve these two problems. Being the RVS also a HIP node, all the communications with other HIP nodes will be secure, and regarding double-jumps, it will act as a packet forwarder to solve the previously mentioned problem.

In addition to the theoretical part, some studies of the performance of HIP, and HIP with DHT or RVS as resolution mechanisms, have been carried out. These tests include measuring the delays added by this new architecture to the current one. Thus, measurements on the initial HIP Base Exchange,

the Readdressing in case of mobile nodes and the Registration, in case a RVS or a DHT are needed, have been performed. The extra overhead introduced in data packets exchanged has been also measured. The results from these experiments conclude in the following:

- The data overhead introduced by HIP is not significant as observed in the RTT measured.
- The HBE increases between 4 and 6 times the initial connection time depending on the resolution mechanism used (4 for the RVS and and 6 for the DHT). Note, however, that it is an exchange only needed at the beginning of the communication between two HIP nodes.
- Comparing DHTs and RVSs, our experiments show a better performance for the DHT in both Registration and Readdressing measurements. Having a bigger time for the Registration is not critical since, as the HBE, this mechanism is only needed at the beginning of the communication between a RVS/DHT and a Mobile node. Nevertheless, regarding the Readdressing, a big time here will be critical for hihgly moving nodes. However, the following has to be noticed within the DHT experiments: i)we are showing the most optimistic result for the DHT case (as only one DHT node is considered) and ii)the DHT, unlike RVS, is a resolution mechanism which does not provide security, nor support for the double-jump problem.
- Assuming the resolver node placed between the Initiator and the Responder, the RVS shows a constant performance that does not depend on its specific position, as seen in the HBE and Readdressing experiments. Instead, the DHT performance depends on its position with respect to the node that has to access the DHT.

Note that if the Readdressing timeout problem found in Chapter 4 could be solved somehow, the HIP + RVS case would perform much better than the most optimistic DHT case (only one node is considered), also in the critical case of mobility, making then the RVS an even better solution.

5.1 Future Work

Concerning the NAT traversal issue mentioned in Chapter 1, some extensions are currently being done in the OpenHIP code in order to enable the Initiator start a HIP communication when it is behind a NAT [13]. Though, the

responder behind a NAT case (how a node initiates a HIP Base Exchange with another node behind a NAT) has not been faced yet. Focusing in this last case, basically, there will be an extension of the RVS to solve it. The *easiest* way to enable NAT traversal in every possible case (the different types of NATs, all mobility cases, moving from behind a NAT to the public network, behind another NAT, etc...) would be to make all the HIP and ESP packets go through a RVS. Thus, the hosts should register on the RVS and keep some connections opened between them and the RVS (some keepalive packets would be required to keep the "hole" in the NAT opened). Then, when they want to communicate with other hosts, all their requests and user data exchanges will be relayed by the RVS (not only the I1 packet). This solution would work in any case although it is not the most efficient one since the RVS becomes a clear bottle neck. A lot of activity is being carried out by the IETF regarding this issue.

This thesis has introduced a new element in the HIP architecture, the RVS. This node has been used as a resolver, able to support mobility, to discover the mapping HIT to IP when a HIP node wants to initiate a communication with another node. But the uses and capabilities of the RVS are not by any means limited to only act as a resolver. Through the registration handshake, also introduced in this thesis, the RVS will be able to advertise new services and then the HIP nodes to subscribe to them. The role that the RVS will play in the NAT traversal problem is only an example, but for sure the more HIP is studied and deployed, the more functions will be added to the Rendezvous Server.

List of Figures

List of Tables

Bibliography

[1] EGGERT, L., LAGANIER, J., LIEBSCH, M. and STIEMERLING, M., "HIP Resolution and Rendezvous Mechanisms", in *Workshop on HIP and Related Architectures*, Washington DC, USA. 2004.

[2] NIKANDER, P., YLITALO, J. and WALL, J. "Integrating Security, Mobility and Multi-homing in a HIP way", Ericsson Research NomadicLab.

[3] AURA, T., NAGARAJAN, A. and GURTOV, A. "Analysis of the HIP Base Exchange Protocol", 2005.

[4] LAGANIER, J., KOPONEN, T., EGGERT, L. "Host Identity Protocol (HIP) Registration Extension", Work in Progress (draft-ietf-hip-registration-01), December 2005.

[5] LAGANIER, J., EGGERT, L. "Host Identity Protocol (HIP) Rendezvous Extension", Work in Progress (draft-ietf-hip-rvs-04), October 2005.

[6] MOSKOWITZ, R., NIKANDER, P., JOKELA, P. "Host Identity Protocol (HIP)", Work in Progress (draft-ietf-hip-base-04), October 2005.

[7] MOSKOWITZ, R., NIKANDER, P. "Host Identity Protocol Architecture", Work in Progress (draft-ietf-hip-arch-03), August 2005.

[8] HENDERSON, T. "End-Host Mobility and Multihoming with the Host Identity Protocol", Work in Progress (draft-ietf-hip-mm-02), July 2005.

[9] NIKANDER, P. and LAGANIER, J. "Host Identity Protocol (HIP) Domain Name System (DNS) Extensions", Work in Progress (draft-ietf-hip-dns-04), December 2005.

[10] JOKELA, P., MOSKOWITZ, R. and NIKANDER, P. "Using ESP transport format with HIP", Work in Progress (draft-ietf-hip-esp-01), October 2005.

