Improved Path Exploration in shim6-based Multihoming

Sébastien Barré*, Olivier Bonaventure
Department of Computer Science
Université Catholique de Louvain (UCL), Belgium
{firstname.lastname}@uclouvain.be

ABSTRACT

The shim6 host-based solution to IPv6 multihoming was designed within the IETF to provide a solution to the multihoming problem by using several IPv6 addresses per host. In this paper, we briefly describe our Linux implementation of shim6. One of the novel mechanisms introduced by shim6 is the REAchability Protocol (REAP) that allows multihomed hosts to switch to an alternate path when a failure occurs. We evaluate the performance of this protocol in a lab environment and show that it performs well in the case of unidirectional or bidirectional failures. We also show that the recovery time can be reduced by allowing REAP to send multiple probes upon failure detection.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Protocols*; C.4 [**Computer Systems Organization**]: Performance of Systems—*Fault tolerance*

General Terms

Measurement, Performance

Keywords

IPv6, Multihoming, REAP

1. INTRODUCTION

The Internet has been growing since its creation, both in terms of supported applications and connected networks. As of June 2007, the Internet is composed of more than $25,000$ different Autonomous Systems (AS) and core Internet routers maintain routes towards more than $200,000$ different IPv4 prefixes [5]. During the last years, most of the growth in the routing table size of core routers has been caused by multihoming and Traffic Engineering [12]. The practice of multihoming consists of being connected to two or more ASes. Multihoming may be done for cost, performance or redundancy reasons.

Today, the IPv6 routing tables are much smaller than their IPv4 equivalent, but they will continue to grow as IPv6 becomes more ubiquitous. Providing a better support for multihomed sites is thus a key factor for the scalability of IPv6 routing. The IETF evaluated many solutions to the multihoming problem [7] and decided to focus on a host-based approach called shim6 [14]. Shim6 changes significantly the way IPv6 hosts behave. The main modification is that a multihomed shim6 host will use several IPv6 addresses, typically one from each of its providers. This brings new opportunities. A first example is the ability for a multihomed host to select the path on which packets will be received for a given flow by selecting the local address that it uses for this flow. A second example is the selection of a best path. When two multihomed hosts exchange packets, they could select the pair of addresses that gives the lowest delay [8] as all pairs will use different paths with different characteristics. A third example is the negotiation of another address pair to find a new operational path in case of link failure.

The development of shim6 has been inspired by the Host Identity Protocol (HIP) [13]. HIP relies on cryptography to generate non routable host identifiers, thus providing authentication capabilities. To the opposite, shim6 was designed to use normal IPv6 addresses as identifiers, while using cryptography to link together all the addresses associated to a multihomed host [4]. Another difference is that shim6 includes a new protocol called REAP [1] to detect failures on the end-to-end path. Upon failure detection, REAP tries to find another pair of IPv6 addresses that still allow the two hosts to communicate, by exploring the alternative paths.

Until now, most of the work on the shim6 approach has focused on discussions related to the development of the protocols themselves. As a result, a large number of Internet drafts have been produced [1, 14, 2]. But the ultimate test will come with the deployment of shim6 in the IPv6 Internet. Our first contribution is the shim6 implementation in the Linux kernel. Our second contribution is a detailed evaluation of the behaviour of the REAP path exploration in a lab testbed.

This paper is organised as follows. We first briefly describe shim6 in section 2. Then we present the associated failure recovery protocol, REAP. In section 5 we evaluate the performance of the protocol, in particular the path exploration, using our Linux implementation. Finally several issues are exposed, as future work on shim6.

2. THE SHIM6 MECHANISM

The shim6 approach to multihoming supposes that each host in a network owns several global IPv6 addresses, one per provider,

*Supported by a grant from FRIA (Fonds pour la Formation à la Recherche dans l'Industrie et dans l'Agriculture, rue d'Egmont 5 - 1000 Bruxelles, Belgium). This work was partially supported by the European-funded 034819 OneLab project.

