Implementation and evaluation of the Shim6 protocol in the Linux kernel

S. Barré[a,*], J. Ronan[b], O. Bonaventure[a]

[a] *Université catholique de Louvain*
B-1348 Louvain-la-Neuve, Belgium
[b] *Telecommunications Software & Systems Group*
Waterford Institute of Technology, Cork Road, Waterford, Ireland

Abstract

In the changing landscape of the todays Internet, several solutions are under investigation to allow efficient, flexible and scalable multihoming. One of the proposals is `shim6`, a host-based multihoming solution based on the use of multiple IPv6 addresses on each host. In this work, we first describe the main features of this protocol, then we explain our implementation of `shim6`, along with the associated security mechanisms in the Linux kernel and, finally, we evaluate its performance. In particular, we analyse the performance impact of the security mechanisms used by `shim6` and the impact of `shim6` on the performance of end-host systems, especially heavily loaded servers. We conclude by discussing the remaining open issues for a widespread deployment of host-based multihoming techniques such as `shim6`.

Keywords: Shim6, IPv6, Multihoming

1. Introduction

The current IPv4 Internet is facing several challenges. Firstly, the IP version 4 (IPv4) address space is limited and the latest projections [1] indicate that during the year 2011 all IPv4 addresses will have been allocated. Secondly, operators and researchers are becoming more and more concerned about the limits on the scalability of the current Internet architecture [31].

For a number of years, several groups have tried to address these problems. Within the IETF, the work on the development of a replacement for IPv4 started more than 15 years ago with the work on IP next generation. This initiated the development of IP version 6 (IPv6) that was expected to replace IP version 4 before the beginning of this century. Today, IPv6 is now supported by most

*Corresponding author
Email addresses: `sebastien.barre@uclouvain.be` (S. Barré), `jronan@tssg.org` (J. Ronan), `olivier.bonaventure@uclouvain.be` (O. Bonaventure)

[1] See e.g. `http://www.potaroo.net/tools/ipv4/index.html`

host and server operating systems. However deployment by network operators is still limited but appears to have been growing recently [25]. We can thus expect that IPv6 will gain more and more importance over the next few years.

On the other hand, the Internet Architecture Board (IAB) has identified several limitations of the current Internet architecture [31]. The first problem is the scalability of the interdomain routing system. This is reflected by the growth of the BGP routing tables and also the growth in the number of messages processed by BGP routers. This routing scalability issue is caused by several main factors. An initial contributor is multihoming, i.e. when an IP network is attached to several Internet Service Providers that need to advertise the corresponding prefix to the global Internet. Another contributor to the growth of the BGP routing tables are the various BGP-based traffic engineering techniques used by network operators to control the flow of their Internet traffic [20, 41]. Finally, the allocation of IP addresses also contributes to the BGP growth. In the early days of the Internet, IP address blocks were allocated on a first-come first-served basis. This led to a huge consumption of address blocks that are almost impossible to aggregate. Since the introduction of Classless Interdomain Routing (CIDR), IP address blocks are allocated by Regional Registries (RIRs). There are two types of address block allocations: Provider Independent (PI) and Provider Aggregatable (PA). In the early days, PI address blocks were reserved for Internet Service Providers and customer networks could not obtain such address blocks directly from the RIRs. This allocation policy assumed that customer networks would be single homed and that they would renumber their network each time they change provider. These assumptions do not hold anymore and many enterprise networks insist on obtaining PI address blocks, which contributes to the growth of the BGP routing tables [30].

The second problem is the overloading of IP address semantics. IP addresses are used for two different purposes: identifiers and locators. In its identifier role, an IP address, combined with a port number, identifies an endpoint of a transport flow. In its locator role, an IP address identifies the paths to reach a host via one of its interfaces through a network.

The large IPv6 address space offers several opportunities to solve these problems differently than with IPv4. Several years ago, after evaluating many alternatives [17, 23], the IETF chartered the `shim6` working group to develop a host-based IPv6 multihoming solution [37]. The `shim6` specifications are now ready and, in this paper, we report our experience with one of the first complete and publicly available implementations of this IPv6 multihoming technique in the Linux kernel.

This paper is organised as follows. First, as `shim6` is not yet widely known, we describe its key features and benefits in section 2. This is followed by a description of the architecture of our LinShim6 implementation in section 3 and an evaluation of the performance of several of its key mechanisms. We conclude by reflecting on the evolution of host-based multihoming techniques based on our experience with `shim6` in section 4 and a discussion of related work in section 5.

2. Shim6 host-based IPv6 multihoming

Before delving into the details of `shim6`, consider that there are at least two scenarios that can provide multihoming. The first type is when a single host has two or more IPv6 addresses from two or more layer-2 interfaces connected to separate networks. This can be the case of a laptop having both WiFi and 3G Internet interfaces, or servers having multiple Ethernet interfaces. In these cases, the multihomed host would like to either be able to efficiently use both interfaces simultaneously or use a primary interface, with automatic redirection of all packets over another interface upon failure of the primary one.

The second type of multihoming occurs when a campus, corporate or ISP network is attached to two different service providers. In such a network, each host gets an address from each service provider, and is accessible over both. A host in such a multihomed network can select, for itself, the provider to use for a given flow, through appropriate selection of the source address. `Shim6` was designed with the latter form of multihoming in mind but also supports the former.

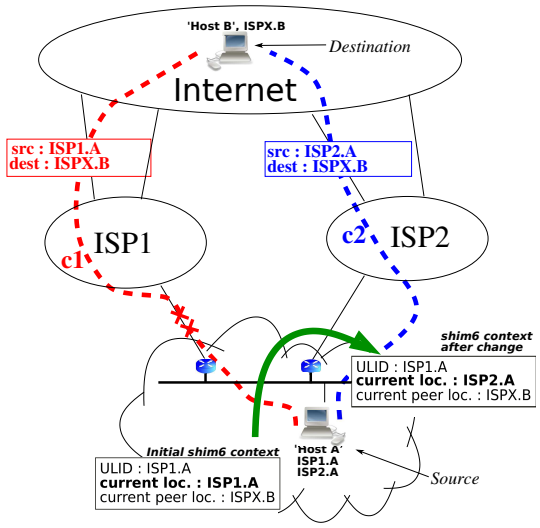

Figure 1: Basic operation of a `shim6` host

Today, in the IPv4 Internet, when a network is multihomed, it receives one IPv4 address range, and uses BGP to advertise its IPv4 prefix to its upstream providers which, in turn, advertise the network to the global Internet. This contributes to the growth of the BGP routing tables. If a link between the multihomed network and one of its providers fails, BGP re-converges, to ensure that the multihomed network remains reachable via its other providers. However, a network relying on `shim6` for its multihoming behaves differently. The main difference from IPv4 multihoming is that each `shim6` host has several IPv6 addresses, one from each of its providers or one on each of its interfaces. This is illustrated in Figure 1. The corporate network shown at the bottom of the figure

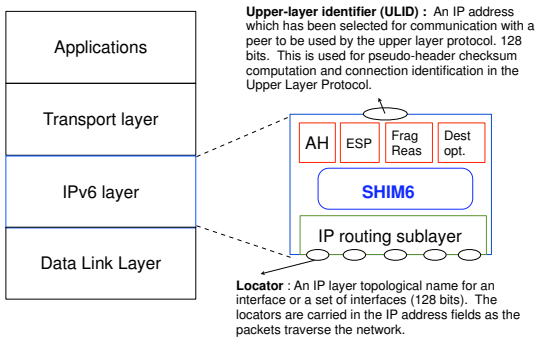

Figure 2: Networking stack with `shim6`

is attached to `ISP1` and `ISP2`. Each ISP has allocated a prefix to the corporate network. Each `shim6` host has one IPv6 address inside each of these subnets. From a BGP routing table viewpoint, the main advantage of `shim6` host-based multihoming is that `AS1` and `AS2` only need to advertise their global `/32` IPv6 prefix and not the more specific prefixes allocated to their customers. However, this also implies that if the link between the corporate network and `ISP1` fails, BGP will not announce the failure to the global Internet. This problem is solved in `shim6` by using a new failure detection and recovery mechanism, the `REAP` protocol[4], that allows `shim6` hosts to detect a failure and switch traffic to an available working path.