[11] JOKELA, P., MOSKOWITZ, R. and NIKANDER, P. "Using HIP with Legacy Applications", Work in Progress (draft-henderson-hip-applications-01), July 2005.

[12] KOMU, M. "Native Application Programming Interfaces for the Host Identity Protocol", Work in Progress (draft-mkomu-hip-native-api-00), March 2005.

[13] SCHMITT, V., PATHAK, A., KOMU, M., EGGERT, L. and STIEMERLING, M. "HIP Extensions for the Traversal of Network Address Translators", Work in Progress (draft-schmitt-hip-nat-traversal-00), February 2006.

[14] "The OpenHIP project", www.openhip.org

[15] "OpenDHT: A Publicly Accessible DHT Service", www.opendht.org

[16] "The Click Modular Router Project", http://pdos.csail.mit.edu/click/

[17] "Ethereal: A Network Protocol Analyzer", www.ethereal.com

[18] "The Mathworks", www.mathworks.com

[19] MONTAVONT, N., NOEL, T., KASSI-LAHLOU, M. "MIPv6 for Multiple Interfaces" Work in Progress (draftmontavont-mobileip-mmi-00), July 2002.

[20] O'SHEA, G., ROE, M. "Child-proof Authentication for MIPv6 (CAM)", ACM Computer Communications Review, Volume 31, Number 2, April 2001.

[21] "Mobile IPv6 for Linux", www.mobile-ipv6.org

[22] "Stream Control Transmission Protocol (SCTP)", www.sctp.org

[23] STEWART, R. et al. "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration" , Work in Progress (draft-ietf-tsvwg-addip-sctp-14), September 2006.

[24] "Site Multihoming in IPv6 (multi6)", http://www.ietf.org/html.charters/multi6-charter.html

[25] "Mobile Nodes and Multiple Interfaces in IPv6 (monami6)", http://www.ietf.org/html.charters/monami6-charter.html

[26] "DNSSEC: DNS Security Extensions. Securing the Domain Name System", www.dnssec.net.

[27] "OpenHIP contributing"
http://www.openhip.org/wiki/index.php?title=Contributing

[28] CORLETT, A., PULLIN., D and SARGOOD, S. "Statistics of One-Way Internet Packet Delays" in *53rd IETF*, Minneapolis, March 2002.

[29] RHEA, S., CHUN, B., KUBIATOWICZ, J. and SHENKER, S., "Fixing the Embarrassing Slowness of OpenDHT on PlanetLab", University of California, Berkeley.

[30] KNUTH, D. "The Art of Computer Programming. Sorting and Searching", Addison Wesley.

[31] HUITEMA, C. "IPv6. The New Internet Protocol", Prentice Hall.

[32] KUROSE, J. and ROSS, K. "Computer Networking. A TOP-DOWN Approach featuring the Internet", International Edition.

Appendix A

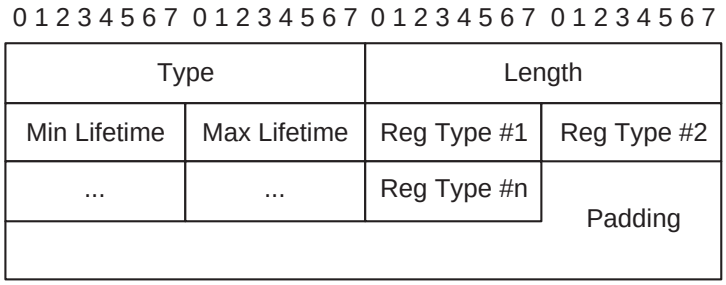
New HIP Parameters

A.1 Registration parameters

In this section, there are described the different HIP parameters created to be able to perform a registration process.

Note that the HIP registration uses an exponential encoding of registration lifetimes. This allows compact encoding of 255 different lifetime values ranging from 4 ms to 178 days into an 8-bit integer field. The lifetime exponent field used throughout this document must be interpreted as representing the lifetime value $2^{((lifetime-64)/8)}$ seconds.

REG_INFO



Type: 930

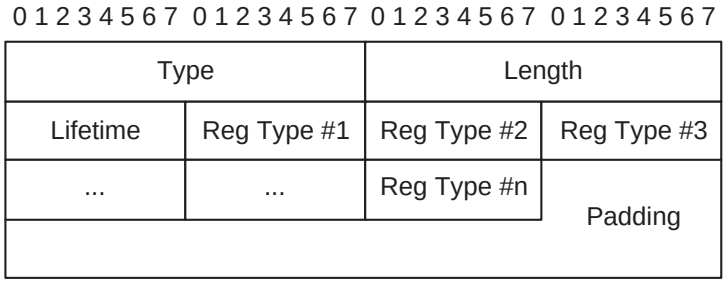
Length: Length in octets, excluding Type, Length, and Padding.

Min Lifetime: Minimum registration lifetime.

Max Lifetime: Maximum registration lifetime (e.g. Rendezvous type 1).