IPv6'07, August 31, 2007, Kyoto, Japan.

so that selecting an address as a source for a packet implies the selection of a provider. Similarly, selecting a peer's address as a destination implies the selection of the peer's provider. Of course other address allocation schemes are possible, so that more generally we can say that the address pair of a packet determines the path used by that packet. In order to get the benefit from this new ability to select a path inside an end host, `shim6` splits up the two semantics of an IPv6 address. An IPv6 address can be seen either as a locator (for IP routing) or as an identifier (known as ULID, Upper Layer IDentifier) for Upper Layer Protocols. This is a subtle but important change to the TCP/IP architecture that requires a change in end-systems implementations. Inside the network stack, `shim6` takes the form of a new sublayer, inside layer 3, whose role is to map ULIDs on locators. This is useful to ensure transport layer survivability across path changes, that is, across changes of locator pairs. The typical walkthrough of a TCP connection over `shim6` is as follows :

- Host A establishes a TCP connection with host B. This is done by sending a SYN segment, without `shim6` doing anything. Because there is no `shim6` context at that moment, the IPv6 addresses play both roles of ULIDs and locators.
- After some time, host A decides to take advantage of `shim6` capabilities, and starts a `shim6` initialisation exchange [14]. After that, each host has a `shim6` context state associated with the flow, which contains a list of local and distant locators.
- The conversation continues, still using identical ULIDs and locators. The context exists but isn't really used (except for accounting purposes).
- For any reason (failure detected, decision taken from application layer), new locators can be used. These are chosen from the context state previously created. After that, ULIDs remain the same but the locators are changed, so that a mapping becomes necessary inside the `shim` layer.

Figure 1 illustrates the case of a failure detection and recovery by host A, in an exchange with host B. The communication is initiated using path c1. After some time, a failure occurs and is detected by host A. This results in a path switch from c1 to c2. Because the ULIDs are chosen as the initial locators, the ULID pair remains `ISP1.A,ISPX.B` throughout the entire exchange. But after the failure, locators become `ISP2.A,ISPX.B`

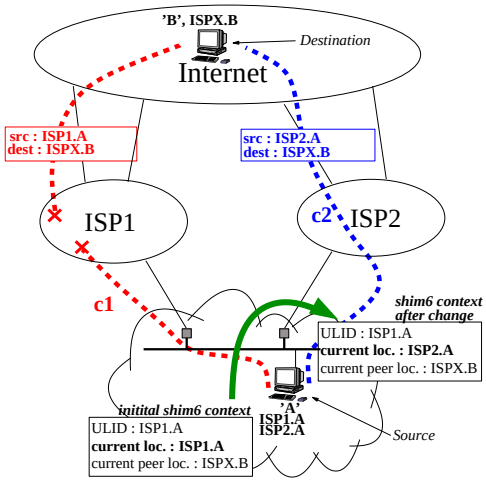

Figure 1: Switch to an alternate path using `shim6`

3. PATH EXPLORATION : REAP

While the `shim6` protocol aims at exchanging locator lists and mapping locators to identifiers, REAP [1] is the companion protocol responsible for failure detection and recovery. On a genuine IPv6 host, failure detection is handled by ICMP and TCP. If the broken path is never restablished, this results in the connection eventually being lost. With `shim6` an alternative operational path may be found, so that TCP (for example) can continue as if the previous path had been restablished. Moreover, failure detection and recovery is addressed in the `shim6` layer, without requiring any change in the upper layers. Failure detection is possible because REAP sends keepalives when no data is produced by upper layers. Because a host should either receive keepalives or data packets, a failure can be detected on the basis of a timer expiry. Failure detection triggers a path exploration procedure. Probes are sent with exponential backoff along various paths, each path being determined by the locator pair used. The probe exchange scheme is designed to allow detection and use of unidirectional paths.

REAP relies on a state machine [1] that can be in one of three states : `operational`, `exploring` or `inbound ok`. If the communication is not experiencing any problem, the state is `operational`. This means that end hosts receive either data packets or keepalives from each other. Keepalives are sent if a host has not sent any packet during some time defined as *keepalive interval* (default is 3 seconds). If data traffic stops for a while, keepalives are sent every *keepalive interval*, for the *keepalive timeout* duration (default is 10 seconds). Then no more keepalives are sent until data packets are sent again or the context is destroyed.

The second state defined in [1] is `exploring`. A context reaches that state if a failure has been detected due to the expiration of the *send timer* (default is 10 seconds). This timer is started when sending a data packet and only if it was not already running. It is stopped upon reception of any packet from the peer. Because REAP ensures that the peer will reply with either data or keepalives, not receiving anything from him means that a failure occurred. Note that there are additional ways to detect failures such as indications from upper layers, lower layers or ICMP error messages [1]. Some of these indications may be faster than the timer expiry, but they are not always available.