In the following subsections, first we describe the `shim6` architecture, then explain how `shim6` solves the security issues and finally describe the `REAP` protocol.

2.1. Shim6

A `shim6` host has several IPv6 addresses. All these addresses are locators, i.e. they identify where a network interface is located within the global routing context. For example, in Figure 1, a packet whose destination is `ISP1.A` will be delivered via `ISP1`. On the other hand, a packet whose destination is `ISP2.A` will be delivered via `ISP2`. As current best practice [9] recommends that ISPs verify the source address of packets received from their customers: a packet produced by host `A` that contains `ISP1.A` as its source address must always be sent via `ISP1`. Such a packet will never be forwarded by `ISP2`.

When an application on host A contacts an application on host B using an upper-layer protocol (ULP), the default address selected [18] by host A is determined to be the upper-layer identifier (ULID) to identify the transport flows between the hosts. Conceptually, the `shim6` sublayer belongs to the network layer and the locators are attached to the lower part of the network layer while the identifier is attached to the upper part of the network layer (Figure 2).

The main purpose of `shim6` is to preserve established flows in spite of network failures, while operating transparently to upper-layer protocols such as TCP or UDP. This is illustrated in Figure 1. Host A has established a flow between

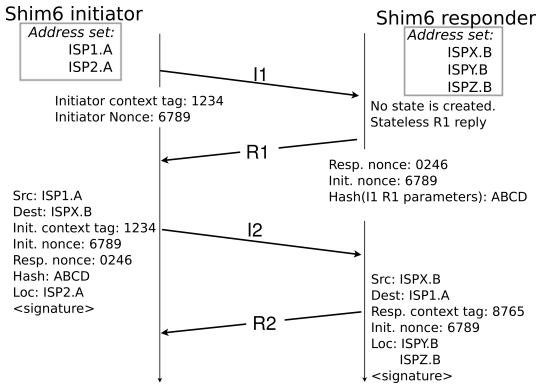

Figure 3: `Shim6` session establishment

ULID `ISP1.A` and destination `ISPX.B`. In addition to its ULID, host A also has the `ISP2.A` locator. Upon failure of the path between `ISP1.A` and `ISPX.B`, host A will use `shim6` to switch its flow on the `ISP2.A`→`ISPX.B` path. For this, all of host A's packets destined to `ISPX.B` must be sent from source `ISP2.A`. `Shim6` ensures the transparency of this operation to the applications.

The `shim6` sublayer performs three different tasks. Firstly, two communicating `shim6` hosts need to discover their respective locator sets. This is performed during the establishment of the `shim6` session. Secondly, during the lifetime of a flow, it may be necessary to switch from the current path to an alternate, e.g. after a failure. Thirdly, `shim6` can be used to advertise any change in the set of locators available on a host.

The first task is conducted at the beginning of a communication. When an application is requested to initiate an exchange towards a host (i.e. a http or other such request), the usual process is that its name is looked up from the Domain Name System (DNS). The DNS answers with one or several addresses. The application then initiates a connection with one of the obtained addresses (through default address selection)[18, 35, 36].

A heuristic on one of the `shim6`-enabled hosts determines whether it is worth the extra `shim6` overhead to protect the communication flow. In the case where the host decides that it is worth the effort, the end hosts communicate to each other their entire set of locators. This is the `shim6` initial exchange. After this negotiation, each host has a set of local and peer addresses that it can use to carry packets.

The establishment of a `shim6` session is performed by using a four-way handshake as shown in Figure 3. This handshake is based on the handshake used by HIP [33]. It was designed [37] to protect against replay attacks, to ensure that all announced addresses belong to the same peer, and to protect against Denial-of-Service (DoS) attacks. More details about how and why the messages are exchanged this manner are available in [37, §7]. However, to understand this paper, it is sufficient to know that the most costly operation in a `shim6`

negotiation (in terms of processing power) is the address signature (which can be done in advance or offline) and verification. These operations are explained in the next subsection.

Typically the second task is triggered when the associated `REAP` protocol has detected a failure and found an alternate working locator pair. More generally, any appropriately interfaced entity (an application interface for example [28]) could trigger a path change. Changing the path in the course of a communication is made possible by rewriting the address pair in use. Obviously one particular path is the one corresponding to the ULIDs. In this case the ULIDs and the locators are identical and no rewriting is needed. In all other cases, rewriting is needed and an extension header is added to the outgoing packets. The extension header contains a *context tag* used to identify the flow at the receiver, so that locators can be replaced by the correct ULIDs in the receiver.

The rewriting function of `shim6` is located in a new IP-sublayer in the networking stack, as shown in Figure 2. Anything located above the `shim6` layer sees stable addresses (ULIDs). This includes parts of the IP layer such as IPsec or fragmentation, so that those functions can operate on stable ULIDs, even though `shim6` may have had to rewrite the packet header. Conversely, the forwarding functionality of the IP layer must be located below the `shim6` layer, so that the locators chosen by `shim6` are correctly used to select a path. The effect of address rewriting over the chosen path is illustrated in Figure 1.

The third task is locator update. This is useful if a new locator appears after the initial exchange, that is, after the set of locators has been announced by each peer. This could happen should another Ethernet or WiFi interface become operational. If a locator appears or disappears on a host, it is possible to tell the peer about an updated locator set, so that changes in available paths are taken into account. These locator updates may be useful in some IP mobility scenarios [42].

In task one, we briefly mention the use of cryptographic mechanisms. These mechanisms have been carefully designed [37] to prevent an attacker from injecting fake addresses, and thus use this attack vector as a basis for new types of attacks. We summarize the critical parts in the next subsection, and address them in more detail later in the paper.

2.2. Securing locator sets

A key problem faced by host-based techniques that rely on multiple locators is that the receiver of that packet must be able to verify the origin of a packet that uses a new locator. RFC4218 [38] describes in details the threats that must be considered while designing any IPv6 multihoming solution. The way `shim6` responds to those attacks is described in section 16 of [37]. While the solution to many of the threats resides in using well-known protection mechanisms, one particular type of attack, namely address injection, is addressed by a new mechanism that is worth describing here. Address injection consists of an attacker presenting a modified address set to one of the communicating hosts (either by sending fake announcements or modifying existing packets).