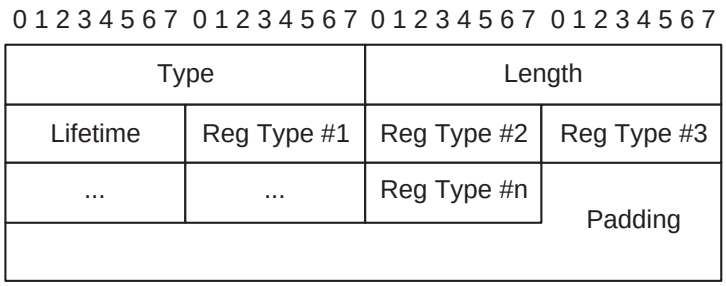
Registrars include the parameter in R1 packets in order to announce their registration capabilities.

REG_REQUEST



Type: 932
Length: Length in octets, excluding Type, Length, and Padding.
Lifetime: Requested registration lifetime.
Reg Type: The preferred registration types in order of preference.
A requester includes the REG_REQUEST parameter in I2 or UPDATE packets to register with a registrar's service(s).

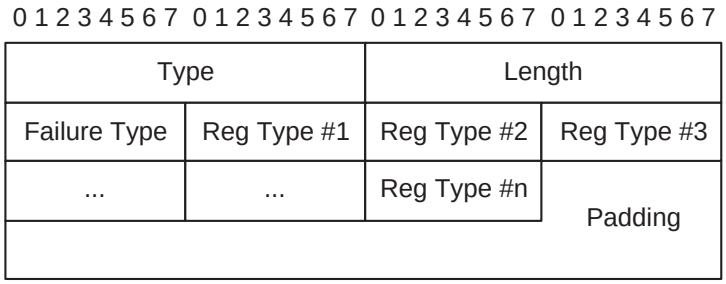
REG_RESPONSE



Type: 934
Length: Length in octets, excluding Type, Length, and Padding.
Lifetime: Granted registration lifetime.
Reg Type: The granted registration types in order of preference.
The registrar includes a REG_RESPONSE parameter in its R2 or UPDATE packet only if a registration has successfully completed.

REG_FAILED

Type: 936
Length: Length in octets, excluding Type, Length, and Padding.
Failure Type: Reason for failure.

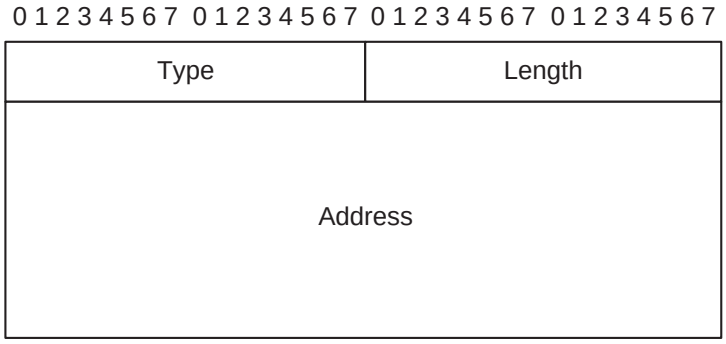


Reg Type: The registration types that failed with the specified reason (e.g. 0 for "'Registration requires additional credentials"' and 1 for "'Registration type unavailable"'.

The registrar should include the REG_FAILED parameter in its R2 or UPDATE packet, if registration with the registration types listed has not completed successfully and a requester is asked to try again with additional credentials for instance.

A.2 Rendezvous mechanism parameters

FROM Parameter



Type: $65498 = 2^{16} - 2^5 - 2$

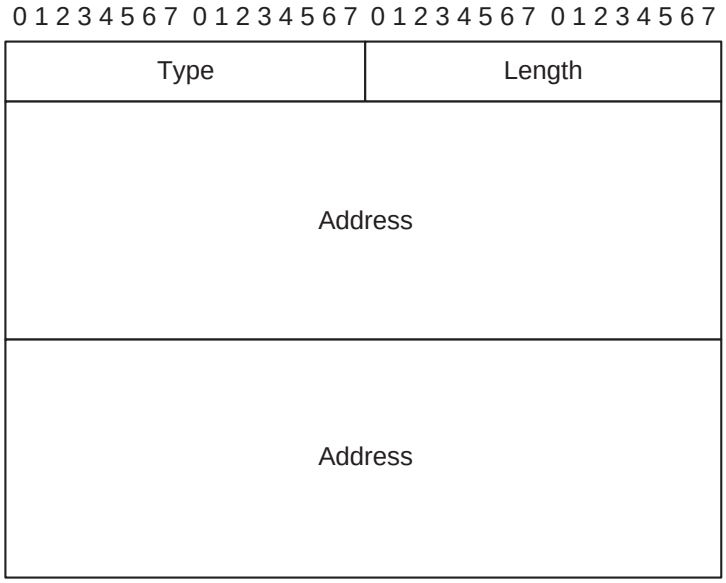
Length: 16

Address: An IPv6 address or an IPv4-in-IPv6 format IPv4 address.

A rendezvous server MUST add a FROM parameter containing the original source IP address of a HIP packet whenever the source IP address in the IP header is rewritten.

VIA_RVS Parameter

Type: $65502 = 2^{16} - 2^5 + 2$



Length: Variable.

Address: An IPv6 address or an IPv4-in-IPv6 format IPv4 address.

After the Responder receives a relayed I1 packet, it can begin to send HIP packets addressed to the Initiator's IP address, without further assistance from an RVS. For debugging purposes, it may include a subset of the IP addresses of its RVSs in some of these packets. When a Responder does so, it must append a newly created VIA_RVS parameter at the end of the HIP packet. The main goal of using the VIA_RVS parameter is to allow operators to diagnose possible issues encountered while establishing a HIP association via an RVS.