The third state is named `inbound ok`. A host is in this state if it is receiving packets (either data, keepalives or probes) from its peer, but there is indication that the peer doesn't receive anything. A host may reach the `inbound ok` state from `operational` if it receives an exploring probe from its peer, or from `exploring` if it receives anything from its peer.

A probe contains the state of the sender, a nonce used as identifier and a number of *reports*. A *report* is defined as a summary of a sent or received probe. According to [1] a report contains the source and destination addresses, the nonce and an option field (currently unused). As we will see in the following example, reports of received probes are necessary to learn a new operational path.

Figure 2 shows the case of the failure of the path from locator `B1` to locator `A1`. `{A1,A2,A3}` and `{B1,B2,B3}` are the locators assigned to A and B respectively. The first few arrows show the exchange of data packets using locators `A1` and `B1`, when the first answer from B is lost. This packet lost triggers a *send timer* expiry inside host `A`. The consequence is that A switches to the `exploring` state and starts sending probes. The first one, a (with locators `(A1,B1)`), goes through, and B learns that its packets no longer reach host A, with the effect of B going to the `inbound ok` state. After that, both hosts are sending probes along every known path. In our example, all probes from B preceding r are

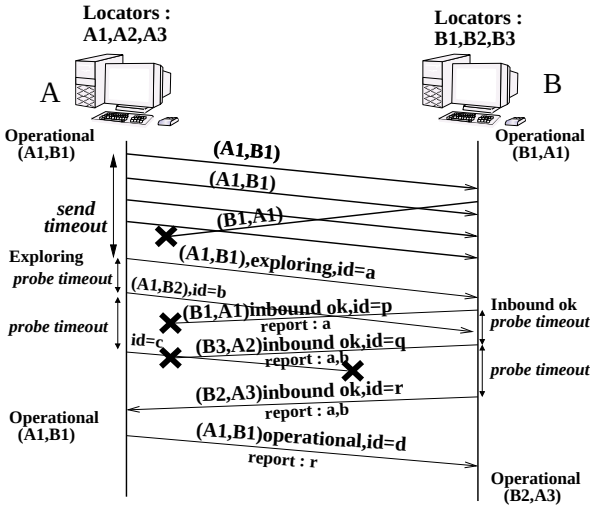

Figure 2: Example of failure detection and recovery

lost. Upon reception, host A reads the reports and learns that probes a and b were successful, that is, locator pairs `A1,B1` and `A1,B2` are eligible as current locators. The first one is chosen, and a final probe is sent to host B with this new locator pair to announce the switch to the `operational` state. Because this final probe is sent using a working locator pair, it reaches host B. B learns that its only successful probe has been the one with *id=r*. This means that the only working locator pair is `B2,A3`. Both hosts update their `shim6` context, for address mapping inside the `shim6` sublayer, and the conversation continues, without upper layers seeing anything else than some delay.

4. IMPLEMENTATION : LINSHIM6

LinShim6 first appeared as a result of a master thesis work, which described the initial design choices and implications on the Linux Networking stack of the new shim6 layer [15]. While this initial Linux implementation was fully integrated to the kernel, in the current version (0.4.3) we have placed the major parts of the REAchability Protocol in user space so as to reduce as much as possible the kernel size, as well as maintenance costs, while still preserving efficiency regarding packet translation. Kernel and user space communicate through the Netlink mechanism. To avoid having many messages exchanged between kernel and user space, we chose to put REAP related timers inside the kernel, because they must be modified for each incoming or outgoing packet. This way, there is no need for kernel to user space exchanges under normal conditions. The kernel is able to detect failures and notify the REAP daemon. The REAP daemon then performs path exploration and provides the kernel with a new operational locator pair.

Currently a shim6 context is created for each new network flow initiated by the local host. A network flow is identified by a pair of IPv6 addresses. Because ports are not part of the context key, several transport flows may be aggregated in one network flow, thus reducing the total amount of memory used by `shim6`.

LinShim6[1] fully conforms to the latest REAP specification [1], and supports the core components of the `shim6` protocol, that is, context negotiation and packet translation [14]. LinShim6 has been instrumented to log exploration times.