The first option proposed by [37] to solve this issue is to use Cryptographically Generated Addresses (CGAs) [6]. This method relies on the use of a signature to prove that all signed addresses have been generated by the same entity. Rather than using certificates to bind an identity to the signature (which would require a trusted third-party to sign the certificate), the CGA approach is to bind the ULID itself to the signature. For a given public/private key pair, the private key is used to sign the locator set, while the public key is hashed so as to generate the 64 low order bits of the ULID. The length of the hash is *artificially extended* (see [6]) so that the actual hash length is not 64 bits, but instead $59 + 16 * sec$ bits (where *sec* is a tunable parameter). Consequently, the security is dependent on an attacker not being able to find a hash collision with a self-generated public key. This property of the hash function is called "pre-image resistance", and the time needed to find a collision when the *sec* parameter is as low as 1 makes such an attack infeasible in short timescales. Over time, when Moores law does eventually make such an attack practical, or for servers that keep a stable address over time, the attack complexity can be increased further by incrementing the *security parameter* (*sec*) on the host that generates the signature [6]. With each increase in the *security parameter*, the complexity required to generate a collision will increase by 2^{16*sec} iterations. This increases the cost of address generation, and thus of brute-force attacks, while keeping the cost of address verification constant.

The second option is to bind all addresses together, without using a signature. This type of address is called Hash Based Addresses (HBA) [7]. The 64 low order bits of each address is the result of a hash computation over all the prefixes of the set. An attacker who wishes to inject his own address into the locator set would need to find an input to the hash function that produces, at least, the locator used for forwarding as part of the generated locator set. Since this is made easier by the short length of the hash, HBA uses the same *security parameter* as CGA to tune the cryptographical strength of the locator set.

HBA is computationally cheaper than CGA, but it also has less flexibility. Its main drawback is that the addition of a new address in a locator set requires regenerating the whole set. This is where CGA-compatible HBA addresses are useful. In that case the hash input includes both a public key and the set of prefixes. This is initially seen by the peer as normal HBA addresses, but if a new prefix must be added afterwards, it can be signed with the public key.

With both these mechanims in place, `shim6`, as part of the initial context establishment, verifies that the host claiming to be representing `ISP2.A` (for example, see Figure 3) can be cryptographically tied to that locator (using either the CGA or HBA mechanisms). While an attacker can generate a new address from a subnet prefix and a public key, this attacker cannot impersonate another hosts address. This is, of course, based on the premise that it is currently beyond the capability of an attacker to harness enough computing power to generate a collision in either the HBA or CGA hash functions.

2.3. Failure detection and recovery

`REAP` is responsible for suggesting to `shim6` when to change the current path, as well as for finding an alternate path when the current one becomes unavailable. `REAP` is closely tied to `shim6` because it uses its state to monitor the active flows. `REAP` can be divided in two main features: *Flow monitoring* and *path exploration*.

Flow monitoring is started immediately after the `shim6` initial locator discovery. It is designed in such a way as to minimise the amount of active probing. The main mechanism that allows for reaching that goal is called Forced Bidirectional Detection (FBD). The communication is forced to be bidirectional in the sense that if an end-host receives Upper Layer Protocol (ULP) data, but does not send anything, then control packets (keepalives) are automatically generated. Given this, it can be concluded that a failure has occurred if a host is sending ULP packets without receiving back any data or keepalives[2]. A host decides that a failure has occurred if its *Send Timer* expires. The default expiry time $Tsend$ is defined as $15s$ in [4]. That timer is stopped whenever a packet enters the networking stack. In addition to the *Send timer*, a host maintains a *Keepalive Timer*, that sends a keepalive packet on expiry. This is to ensure that the peer does not think that a failure occurred when in fact the application just stopped sending data. The requirement for a *Keepalive Timer* is to have an expiry time that verifies $Tka + \textit{one-way delay} < TSend$. Currently we set Tka as one third of $Tsend$, as recommended in [4].

The second feature of `REAP` is path exploration. Due to its flow monitoring capability, `REAP` can react to failures by probing the known paths (address combinations). The probing process, described in detail in [4], allows for finding an alternate working path, for each direction of the communication. It can even result in the use of different paths for each direction, as it is able to detect unidirectional paths.

3. Implementation

In this section, we first describe the architecture of our LinShim6 implementation. Then we use measurements to evaluate its performance and motivate some of our design choices. We also take into account related simulation-based studies [15], and provide further insight on those results based on our implementation.

3.1. Architecture overview

We have explained already that `shim6` is transparent to applications, however, it should also be economical with system resources. While any implementation will, of necessity, consume resources, efforts to minimise this have been

[2]Note that the number of control packets is kept minimal since no keepalive is needed if data exchange is either bidirectional or paused.

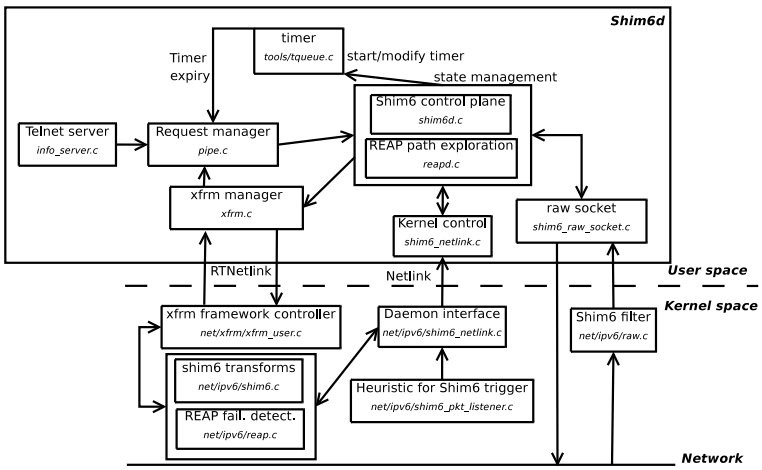

Figure 4: `Shim6` overall architecture

made in LinShim6. This means that special care was taken with parameters that may impact performance. Those parameters are mainly identified as *per-packet processing*, *state management* and *cryptographic operations*. In this section we describe each of those parameters and the approach taken for the LinShim6 implementation. Note that the following explanations are based on Figures 4 and 5. While Figure 4 shows the module interconnections, Figure 5 clarifies the particular ations undertaken by each of them (described below), based on a common scenario.

Considering the separation between kernel space and user space, *per-packet processing* must be kept in the kernel, as doing otherwise would be very expensive in terms of performance, requiring a context switch for each packet that enters or leaves the system. On the other hand, *state management* and *cryptographic operations* are better placed in user space, to allow for ease of maintenance and to avoid kernel bloat.