Appendix B

HIP State

The HIP protocol itself has little state. In the HIP base exchange, there is an Initiator and a Responder. Once the Security Associations (SAs) are established, this distinction is lost. If the HIP state needs to be re-established, the controlling parameters are which peer still has state and which has a datagram to send to its peer. The following state machine attempts to capture these processes.

The state machine is presented in a single system view, representing either an Initiator or a Responder. There is not a complete overlap of processing logic here and in the packet definitions. Both are needed to completely implement HIP.

Section B.1 shows the different HIP states while Section B.3 describes the packet processing rules. This state machine focuses on the HIP I1, R1, I2, and R2 packets only. Other states may be introduced by mechanisms in other specifications (such as mobility and multihoming).

B.1 HIP states

State	Explanation
UNASSOCIATED	State machine start
I1-SENT	Initiating base exchange
I2-SENT	Waiting to complete base exchange
R2-SENT	Waiting to complete base exchange
ESTABLISHED	HIP association established
CLOSING	HIP association closing, no data can be sent
CLOSED	HIP association closed, no data can be sent
E-FAILED	HIP exchange failed

B.2 Simplified HIP state diagram

The following diagram shows the major state transitions. Transitions based on received packets implicitly assume that the packets are successfully authenticated or processed.

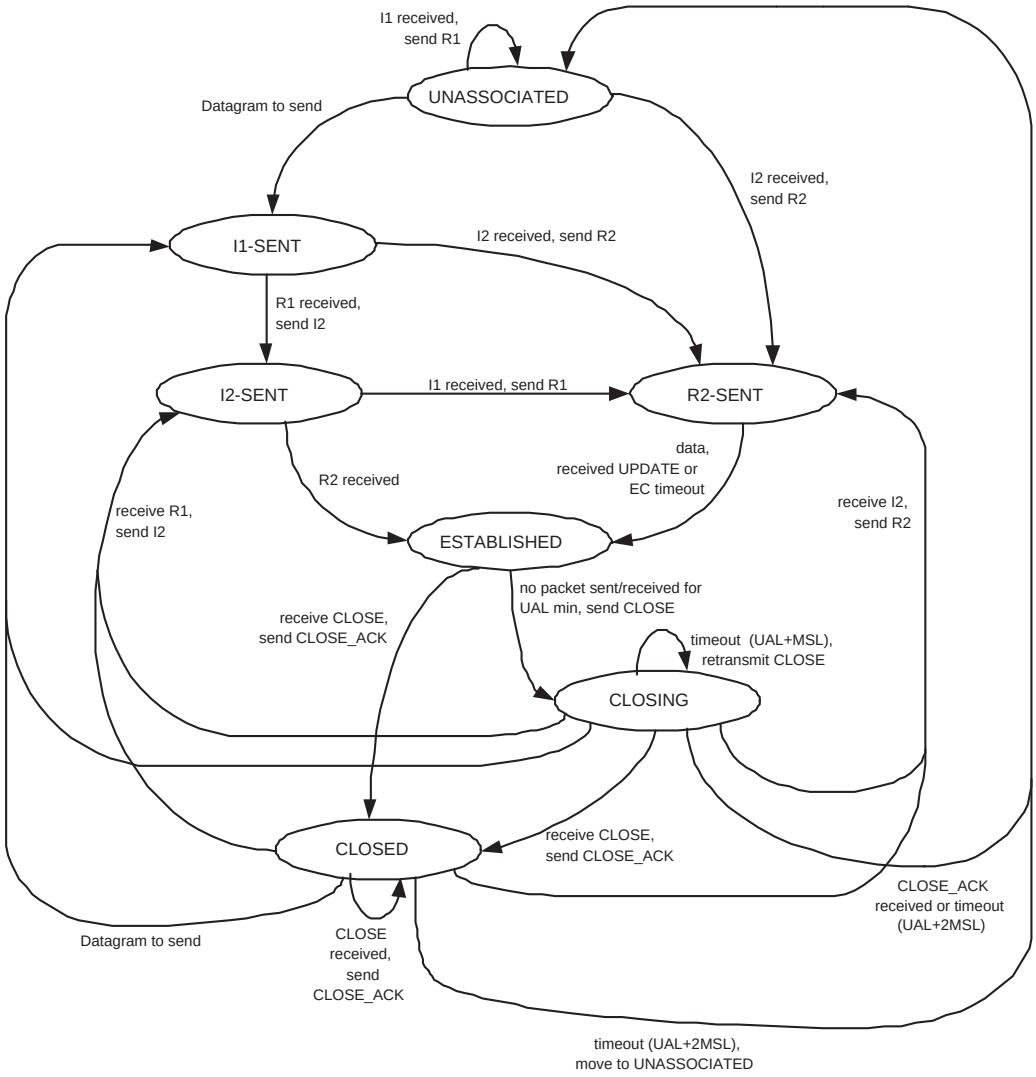


Figure B.1:
HIP state machine.

B.3 HIP state processes

The following tables show the behaviour of the system in every state.