[1]Our implementation is available from `http://inl.info.ucl.ac.be/LinShim6`

5. PERFORMANCE EVALUATION

To our knowledge, no evaluation of the REAP path exploration has been published yet. An evaluation of the effect of the *send timer* has been shown in [6], on the basis of a simulation. In this paper, we propose an evaluation on the basis of our Linux implementation.

In this section, we first present the testbed used for all experiments. Then we provide a short validation of our implementation, which also demonstrates the interest of `shim6` for transport layer survivability. The last two subsections present a detailed evaluation of the REAP path exploration.

5.1 The testbed

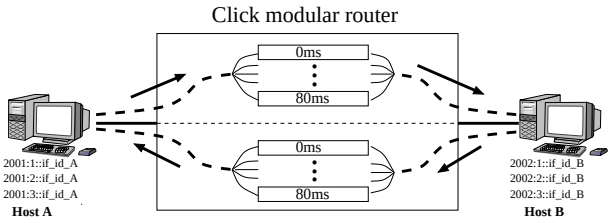

Figure 3: Testbed for REAP measurements

Our testbed is composed of three Linux computers. Two of them support `shim6`, and the third acts as a Click router [11], used to emulate the different paths. The router and one of the end hosts are Pentium II, 300Mhz with 128MB of RAM and 100baseTx-FD Ethernet cards. The other end host is a Pentium Pro 200Mhz with 64MB of RAM and 100baseTx-FD Ethernet cards. Both end hosts run the Linux kernel 2.6.17.11 patched with `shim6/REAP` release 0.4.3. The Click router runs Linux kernel 2.6.16.13 patched with Click release 1.5.0. In order to make measurements faster, the *send timer* has been set to 3 seconds. The setup is shown in figure 3. The router runs the Router Advertisement Daemon [2], that distributes the three prefixes of host A and host B. One Click queue is defined for each possible pair of addresses. Since each `shim6` computer receives three prefixes, 9 queues are defined for each direction. Each queue may be configured to be delayed or stopped. Thus we have a total of 18 configurable queues inside the router. This gives the flexibility of simulating unidirectional paths in the Internet, with a configurable delay for each one separately. We may also create a failure in one direction while keeping the other direction of communication operational.

5.2 Validation

To show the benefits of `shim6` for a TCP application, we present in figure 4 the effect of a path failure on the throughput of an iperf TCP session. The path is broken approximately 20 seconds after starting the iperf client. The different curves are obtained by artificially adding delay to the paths inside the Click router. This figure shows one of the most important benefits of `shim6`, that is, transport layer survivability across failures, without any change to TCP. Note that normal TCP/IP is already able to survive if the broken path comes back to life quickly enough. The difference here is that TCP behaves *as if* the path came back to life, while in fact another path has been selected thanks to the REAP path exploration. After the recovery, ULIDs are kept constant, while locators are changed, as a result of the path change.

The throughput drop of Fig. 4 represents the full recovery time, including the expiration of the *send timer* (3 seconds). Now it

[2]`http://www.litech.org/radvd/`

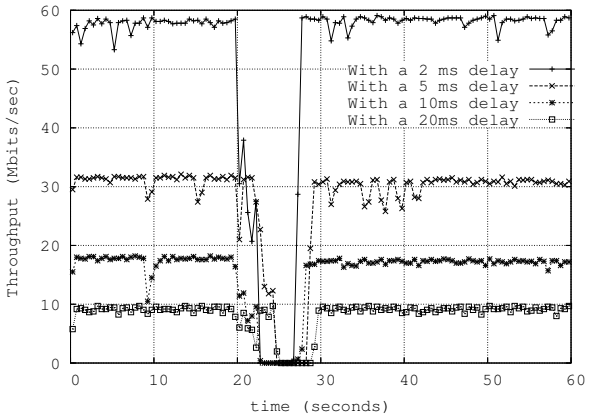

Figure 4: Evolution of throughput for an iperf TCP session

would be desirable to make this recovery time as short as possible, that is, the throughput drop of Fig. 4 should be as narrow as possible.