Per-packet processing can itself be further sub-divided into ***`shim6`*** *trigger heuristic*, *REAP flow monitoring* and *address rewriting*. For efficiency reasons, all three functions are implemented in the kernel. The ***`shim6`*** *trigger heuristic* uses the Netfilter hooks `IP6_LOCAL_IN` and `IP6_LOCAL_OUT`. However, LinShim6 does not initiate a new `shim6` session each time a packet is sent to a new destination. For each new flow, a `shim6` negotiation is triggered if either *Trigger* bytes of data have passed (we suggest a default of $Trigger{=}2KB$ based on our measurements), or one minute has elapsed with data flowing between two hosts. These values were chosen because they avoid engaging `shim6` unless the data flow is either significantly large, or of long enough duration to warrant it. *REAP flow monitoring* and address rewriting both use the `xfrm` framework. This framework [50] allows for ease of integration of new layers in the networking stack. `xfrm` (meaning *transform*) is designed to allow efficient

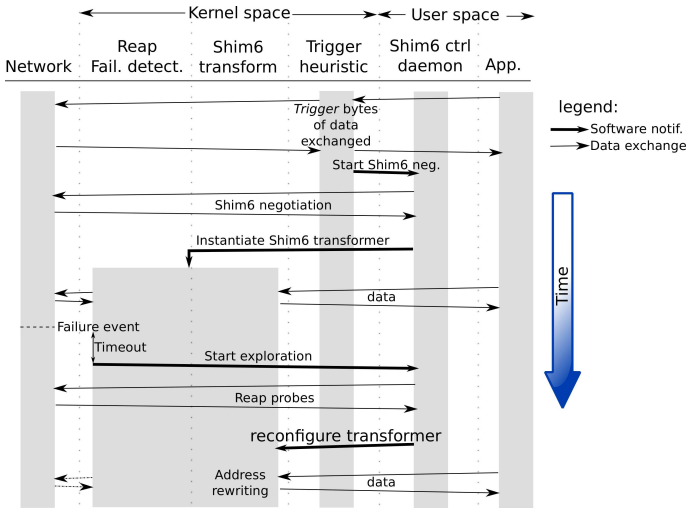

Figure 5: LinShim6 example sequence diagram

support for multiple transformations of packets in any order. Originally created for IPsec, where packets must be modified according to certain policies, this framework was later used for Mobile IPv6. In LinShim6 we defined a new type of `xfrm` transformer, capable of rewriting addresses, inserting the `shim6` extension header and detecting failures. Furthermore, we have implemented a variant of LinShim6, called Multipath LinShim6, that handles several paths simultaneously. We use a more efficient approach for this than the one proposed in [37]. [37] defines a concept of *context forking*, which consists of using one fork of the original Shim6 state per additional path that an application wants to use simultaneously with the first one. This is costly because each additional context needs to be negotiated and occupies space in the system, mostly duplicated from the original context. Instead, our Multipath LinShim6 adds only a few lines of code in the kernel (compared to pure LinShim6) to give the `xfrm` entity the knowledge of **all** paths, and attaching a label (called a path index) to them. Any multipath-aware upper layer (transport or application) can then impose a particular path to Shim6 by attaching a path index to each outgoing packet. An application of Multipath LinShim6 is described in [47, §4].

State management is actually the `shim6` control plane. It starts a `shim6` negotiation when asked by the ***`shim6`*** *trigger heuristic*. When notified about a failure by the `xfrm` module, it starts a REAP exploration, and informs the `xfrm` transformer when a new path has been found, so that the corresponding address rewriting can be performed by the kernel. If a new path is discovered (new prefix available on an interface), it updates the peer host with this information.

Finally, the last task performed by LinShim6 are the *cryptographic operations*. These functions are the most costly from a CPU viewpoint, and must be done as much as possible in advance. Cryptographic operations include the

generation and verification of addresses, as well as signing and verification of important `shim6` messages. In the LinShim6 design, a separate tool allows for CGA/HBA address generation[3]. The tool can be configured to favour either speed of address generation, versus the strength of generated addresses, incurring extra computational cost for the latter. Also, address generation can optionally be run on a configurable number of parallel processor threads, thus taking advantage of multi-core processors. Much of this address generation code was originally written by DoCoMo [26] for SEcure Neighbour Discovery [3], it was integrated into LinShim6 for the purposes of address generation, and thus it is very similar to the SEND implementation, although our tool features HBA generation as well.

Also, for the purposes of LinShim6, the `xfrm` framework has been extended by adding the possibility to link two `xfrm` states that correspond to the two uni-directional communication paths of a data flow. A lookup method based on the context tag has also been added. Details on the working of `xfrm` and our extensions can be found in [12].

3.2. HBA/CGA

CGA and HBA addresses, which are used to secure `shim6`, are an integral part of the protocol. At the time of writing and to the best of our knowledge, LinShim6 is the only implementation with full support of HBA and CGA. Supporting these addresses raises operational and performance challenges. From a performance viewpoint, using CGA and HBA addresses may lower the performance of `shim6` when compared to normal IPv6. HBA and CGA operations are the most computationally costly parts of LinShim6. We evaluate here the computational impact of using those addresses, first at the generation time (can be done offline), then at the signature and generation time (always online).

HBA/CGA address generation: As the CGA addresses depend on a public/private key pair, our implementation automatically generates such a key-pair during its installation. This is done so that LinShim6 will work "out-of-the-box", without complex configuration effort from the user. LinShim6 configures itself automatically with CGA addresses by using this public/private key-pair. It is, of course, perfectly possible to manually configure several public/private key pairs, and also to define any number of HBA-sets. For LinShim6, as CGA addresses can be generated as soon as the host discovers the IPv6 prefix for a network interface, we selected CGA addresses as the default. This implies that CGA addresses are useable on laptops that move regularly, whereas HBA's are not for reasons mentioned previously in 2.2.

As explained in [6], the cost of the CGA generation is of 2^{16*sec} iterations in the worst case, where *sec* is the security parameter (a larger *sec* increases the security but also the time required to generate an address). The same worst case cost applies to HBA generation [7]. The worst case complexity for an attacker

[3]Address generation could potentially be done on a completely separate, more powerful machine

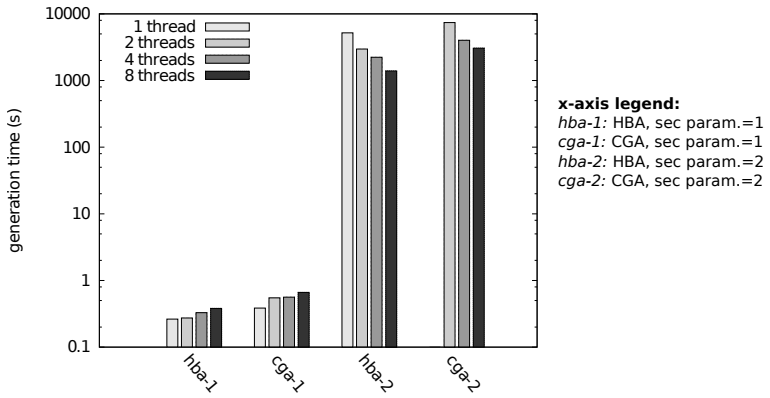

Figure 6: HBA/CGA generation time

to find a matching hash for the address is of the order of $2^{59+16*sec}$ iterations [6], which means that the generator has 2^{59} iterations of an advantage in computational cost over the attacker. That said, in general, as processing power increases one should consider increasing the value of the security parameter to protect against brute-force attacks.