Trigger	Action
User data to send, requiring a new HIP association	Send I1 and go to I1-SENT
Receive I1	Send R1 and stay at UNASSOCIATED
Receive I2, process	If successful, send R2 and go to R2-SENT If fail, stay at UNASSOCIATED
Receive user data for unknown HIP association	Optionally send ICMP and stay at UNASSOCIATED
Receive CLOSE	Optionally send ICMP Parameter Problem and stay at UNASSOCIATED
Receive ANYOTHER	Drop and stay at UNASSOCIATED

Table B.1:
System behaviour in state UNASSOCIATED

Trigger	Action
Receive I1	If the local HIT is smaller than the peer HIT, drop I1 and stay at I1-SENT If the local HIT is greater than the peer HIT, send R1 and stay at I1$_SENT$
Receive I2, process	If successful, send R2 and go to R2-SENT If fail, stay at I1-SENT
Receive R1, process	If successful, send I2 and go to I2-SENT If fail, go to E-FAILED
Receive ANYOTHER	Drop and stay at I1-SENT
Timeout, increment timeout counter	If counter is less than I1_RETRIES_MAX, send I1 and stay at I1-SENT If counter is greater than I1_RETRIES_MAX, go to E-FAILED

Table B.2:
System behaviour in state I1-SENT

Trigger	Action
Receive I1	Send R1 and stay at I2-SENT
Receive R1, process	If successful, send I2 and cycle at I2-SENT If fail, stay at I2-SENT
Receive I2, process	If successful and local HIT is smaller than the peer HIT, drop I2 and stay at I2-SENT If succesful and local HIT is greater than the peer HIT, send R2 and go to R2-SENT If fail, stay at I2-SENT
Receive R2, process	If successful, go to ESTABLISHED If fail, go to E-FAILED
Receive ANYOTHER	Drop and stay at I2-SENT
Timeout, increment timeout counter	If counter is less than I2_RETRIES_MAX, send I2 and stay at I2-SENT If counter is greater than I2_RETRIES_MAX, go to E-FAILED

Table B.3:
System behaviour in state I2-SENT

Trigger	Action
Receive I1	Send R1 and stay at R2-SENT
Receive I2, process	If successful, send R2 and cycle at R2-SENT If fail, stay at R2-SENT
Receive R1	Drop and stay at R2-SENT
Receive R2	Drop and stay at R2-SENT
Receive data or UPDATE	Move to ESTABLISHED
Exchange Complete Timeout	Move to ESTABLISHED

Table B.4:
System behaviour in state R2-SENT

Trigger	Action
Receive I1	Send R1 and stay at ESTABLISHED
Receive I2, process with puzzle and possible Opaque data verification	If successful, send R2, drop old HIP association, establish a new HIP association, go to R2-SENT If fail, stay at ESTABLISHED
Receive R1	Drop and stay at ESTABLISHED
Receive R2	Drop and stay at ESTABLISHED
Receive user data for HIP association	Process and stay at ESTABLISHED
No packet sent/received during UAL minutes	Send CLOSE and go to CLOSING
Receive CLOSE, process	If successful, send CLOSE_ACK and go to CLOSED If fail, stay at ESTABLISHED

Table B.5:
System behaviour in state ESTABLISHED

Trigger	Action
User data to send, requires the creation of another incarnation of the HIP association	Send I1 and stay at CLOSING
Receive I1	Send R1 and stay at CLOSING
Receive I2, process	If successful, send R2 and go to R2-SENT If fail, stay at CLOSING
Receive R1, process	If successful, send I2 and go to I2-SENT If fail, stay at CLOSING
Receive CLOSE, process	If successful, send CLOSE_ACK, discard state and go to CLOSED If fail, stay at CLOSING
Receive CLOSE_ACK, process	If successful, discard state and go to UNASSOCIATED If fail, stay at CLOSING
Receive ANYOTHER	Drop and stay at CLOSING
Timeout, increment timeout sum, reset timer	If timeout sum is less than UAL+MSL minutes, retransmit CLOSE and stay at CLOSING If timeout sum is greater than UAL+MSL minutes, go to UNASSOCIATED

Table B.6:
System behaviour in state CLOSING

Trigger	Action
Datagram to send, requires the creation of another incarnation of the HIP association	Send I1, and stay at CLOSED
Receive I1	Send R1 and stay at CLOSED
Receive I2, process	If successful, send R2 and go to R2-SENT If fail, stay at CLOSED
Receive R1, process	If successful, send I2 and go to I2-SENT If fail, stay at CLOSED
Receive CLOSE, process	If successful, send CLOSE_ACK, stay at CLOSED If fail, stay at CLOSED
Receive CLOSE_ACK, process	If successful, discard state and go to UNASSOCIATED If fail, stay at CLOSED
Receive ANYOTHER	Drop and stay at CLOSED
Timeout (UAL+2MSL)	Discard state and go to UNASSOCIATED

Table B.7:
System behaviour in state CLOSED

Trigger	Action
Wait for implementation specific time	Go to UNASSOCIATED. Re-negotiation is possible after moving to UNASSOCIATED state

Table B.8:
System behaviour in state E-FAILED

Appendix C

Security, Cryptography and more

C.1 IPSEC and HIP

IPSEC (IP security) is a standard for securing Internet Protocol (IP) communications by encrypting and/or authenticating all IP packets. IPSEC provides security at the network layer.

IPSEC is a set of cryptographic protocols for (1) securing packet flows and (2) key exchange. Of the former, there are two: Encapsulating Security Payload (ESP) provides authentication, data confidentiality and message integrity; Authentication Header (AH) provides authentication and message integrity, but does not offer confidentiality. Originally AH was only used for integrity and ESP was used only for encryption; authentication functionality was added subsequently to ESP.