5.3 Exploration time

We define the *exploration time* as the duration between leaving and coming back to the REAP `operational` state. This is different from the *detection time*, defined as the interval between the occurrence of a failure, and failure detection by REAP. Finally, the *recovery time* is the sum of the detection and exploration times. The detection time is mainly influenced by the value of the *send timer*. But since this timer is started every time a data packet is sent (if it is not already running), the detection time is also influenced by the frequency of outgoing data packets. For example, if one starts an `ssh` session, then stops activity during some time, keepalives will be sent by `shim6` until *keepalive timeout*, but after that no keepalives will be exchanged anymore until `ssh` becomes active again. In that case the failure will be detected up to *send timeout* seconds after the first data packet has been sent. This is of course a worst case. There is also a best case, which may occur if we can avoid to rely on timers for failure detection.

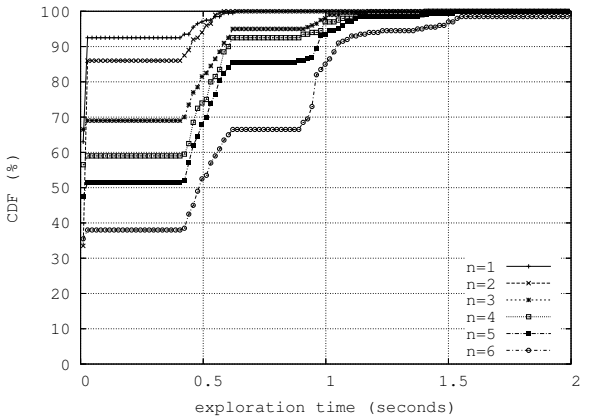

Figure 5: CDF of exploration times when n paths are broken

The exploration time depends exponentially on the number of probes sent by each of the peers before finding a working path for each direction of communication. It is important here to specify how often a host may send probes. According to [1], four initial probes are sent with an interval of 500 ms. Then exponential backoff is started, and the interval is doubled each time a probe is sent. When the time interval between probes reaches 60 seconds, exponential backoff is stopped and one probe is sent every 60 seconds.

Our implementation randomly adds or removes, for each probe, 0 to 20 % of the above described intervals to avoid self-synchronization [9] (the maximum value of 20 % has been fixed arbitrarily). It is also important to note that our implementation selects address pairs by cycling randomly over all possible paths. One part of the future work is to evaluate other selection mechanisms.

Figure 5 shows the cumulative distribution of the exploration times for the testbed described in section 5.1. Because failure detection requires the existence of a data stream, a UDP client and server have been placed on each host, to establish a UDP bidirectional flow of one packet every second.

Figure 5 shows the measured exploration times for different values of the number n of broken paths. Broken paths are emulated by tearing down queues inside the Click router. Note that one queue inside the Click router corresponds to a *unidirectional* path from A to B or from B to A, as shown in Fig. 3. The highest CDF ($n = 1$) is made from 200 emulated unidirectional link failures (100 for each direction of the communication). All the other distributions represent bidirectional failures, that is, we tear down at least the currently used queues for each simulated failure. Additional queues (thus additional unidirectional paths) are selected by running through the possible queue combinations, without taking twice the same combination when possible. For each value of n, 200 measurements have been conducted. No artificial delay has been introduced inside queues for that experiment. Note that n may have odd values. This means that, although we always break each direction of the current communication (except for $n = 1$, additional broken paths are chosen on a unidirectional basis, so that the existence of a path for a given address pair does not imply the existence of a path for the reverse address pair.

The highest curve in figure 5 is obtained for $n = 1$. In that case only one direction is broken. The second highest curve is obtained for $n = 2$. We see that if only the active queues are disabled, keeping 16 queues enabled, the first probe sent is successfull in 86% of the trials. As in the case of $n = 1$, the second probe is always successful. We can conclude this because the initial probes are sent with a 500 ms interval, and 100% of the trials take less than 600 ms (that is, 500 ms plus the 20% jitter).

As we shutdown more and more queues, we decrease the probability for a randomly chosen path to be successful. This gives lower curves as n increases, because each exploration has a higher probability of demanding more probes, thus more time. This is observed in figure 5, where the curves are lower and lower with increasing values of n.

Because the testbed has 18 paths (9 in each direction), we have measured the worst case of breaking 16 paths, letting only one available path for each direction. Since more then four probes must be sent in most of the cases, exponential backoff is applied. 50% of the explorations lasted less than 7 seconds, 80% less than 17 seconds and 95% lasted less than 31 seconds. This is an academic case however. In the real world, it is indeed very unlikely to have only two operational paths among 18 available.