To evaluate the cost of generating HBA and CGA addresses, we used a Sunblade x6440 equipped with 4 AMD Opteron 8431 processors, each with 6 cores, clocked at 2.4GHz. Figure 6 shows the mean time required to generate HBA or CGA addresses, each bar being the mean of 100 trials. Each bar shows the mean generation time of **two** addresses, in log scale. The first two sets of bars are generated with a security parameter of 1. The other bars were generated with a security parameter of 2. It is worth noting that the standard deviation (not shown in the figure) is very large, because of the brute-force algorithm [6] used in the generation process. For the results presented in Figure 6, we observed a standard deviation ranging from 23% to 77% of the reported mean.

The address generation tool [26] is able to use any number of concurrent threads. This capability was retained and extended in order to support HBA. With the security parameter set to 2, multithreading is necessary in order to obtain a result in a reasonable time. Hence using a security parameter of 1 is the only option on current commodity hardware. When the computational time increases due to a required higher security level, it is clearly beneficial to use as many threads as possible (see the results with 8 threads in the figure, and note that the scale is logarithmic). On the other hand, multithreading gives slightly worse results when the security parameter is 1, because the threading overhead takes a higher proportion of the processing time.

While Figure 6 shows the generation time for two addresses, we note that if the number of generated addresses is increased, the CGA generation time increases linearly with the number of addresses. On the other hand, there is a barely perceptible increase in HBA generation time. This is because the expensive part (called *modifier generation* in [6]) is conducted only once for the

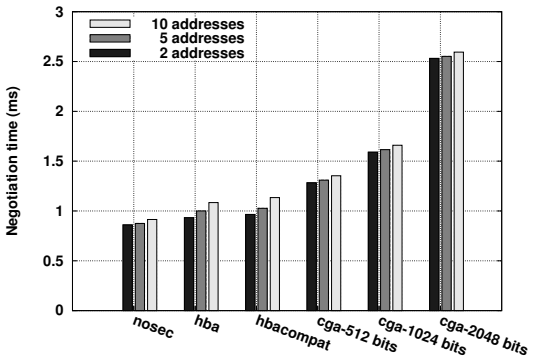

Figure 7: HBA/CGA evaluation

whole set, in the HBA case.

Address signature and verification: Both operations take place during the initialisation of a `shim6` exchange, or when one of the peers announces changes in its locator set. To evaluate the cost of these cryptographical operations we measured the time needed to carry a `shim6` negotiation with different security mechanisms. These tests were performed between two hosts on a 100 Mbps Ethernet. The initiator was a Pentium 4 dual core, 2.6GHz with 1GB RAM while the responder was a Pentium 3, 600MHz with 256MB RAM.

The results are reported in Figure 7. For each security configuration, the case of each peer announcing 2, 5 or 10 addresses in its locator set is compared. The *negotiation time* is defined as the time elapsed between the transmission of the first I1 message, and the reception of the last negotiation message (R2) (see Figure 3). Note that the negotiation time includes two signatures and two verifications, that is, one for each peer. Each bar shows the median negotiation time over 20 consecutive runs.

Looking at the right hand side of Figure 7, we note that there is a strong correlation between the length of the RSA key used for signing messages and the negotiation time. Conversely, HBA addresses involve a negotiation time that is almost the same as if no security were used at all. This is because no signature is needed in the case of HBA addresses.

An important consideration, discussed previously, is that HBA addresses require the knowledge of all the prefixes before commencing a `shim6` negotiation. This motivates the use of CGA-compatible HBA addresses which are defined in [7]. While pure HBA addresses use a random number as input of the SHA-1 hash used during the generation process, CGA-compatible HBA addresses use a public key instead of such a number. However, no signature is needed until the host learns of a new prefix that can be used. At this point a CGA address is generated based on the new prefix, and the key used to generate the previous HBA set. A signed message can be sent to the peer, which will use the already known public key to verify the locator update. In Figure 7, the

bar labelled *hbacompat* shows the negotiation time needed when an HBA set generated based on a 1024-bit public key is used.

This shows very similar results to the use of pure HBA. The small increase in time is explained by the fact that pure-HBA uses a random number formatted as a 384-bit RSA key, as defined in [7]. A final observation is that the impact of the number of announced addresses (2, 5 or 10 in the figure) is insignificant compared to the security mechanism used.

From the above observations, we conclude that from a performance point of view, there is a strong argument to be made for using HBA addresses, or even better, CGA-compatible HBA addresses. LinShim6 allows for generation of HBA/CGA addresses in advance of their use[4]. Once they are generated, they become active only when configured in the system, either manually or by auto-configuration through the *cgad* daemon. By default, LinShim6 disables the standard IPv6 auto-configuration mechanism, in order to avoid having both unsecured addresses and HBA/CGA addresses in the system. This mechanism is replaced by the *cgad* daemon, that listens for Router Advertisements, and configures the appropriate addresses when a new prefix is received.

3.3. Improving failure recovery time

The REAP Failure detection mechanism has been evaluated by simulations in [15]. We have evaluated the performance of the implemented path exploration mechanism in [11]. In [15], de la Oliva et al. emphasise that the TCP exponential backoff has a negative impact on the recovery time seen by an application. The reason is that after a failure, TCP tries to retransmit until a response is received. The delay between successive retransmissions is exponentially increased. Consequently, when REAP finds a new path, TCP unnecessarily waits for its next retransmission before noticing that the communication path is operational again. [15] suggests informing TCP when a new path is found so that it immediately retransmits and recovers. Figure 7 of [15] provides simulation results that show the effect of the improvement.

In LinShim6, the authors have added a mechanism that allows for notifications to be issued when any multihoming event occurs. This uses the Linux `netevent` framework. Any module in the kernel can register for such notifications, without the knowledge of the `shim6` module. This is important for maintaining the layer separation inside the kernel. TCP thus registers for the `PATH_UPDATE` event, and receives a notification when a path has been updated. It reacts by resetting all its RTO (Retransmission TimeOut) timers for the TCP sessions that use that path. LinShim6 is also modular enough to support external control from informed entities (such as network monitoring daemons), for example to force a change to another path. One of us (Ronan) has written such a controlling daemon to take ECN information into account [43].

We tested that implementation in our testbed consisting of a Juniper M10i, as the router, with two Dual Pentium III Blade servers with 512MB of Ram

[4]By default LinShim6 will generate CGA addresses on installation

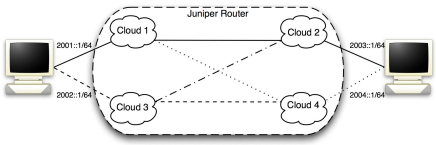

Figure 8: `Shim6` testbed

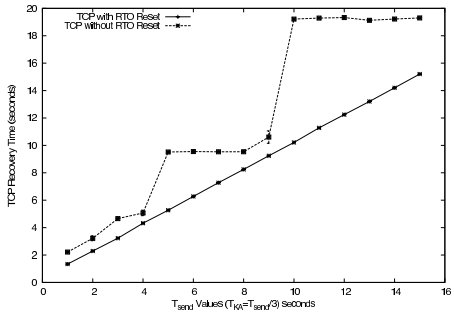

Figure 9: Tsend impact on ART

and Gigabit Interfaces (Figure 8). Both computers were running the IPerf tool for traffic generation. To simulate link failures, links were switched off in the Juniper router via expect scripts. The gain in recovery time is presented in Figure 9. This figure is deliberately very similar to Figure 7 of [15]. The goal was to compare the simulation results with the implementation results. We measure the Application Recovery Time (ART), defined in [15] as the time elapsed between the last packet reception before a failure, and the first packet received after the recovery. The measurement is repeated for different values of $TSend$ (Failure detection timeout). Each point in the figure is the median of 45 measurements performed in the same conditions. Errorbars with percentiles 5 and 95 are also shown.