In HIP, ESP Security Associations are setup between the HIP nodes during the base exchange. Existing ESP SAs can be also updated later using UPDATE messages. The reason for updating the ESP SA later can be e.g. need for rekeying the SA because of sequence number rollover.

Upon setting up a HIP association, each association is linked to two ESP SAs, one for incoming packets and one for outgoing packets. The Initiator's incoming SA corresponds with the Responder's outgoing one, and viceversa. The Initiator defines the SPI for the former association. This SA is called SA-RI, and the corresponding SPI is called SPI-RI. Respectively, the Responder's incoming SA corresponds with the Initiator's outgoing SA and is called SA-IR, with the SPI being called SPI-IR.

The Initiator creates SA-RI as a part of R1 processing, before sending out the I2. The keys are derived from the key material created after the

Diffie-Hellman key exchange. The Responder creates SA-RI as a part of I2 processing.

The Responder creates SA-IR as a part of I2 processing, before sending out R2. The Initiator creates SA-IR when processing R2.

When the HIP association is removed, the related ESP SAs must also be removed.

C.1.1 Security Parameter Index (SPI)

An SPI is an arbitrary value that uniquely identifies which SA to use at the receiving host. The sending host uses the SPI to identify and select which SA to use to secure every packet while the receiving one uses it to identify and select the encryption algorithm and key used to decrypt packets.

Thus, representations of host identity are not carried explicitly in the headers of user data packets. Instead, the ESP Security Parameter Index (SPI) is used to indicate the right host context. The ESP SPIs have added significance when used with HIP; they are a compressed representation of a pair of HITs.

The SPI selection should be random and a different SPI should be used for each HIP exchange with a particular host; this is to avoid a replay attack. Additionally, when a host rekeys, the SPI must be changed as well as when a host changes over to use a different IP address.

One method for SPI creation could be to concatenate the HIT with a 32-bit random or sequential number, hash this (using SHA1), and then use the high order 32 bits as the SPI.

The selected SPI is communicated to the peer in the third (I2) and fourth (R2) packets of the base HIP exchange. Changes in SPI are signaled with ESP_INFO parameters.

C.2 Diffie-Hellman key exchange description

The simplest, and original, implementation of the protocol uses the multiplicative group of integers modulo p, where p is prime and g is primitive mod p. Modulo (or mod) simply means that the integers between 1 and p - 1 are used with normal multiplication, exponentiation and division, except that after each operation the result keeps only the remainder after dividing by p. Here is an example of the protocol:

1. Alice and Bob agree to use a prime number p=23 and base g=5.
2. Alice chooses a secret integer a=6, then sends Bob (g^a mod p)

5^6 mod 23 = 8.

3. Bob chooses a secret integer b=15, then sends Alice (g^b mod p)
5^{15} mod 23 = 19.
4. Alice computes (g^b mod p)^a mod p
19^6 mod 23 = 2.
5. Bob computes (g^a mod p)^b mod p
8^{15} mod 23 = 2.

Both Alice and Bob have arrived at the same value, because g^{ab} and g^{ba} are equal. Note that only a, b and $g^{ab} = g^{ba}$ are kept secret. All the other values are sent in the clear. Once Alice and Bob compute the shared secret they can use it as an encryption key, known only to them, for sending messages across the same open communications channel. Of course, much larger values of a, b, and p would be needed to make this example secure, since it is easy to try all the possible values of g^{ab} mod 23 (there will be, at most, 22 such values, even if a and b are large). If p was a prime of more than 300 digits, and a and b were at least 100 digits long, then even the best known algorithms for finding a given only g, p, and g^a mod p (known as the discrete logarithm problem) would take longer than the lifetime of the universe to run. g need not be large at all, and in practice is usually either 2 or 5.

Here's a more general description of the protocol:

1. Alice and Bob agree on a finite cyclic group G and a generating element g in G. (This is usually done long before the rest of the protocol; g is assumed to be known by all attackers.) We will write the group G multiplicatively.
2. Alice picks a random natural number a and sends ga to Bob.
3. Bob picks a random natural number b and sends gb to Alice.
4. Alice computes $(g^b)^a$.
5. Bob computes $(g^a)^b$.

Both Alice and Bob are now in possession of the group element gab which can serve as the shared secret key. The values of $(g^a)^b$ and $(g^b)^a$ are the same because groups are power associative.

C.3 SHA1

Secure Hash Algorithm 1 is a popular one-way hash algorithm used to create digital signatures. SHA was developed by the NIST, and SHA1 is a revision to the standard released in 1994. SHA1 is similar to the MD4 and MD5 algorithms developed by Rivest, but it is slightly slower and more secure.

The Secure Hash Algorithm (SHA1 or SHA-1) is required for use with the Digital Signature Algorithm (DSA) as specified in the Digital Signature Standard (DSS) and whenever a secure hash algorithm is required for federal

applications. For a message of length $< 2^{64}$ bits, the SHA-1 produces a 160-bit condensed representation of the message called a message digest. The message digest is used during generation of a signature for the message. The SHA-1 is also used to compute a message digest for the received version of the message during the process of verifying the signature. Any change to the message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.

The SHA-1 is designed to have the following properties: it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest.