With our testbed setup, an Application Recovery Time (ART) cannot be below $Tsend$, because the time of the last packet received is almost equal to the time of the last packet sent (due to the configured packet rate), and the path exploration starts $Tsend$ seconds after the last packet has been sent. In the more general case the lower bound for an ART can be slightly lower as explained in [15].

Figure 9 confirms the simulation results from [15]. We observe an ART that increases linearly with $Tsend$ if the RTO (TCP Retransmission TimeOut) is reset. On the other hand, in the absence of RTO reset, we observe steps in the curve that are due to TCP waiting for its RTO before performing a retransmission.

Regarding the path exploration, our implementation separates the address pairs used for sending probes into two sets, each one randomised. The first set contains all pairs that are completely distinct from the current (stalled) address pair. The second set contains all other address pairs. The first probes sent use address pairs from the first set, in the hope that using an orthogonal path increases the chance our implementation can find a working path on the first attempt. Indeed, in the testbed setup, this proved to be the case. This also confirms what was simulated in [15].

While many things are common between our figure and Figure 7 of [15], all our experiments (either with or without RTO reset) reveal a faster ART than the one obtained in [15]. One explanation is that while [15] sends a probe to the

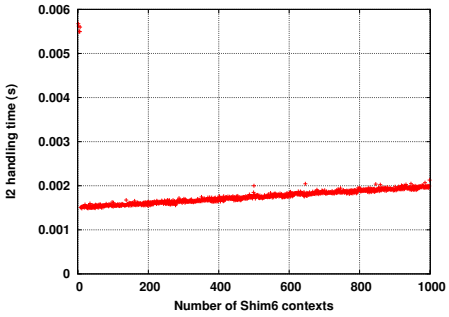

Figure 10: I2 generation time under high load

current address pair before actually triggering an exploration, our implementation begins exploration immediately upon expiration of the *Send* timer.

Probing the current locator pair, before commencing the exploration process, is useful when there is some doubt about the failure. For example, a host could receive a spoofed ICMP destination unreachable message, which should trigger a probe on the current pair, but not an exploration. This is so that the host can attempt to detect whether it was a genuine ICMP message or not. In our instance of Figure 9 and Figure 7 of [15], there is a timer expiry that indicates that no traffic has been seen during $Tsend$ seconds. Since REAP ensures (through keepalives) that the Tsend timer expires only when there is truly a network failure, we argue that this is a sufficient condition to immediately start the path exploration, with the benefit of lowering the ART.

3.4. Cost of state maintenance

`Shim6` requires state to be maintained at both the initiator (client) and responder (server). Obviously servers usually manage many connections simultaneously, this would then mean that a server could potentially have to manage many `shim6` contexts. In order to reduce the load on a server, it may be preferable to disable the LinShim6 heuristic. i.e. the context establishment trigger mechanism. That way, the server will never initiate a `shim6` negotiation, but only respond to context creation requests from clients. The first step of a `shim6` context initialisation would be the sending of an I1 message by the client. The server would reply with an R1 without creating state. Finally the client would send an I2, at which point context state would be created in the server. The I2 message holds the list of locators from the client, secured with a signature that the server is required to verify. If the I2 message is found to be valid, the server would then create a new context, and reply with an R2 message containing it's own signed locator set. As the locators of a server generally do not change all that often, our implementation computes the signatures in advance, in order to spare computing time during the context negotiation.

In our testbed (the same one as used to generate Figure 7), we evaluated the I2 processing time, the results of which can be seen in Figure 10. Our

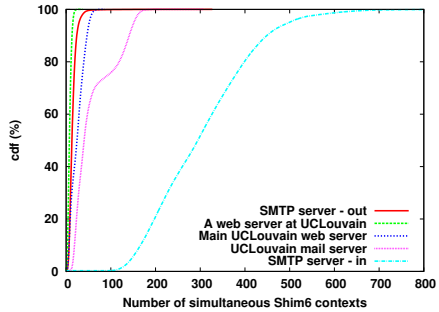

Figure 11: Case study of state maintenance on selected servers

tests consisted of the following. Every $50ms$, a client initiated a new context negotiation, each client used a different CGA source address, in order to force the creation of a new context in the server. The CGA was generated with a 1024-bit public key. 1000 such contexts have been created, and the I2 processing time measured. The x-axis shows the number of the clients (and hence context creation requests), sorted in chronological order (context 1000 is created $1000 * 50ms = 50s$ after the first one). The figure shows that even when a hosts has 1000 active contexts, the I2 processing time remains at around 2 milliseconds.

Figure 11 shows the result of a case study of `shim6` context management in our university. The netflow traces of several critical servers in our campus have been analysed (Full IPv4 netflow). Traffic was collected from the 1^{st} to the 7^{th} of August, 2008. In our analysis, we assume that each peer would trigger a `shim6` negotiation immediately after the first packet is exchanged and that servers are configured with a garbage collection time of 10 seconds (that is, if no traffic is seen during 10 seconds related to a particular context, then the server decides that it is no longer used and removes it. Peers having more than 10 seconds of idle time then need to renegotiate their context). By comparing with Figure 10, we can infer that the I2 processing time (cost of creating a new context) would not exceed $2ms$ for any of those servers. We also observe from Figure 11 that even in the worst case where each peer would trigger a `shim6` context establishment, the number of concurrent `shim6` contexts that need to be maintained is less than 800. Note that in case an administrator wants to reduce the observed number of simultaneous Shim6 contexts, he can lower the garbage collection time, in order to more aggressively drop Shim6 states. This tuning corresponds to moving state from the server to the network: the more aggressive a server is in dropping contexts, the more often clients will need to refeed the context data through network messages.

Having just explained that servers can avoid unnecessary context creation by simply disabling the `shim6` heuristic, and only create contexts upon request from the clients, one simple method that clients could use to reduce their `shim6` activity would be to introduce into the heuristic "hints" about whether the peer supports `shim6`. In particular we know that, currently, the majority of IPv6

addresses correspond to auto-generated MAC-based addresses. Those addresses can easily be detected thanks to their format, i.e. `ff:fe` in the middle of the interface identifier. If the peer uses such an address, most probably it has no support for `shim6`, because the use of multiple addresses by `shim6` requires their format to be either HBA or CGA. Heuristics can be implemented as a kernel module, and a user can define his/her own (or indeed modify the existing one), without having to modify the core implementation. So, for example, a heuristic could be defined to ignore auto-generated addresses, or to limit to some maximum the number of simultaneous `shim6` contexts.

4. Open Issues with Shim6 Multihoming

From a standardisation viewpoint, most of the work on `shim6` is finished with several RFC's now published [37, 4, 7]. Our implementation supports all the important features of `shim6`. However, there are still several outstanding issues to be solved before there will be a widespread deployment of `shim6`.