C.4 HMAC

A keyed-hash message authentication code, or HMAC, is a type of message authentication code (MAC) calculated using a cryptographic hash function in combination with a secret key. As with any MAC, it may be used to simultaneously verify both the data integrity and the authenticity of a message. Any iterative cryptographic hash function, such as MD5 or SHA-1, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5 or HMAC-SHA-1 accordingly. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function and on the size and quality of the key.

An iterative hash function breaks up a message into blocks of a fixed size and iterates over them with a compression function. For example, MD5 and SHA-1 operate on 512-bit blocks. The size of the output of HMAC is the same as that of the underlying hash function (128 or 160 bits in the case of MD5 and SHA-1), although it can be truncated if desired.

HMAC is defined as

$$\mathrm{HMAC}_K(m) = h\Big((K \oplus opad) \mid h\big((K \oplus ipad) \mid m\big)\Big),$$

where

h: is an iterated hash function

K: is a secret key padded with extra zeros to the block size of the hash function

m: is the message to be authenticated.

$|$: denotes concatenation

$\oplus$: denotes exclusive or.

The two constants ipad and opad each one block long are defined as

ipad = 0x363636...3636

opad = 0x5c5c5c...5c5c

That is, if block size of the hash function is 512 ipad and opad are 64 repetitions of the (hexadecimal) bytes 0x36 and 0x5c respectively.

Appendix D

Transmission Control Protocol

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite. Using TCP, applications on networked hosts can create connections to one another, over which they can exchange data or packets. The protocol guarantees reliable and in-order delivery of sender to receiver data. TCP also distinguishes data for multiple, concurrent applications (e.g. Web server and email server) running on the same host.

TCP supports many of the Internet's most popular application protocols and resulting applications, including the World Wide Web, email and Secure Shell.

In the Internet protocol suite, TCP is the intermediate layer between the Internet Protocol below it, and an application above it. Applications often need reliable pipe-like connections to each other, whereas the Internet Protocol does not provide such streams, but rather only unreliable packets. TCP does the task of the transport layer in the simplified OSI model of computer networks.

Applications send streams of octets (8-bit bytes) to TCP for delivery through the network, and TCP divides the byte stream into appropriately sized segments (usually delineated by the maximum transmission unit (MTU) size of the data link layer of the network the computer is attached to). TCP then passes the resulting packets to the Internet Protocol, for delivery through the internet to the TCP module of the entity at the other end. TCP checks to make sure that no packets are lost by giving each packet a sequence number, which is also used to make sure that the data are delivered to the entity at the other end in the correct order. The TCP module at the far end sends back an acknowledgement for packets which have been successfully received; a timer at the sending TCP will cause a timeout if an acknowledgement is not received within a reasonable round-trip time (or RTT), and the (presumably lost) data will then be re-transmitted. The TCP

checks that no bytes are damaged by using a checksum; one is computed at the sender for each block of data before it is sent, and checked at the receiver.

Unlike TCP's traditional counterpart User Datagram Protocol that can immediately start sending packets, TCP requires a connection establishment before sending data and a connection termination on completion of sending data. More succinctly, TCP connections have three phases:

1. connection establishment
2. data transfer
3. connection termination

Before describing these three phases, a note about the various states of a socket:

1. LISTEN
2. SYN-SENT
3. SYN-RECEIVED
4. ESTABLISHED
5. FIN-WAIT-1
6. FIN-WAIT-2
7. CLOSE-WAIT
8. CLOSING
9. LAST-ACK
10. TIME-WAIT
11. CLOSED

LISTEN: represents waiting for a connection request from any remote TCP and port. (usually set by TCP servers)

SYN-SENT: represents waiting for the remote TCP to send back a TCP packet with the SYN and ACK flags set. (usually set by TCP clients)

SYN-RECEIVED: represents waiting for the remote TCP to send back an acknowledgment after have sent back a connection acknowledgment to the remote TCP. (usually set by TCP servers)

ESTABLISHED: represents that the port is ready to receive/send data from/to the remote TCP. (set by TCP clients and servers)

TIME-WAIT: represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request. According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes.

D.1 Connection establishment

To establish a connection, TCP uses a 3-way handshake. Before a client attempts to connect with a server, the server must first bind to a port to

open it up for connections: this is called a passive open. Once the passive open is established then a client may initiate an active open. To establish a connection, the 3-way (or 3-step) handshake occurs:

1. The active open is performed by sending a SYN to the server.
2. In response, the server replies with a SYN-ACK.
3. Finally the client sends an ACK back to the server.

At this point, both the client and server have received an acknowledgement of the connection.

Example:

1. The initiating host (client) sends a synchronization (SYN flag set) packet to initiate a connection. Any SYN packet holds Sequence Number. The Sequence Number is a 32 bit field TCP segment header. For example let the Sequence Number value for this session be x.

2. The other host receives the packet, records the Sequence Number of x from the client, and replies with an acknowledgment (ACK). The Acknowledgment Number is a 32 bit field in TCP segment header. It contains the next sequence number that this host is expecting to receive (x + 1).The host also initiates a return session. This includes a TCP segment with its own initial Sequence Number value of y.

3. The initiating host responds with a next Sequence Number (x+1) and a simple Acknowledgment Number value of y + 1, which is the Sequence Number value of the other host + 1.