A primary issue is that `shim6` requires IPv6. As of this writing, the Internet still mainly uses IPv4, but given the expected exhaustion of the IPv4 address space, more and more networks are seriously considering IPv6 and have started deployments [25]. `Shim6` could be a very useful feature for multihomed networks. Initially, `shim6` could be used for important flows such as VPN, e-commerce or IP telephony servers where rapid recovery from link or interface failures is important. An important advantage of `shim6` over other multihoming solutions such as HIP or SCTP, is that `shim6` does not require any change to the applications running on hosts. Thus, applications can benefit from `shim6` without being aware of it. Simulations studies performed during the early phases of the `shim6` development have shown that host-based multihoming techniques such as `shim6` allow hosts to use many more paths to send their packets than traditional IPv4 BGP-based multihoming [16]. Furthermore, measurements have shown that by using these additional paths, it is possible to achieve much better performance, e.g. lower delays [16].

However, there are also some forces against a widespread deployment of `shim6`. At present, Internet Service Providers are very reluctant to consider it [44]. Their main concern with `shim6` is that it allows hosts to influence the path used to send and receive packets towards any multihomed destination. ISP operators have become accustomed to performing traffic engineering by refining their BGP configurations to take into account business policies. Consequently many consider that the deployment of `shim6` would limit their traffic engineering capabilities and make the network more difficult to manage [44]. We disagree with such statements. `Shim6` provides benefits to both ISPs and their clients. ISPs can benefit from a much more scalable interdomain routing system while clients can benefit from a much larger number of paths providing better performance and more redundancy.

In fact, peer-to-peer applications are also exploiting these alternate paths. Network operators could market `shim6` has an added value service to their customers willing to obtain improved performance or reliability. This service could

be combined with a path selection service provided by the ISP that allows its clients to easily determine the best path to reach a given destination. This type of service is already being developed to support peer-to-peer applications [1]. Recently, the IETF has chartered the ALTO working group to work on such a service [45] which would be very useful for `shim6` to integrate with.

A second issue concerning a widespread deployment of `shim6` is that many corporate networks insist on using Provider Independent IP addresses, even for IPv6, instead of Provider Aggregatable addresses [39]. This is because most operators consider that renumbering a network is too complex. Despite a lot of discussions on this topic [14], the IETF does not provide a solution to easily renumber a corporate network. Thanks to DHCP and IPv6's stateless auto-configuration, most hosts can easily change their address, but for servers and routers this remains difficult. For the specific case of `shim6`, a complete renumbering solution is not necessary. To easily support provider changes, a corporate network could use private addresses internally (e.g. for the routers and the management servers) and simply add the prefixes allocated by their providers to all their routers. Solutions to address this issue have been proposed in [29].

5. Related Work

The closest related work are the two prototype implementations developed by Park et al. [40] and Ahrenholz and Henderson [2]. These two implementations have also been developed on the Linux platform. They are mainly user-space implementations with `netfilter` hooks to capture the `shim6` packets and process them in user space daemons. In contrast, our implementation uses the `xfrm` framework and is implemented partially in the kernel with the non time-sensitive functions in user space. Another important difference is that our implementation completely supports the security mechanisms designed for `shim6`.

Other solutions have been developed to solve the multihoming problem. The SCTP transport protocol [46] was initially designed to support signaling servers in IP telephony environments. It has now been extended to support wider deployment scenarios and is supported by several operating systems. Another example is the Host Identity Protocol (HIP) [32]. HIP has been developed to evaluate the benefits and drawbacks of using a new cryptographical identifier namespace on top of IP. HIP has been extended to support multihoming and mobility [34] and there are several implementations of HIP available [27]. Compared to these solutions, the main benefit of `shim6` is that it does not require any change to the applications. This is very important for a new technique that needs to be incrementally deployed.

Several years ago, based on the recommendation from [31], the Routing Research Group of the Internet Research Task Force (IRTF) was rechartered to consider the evolution of the Internet architecture. Several of the techniques being evaluated within this working group [5, 19, 48] rely on separating the identifier and locator roles of the IP addresses, as in `shim6`. Although the details between these protocols and `shim6` vary, the experience gained by the

implementation and utilisation of `shim6` will be beneficial for the development of these new protocols.

For some, the issue of mobility is related to multihoming. Briefly, mobility consists of using several addresses in sequence, while multihoming consists of using them in parallel. Several issues are shared by both approaches as raised by Bagnulo et al. [8]. One of the authors (Barré), has evaluated a similar approach to [8] for managing mobility and multihoming in one shared solution. This resulted as well in a new implementation based on LinShim6, called MipShim6 [13].

Finally, recently an IETF working group has been created (MPTCP) to design a modified version of TCP, called Multipath TCP [22], that is able to failover from one path to another, and to spread one single transport flow across several paths. This is achieved by appropriately rethinking the congestion control algorithms. This is a promising approach to achieve better resource pooling in the Internet [49]. The architecture of the new MPTCP protocol [21] supports any technology that is able to expose multiple paths to the end-host, `shim6` being one of them. One of us (Barré) is also the author of the reference implementation for MPTCP [10]. Although the IETF requested that such a reference implementation be independent of Shim6, the multipath version of LinShim6 (see section 3.1) can be plugged into that implementation to provide a path management service for the MPTCP transport layer.

6. Conclusion

Multihoming is one of the problems that limits the scalability of the current Internet architecture, because it is currently obtained through injection of additional routes in the BGP system. In this paper, we have described and evaluated our implementation in the Linux kernel of the `shim6` host-based multihoming technique developed within the IETF. We have first explained the basics of this technique including the defined security mechanisms. Then, we detailed the architecture of our LinShim6 implementation along with the motivations for our main design choices. We have then evaluated the performance impact of the main security mechanisms of `shim6`. Finally, we have discussed the issues that remain open for a widespread deployment of `shim6`, while recognising that LinShim6 is currently the only complete and publicly available `shim6` implementation. The modularity of LinShim6 has been shown already by its applications to mobility and multipath. This efficient, modular kernel architecture can be reused in the design of future protocols based on locator/id separation.

Acknowledgements

J. Ronan is funded by the European Commission via the 7th Framework Programme Integrated Project EFIPSANS (grant no. 215549). S. Barré has been partly supported by a grant from FRIA and by the European Commission via the 7th Framework Programme Integrated Project TRILOGY (grant no. 216372).

[1] V. Aggarwal, A. Feldmann, and C. Scheideler. Can isps and p2p systems co-operate for improved performance? *ACM SIGCOMM Computer Communications Review (CCR)*, 37(3):29–40, 2008.

[2] J. Ahrenholz and T. Henderson. shim6 implementation. `http://www.openhip.org/docs/shim6.pdf`, 2008.

[3] J. Arkko, J. Kempf, B. Zill, and P. Nikander. SEcure Neighbor Discovery (SEND). RFC 3971, Mar. 2005.

[4] J. Arkko and I. van Beijnum. Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming. RFC 5534, June 2009.

[5] R. Atkinson. ILNP Concept of Operations. Internet draft, draft-rja-ilnp-intro-06.txt, work in progress, August 2010.

[6] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972, Mar. 2005.

[7] M. Bagnulo. Hash-Based Addresses (HBA). RFC 5535, June 2009.

[8] M. Bagnulo, A. Garcia-Martinez and A. Azcorra. IPv6 multihoming support in the mobile internet *Wireless Communications, IEEE 14(5), volume 14, IEEE, 92–98*, 2007.

[9] F. Baker and P. Savola. Ingress Filtering for Multihomed Networks. RFC 3704, Mar. 2004.

[10] S. Barré. Linux multipath tcp implementation. available from http://inl.info.ucl.ac.be/mptcp, Dec. 2010.

[11] S. Barré and O. Bonaventure. Improved path exploration in shim6-based multihoming. In *SIGCOMM 2007 Workshop "IPv6 and the Future of the Internet"*, 2007.

[12] S. Barré. Linshim6 - implementation of the shim6 protocol. Technical report, Université catholique de Louvain, Feb 2008. Available from http://inl.info.ucl.ac.be/linshim6.

[13] S. Barré, A. Dhraief, N. Montavont and O. Bonaventure. MipShim6: une approche combinée pour la mobilité et la multi-domiciliation In *Actes du 14ème Colloque Francophone sur l'Ingénierie des Protocoles (CFIP 09)*, October 2009.

[14] B. Carpenter, R. Atkinson, and H. Flinck. Renumbering Still Needs Work. RFC5887, May 2010.

[15] A. de la Oliva, M. Bagnulo, A. Garcia-Martinez, and I. Soto. Performance Analysis of the REAchability Protocol for IPv6 Multihoming. In *Conference on Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN 2007)*, Sept 2007.

[16] C. de Launois. *Unleashing traffic engineering for IPv6 multihomed sites*. PhD thesis, Université catholique de Louvain, October 2005.

[17] C. de Launois and M. Bagnulo. The paths towards IPv6 multihoming. *IEEE Communications Surveys and Tutorials*, 8(2), 2006.

[18] R. Draves. Default Address Selection for Internet Protocol version 6 (IPv6). RFC 3484, Feb. 2003.

[19] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID Separation Protocol (LISP). Internet draft, draft-ietf-lisp-09.txt, work in progress, Oct. 2010.

[20] N. Feamster, J. Borkenhagen, and J. Rexford. Guidelines for interdomain traffic engineering. *SIGCOMM Comput. Commun. Rev.*, 33(5):19–30, 2003.

[21] A. Ford, C. Raiciu, S. Barré, J. Iyengar, and B. Ford. Architectural Guidelines for Multipath TCP Development. Internet draft, draft-ietf-mptcp-architecture-03.txt, work in progress, Dec. 2010.

[22] A. Ford, C. Raiciu, and M. Handley. TCP Extensions for Multipath Operation with Multiple Addresses. Internet draft, draft-ietf-mptcp-multiaddressed-02.txt, work in progress, October 2010.

[23] G. Huston. Architectural Approaches to Multi-homing for IPv6. RFC 4177, Sept. 2005.

[24] G. Huston. Predicting the end of the world. ISP Column, `http://www.potaroo.net/ispcol/2009-05/ipv4model.html`, May 2009.

[25] G. Huston. Addressing 2009. ISP Column, `http://www.potaroo.net/ispcol/2010-01/addresses-2009.html`, January 2010.

[26] J. Kempf, J. Wood, Z. Ramzan, and C. Gentry. IP Address Authorization for Secure Address Proxying using Multi-key CGAs and Ring Signatures. *IEICE TRANSACTIONS on Communications*, E87-B(3):429–436, March 2004.

[27] A. Khurri, E. Vorobyeva, and A. Gurtov. Performance of host identity protocol on lightweight hardware. In *MobiArch '07: Proceedings of 2nd ACM/IEEE international workshop on Mobility in the evolving internet architecture*, pages 1–8, New York, NY, USA, 2007. ACM.

[28] M. Komu, M. Bagnulo, K. Slavov, and S. Sugimoto. Socket Application Program Interface (API) for Multihoming Shim. Internet draft, draft-ietf-shim6-multihome-shim-api-15.txt, work in progress, Oct. 2010.

[29] D. Leroy and O. Bonaventure. Preparing network configurations for ipv6 renumbering. *International Journal of Network Management, Wiley InterScience*, 19(5):415–426, Sept-Oct 2009. Online ISSN: 1099-1190, Print ISSN: 1055-7148, DOI: 10.1002/nem.717, Copyright 2009 John Wiley & Sons, Ltd.

[30] J. P. Martinez. Provider Independent (PI) IPv6 Assignments for End User Organisations. Ripe Policy Proposal 2006-01, February 2009.

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

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

[33] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host Identity Protocol. RFC 5201, Apr. 2008.

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

[35] A. Matsumoto and T. Fujisaki and R. Hiromi and K. Kanayama. Problem Statement for Default Address Selection in Multi-Prefix Environments: Operational Issues of RFC 3484 Default Rules". RFC 5220, July 2008.

[36] A. Matsumoto and T. Fujisaki and R. Hiromi and K. Kanayama. Requirements for Address Selection Mechanisms. RFC 5221, July 2008.

[37] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533, June 2009.

[38] E. Nordmark and T. Li. Threats Relating to IPv6 Multihoming Solutions. RFC 4218, Oct. 2005.

[39] J. Palet. Provider independent (PI) IPv6 assignments for end user organisations. RIPE Policy Proposal 2006-01, `http://www.ripe.net/ripe/policies/proposals/2006-01.html`.

[40] K. Park, H. Cho, Y. Choi, T. Kwon, T. You, and S. Lee. The implementation of layer-three site multihoming protocol (l3shim). *Advanced Communication Technology, The 9th International Conference on*, 1:234–237, Feb. 2007.

[41] B. Quoitin, S. Uhlig, C. Pelsser, L. Swinnen, and O. Bonaventure. Interdomain traffic engineering with BGP. *IEEE Communications Magazine*, May 2003.

[42] J. Ronan, S. Balasubramaniam, A. K. Kiani, and W. Yao. On the use of SHIM6 for mobility support in IMS networks. In M. P. de Leon, editor, *TRIDENTCOM*, page 40. ICST, 2008.

[43] J. Ronan and J. McLaughlin. An empirical evaluation of a Shim6 implementation. *2nd International ICST Conference on Mobile Networks and Management*, Sept. 2010

[44] J. Schiller. Inter-as traffic engineering case studies as requirements for ipv6 multihoming solutions. NANOG 34 presentation, may 2005.

[45] J. Seedorf and E. Burger. Application-Layer Traffic Optimization (ALTO) Problem Statement. RFC 5693, Oct. 2009.

[46] R. Stewart. Stream Control Transmission Protocol. RFC 4960, Sept. 2007.

[47] The Trilogy Project. Design space analysis for reachability:approaches to routing, locator-identifier resolution and multiple path exposure. *Project deliverable 4*, http://trilogy-project.org/publications/deliverables.html, Dec. 2008.

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

[49] D. Wischik, M. Handley, and M. B. Braun. The resource pooling principle. *SIGCOMM Comput. Commun. Rev.*, 38(5):47–52, 2008.

[50] H. Yoshifuji, K. Miyazawa, M. Nakamura, Y. Sekiya, H. Esaki, and J. Murai. Linux IPv6 Stack Implementation Based on Serialized Data State Processing. *IEICE TRANSACTIONS on Communications*, E87-B(3):429–436, March 2004.