D.2 Data transfer

There are a few key features that set TCP apart from UDP:

* Error-free data transfer
* Ordered-data transfer
* Retransmission of lost packets
* Discarding duplicate packets
* Congestion throttling

In the first two steps of the 3-way handshaking, both computers exchange an initial sequence number (ISN). This number can be arbitrary. This sequence number identifies the order of the bytes sent from each computer so that the data transferred is order regardless of any fragmentation or misordering that occurs during transmission. For every byte transmitted the sequence number must be incremented.

Conceptually, each byte sent is assigned a sequence number and the receiver then sends an acknowledgement back to the sender that effectively states that they received it. What is done in practice is only the first data

byte is assigned a sequence number which is inserted in the sequence number field and the receiver sends an acknowledgement value of the next byte they expect to receive.

For example, if computer A sends 4 bytes with a sequence number of 100 (conceptually, the four bytes would have a sequence number of 100, 101, 102, 103 assigned) then the receiver would send back an acknowledgement of 104 since that is the next byte it expects to receive in the next packet. By sending an acknowledgement of 104, the receiver is signaling that it received bytes 100, 101, 102, 103 correctly. If, by some chance, the last two bytes were corrupted then an acknowledgement value of 102 would be sent since 100 101 were received successfully.

This would not happen for a packet of 4 bytes but it can happen if, for example, 10,000 bytes are sent in 10 different TCP packets and a packet is lost during transmission. If the first packet is lost then the sender would have to resend all 10,000 bytes since the acknowledgement cannot say that it received bytes 1,000 to 10,000 but only that it expects byte 0 because 0 through 9,999 were lost. (This issue is address in SCTP by adding a selective acknowledgement.)

Sequence numbers and acknowledgments cover discarding duplicate packets, retransmission of lost packets, and ordered-data transfer. To assure correctness a checksum field is included.

The TCP checksum is a quite weak check by modern standards. Data Link Layers with a high probability of bit error rates may require additional link error correction/detection capabilities. If TCP were to be redesigned today, it would most probably have a 32-bit cyclic redundancy check specified as an error check instead of the current checksum. The weak checksum is partially compensated for by the common use of a CRC or better integrity check at layer 2, below both TCP and IP, such as is used in PPP or the Ethernet frame. However, this does not mean that the 16-bit TCP checksum is redundant: remarkably, surveys of Internet traffic have shown that software and hardware errors that introduce errors in packets between CRC-protected hops are common, and that the end-to-end 16-bit TCP checksum catches most of these simple errors. This is the end-to-end principle at work.

The final part to TCP is congestion throttling.

Acknowledgements for data sent, or lack of acknowledgements, are used by senders to implicitly interpret network conditions between the TCP sender and receiver. Coupled with timers, TCP senders and receivers can alter the behavior of the flow of data. This is more generally referred to as flow control, congestion control and/or network congestion avoidance. TCP uses a number of mechanisms to achieve high performance and avoid congesting the network (i.e., send data faster than either the network, or the host on

the other end, can utilize it). These mechanisms include the use of a sliding window, the slow-start algorithm, the congestion avoidance algorithm, the fast retransmit and fast recovery algorithms, and more. Enhancing TCP to reliably handle loss, minimize errors, manage congestion and go fast in very high-speed environments are ongoing areas of research and standards development.

D.2.1 TCP window size

TCP sequence numbers and windows behave very much like a clock. The window, whose width (in bytes) is defined by the receiving host, shifts each time it receives and acks a segment of data. Once it runs out of sequence numbers, it loops back to 0.

TCP sequence numbers and windows behave very much like a clock. The window, whose width (in bytes) is defined by the receiving host, shifts each time it receives and acks a segment of data. Once it runs out of sequence numbers, it loops back to 0.

The TCP receive window size is the amount of received data (in bytes) that can be buffered during a connection. The sending host can send only that amount of data before it must wait for an acknowledgment and window update from the receiving host.

D.2.2 Window scaling

For more efficient use of high bandwidth networks, a larger TCP window size may be used. The TCP window size field controls the flow of data and is limited to 2 bytes, or a window size of 65,535 bytes.

Since the size field cannot be expanded, a scaling factor is used. TCP window scale, as defined in RFC 1323, is an option used to increase the maximum window size from 65,535 bytes to 1 Gigabyte. Scaling up to larger window sizes is a part of what is necessary for TCP Tuning.

The window scale option is used only during the TCP 3-way handshake. The window scale value represents the number of bits to left-shift the 16-bit window size field. The window scale value can be set from 0 (no shift) to 14.

D.3 Connection termination

The connection termination phase uses, at most, a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the

other end acknowledges with an ACK. Therefore, a typical teardown requires a pair of FIN and ACK segments from each TCP endpoint.

A connection can be "half-open", in which case one side has terminated its end, but the other has not. The side which has terminated can no longer send any data into the connection, but the other side can.

It is also possible for a 3-way handshake when host A sends a FIN and host B replies with a FIN and ACK (merely combines 2 steps into one) and host A replies with an ACK. This is perhaps the most common method.

Finally, it is possible for both hosts to send FINs simultaneously then both just have to ACK. This could possibly be considered a 2-way handshake since the FIN/ACK sequence is done in parallel for both directions.

Appendix E

Submitted patches

Added in the CD version.

E.1 Registration and Rendezvous Server implementations

Submitted in release 0.3.2

E.2 Performance improvement

Submitted in release 0.3.2

E.3 New transition between R2-SENT and ESTABLISHED HIP-states

Submitted in release 0.3.1