5.4 Paths with different delays

After having studied the impact of the number of broken paths on the exploration time, we now evaluate whether the REAP protocol is able to choose the path with lowest delay. For that experiment, we have configured each queue with an increasing delay in Click. We assigned a delay starting at 0 ms with a 10 ms increment. The

last queue (ninth) has a delay of 80 ms. The 9 queues from A to B and from B to A have a symmetric configuration. 500 failures have been simulated. For each failure, we save the path selected after recovery. Figure 6 shows a histogram that gives the frequency of selection for each kind of path, sorted by Round Trip Times. For example, if REAP has selected a path with 10 ms for one direction, and another one with 20 ms for the other direction, we increment by one the 30ms histogram bar. On the other hand figure 7 gathers information about the use of each one-way path. For the above example of a selection of 10 and 20ms, we would increment by one both the 10 and 20 ms histogram bars.

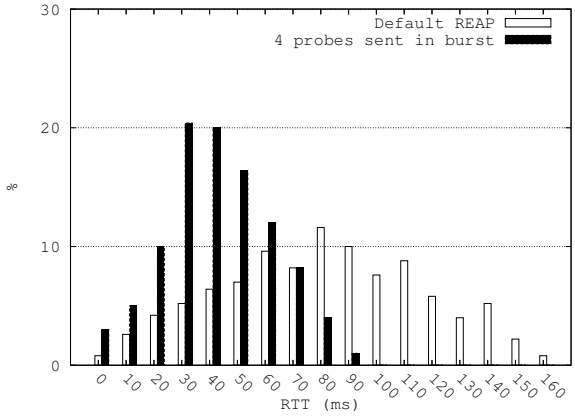

Figure 6: Proportions of use for pairs of paths with different delays

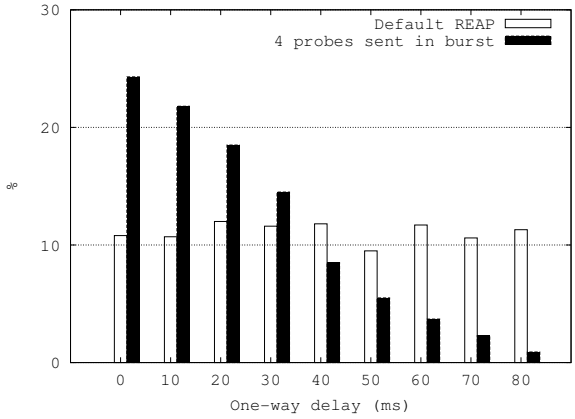

Figure 7: Proportions of use for paths with different delays

The ideal case would be to have 50 % of use for each of the two best paths. This is because in the case of the 0 ms path being broken (if it is the current one), an ideal REAP would select the 10 ms path (that is, the pair of paths with 20ms as RTT). The next trial would lead to break the 10 ms path and selecting the 0 ms path. Thus, in a perfect world, this experiment would consist of continously jumping between the best path and the second best path.

We can observe in figure 7 that REAP, with the default parameters [1], gives an almost uniform distribution, due to the random selection of paths. We have also tried to slightly modify REAP, replacing the 500ms initial inter-probe delay with a 0 ms delay. That is, sending four probes in burst. The solid bars show the same measurements performed with the modified REAP. While the two best paths have a total selection proportion of 21.5% with legacy REAP, we obtain a proportion of 46.1% when sending 4 probes in burst. This is due to the fact that if probes are sent in burst, the first received answer is taken as the new current path. But that answer is the one corresponding to the path with the lowest round trip time among the four tried. If we increase the size of the burst, we will get a higher proportion of selection for the best paths, at the expense of more control packets being sent.

The comparison between figures 6 and 7 clearly shows the undirectional nature of path exploration : Figure 7 says us that for default REAP, each unidirectional path has an equal probability to be chosen, regardless of the path used in the other direction. Figure 6 shows that it is frequent to have different address pairs for the two directions of communication. Since we configured the same delay for a given address pair and its reverse combination, bidirectional path selection would have implied an absence of bars for RTTS such as 30ms, since we have no 15ms queue in the lab. Actually the 30ms RTT is the highest bar for the modified REAP.

Now, one could wonder why not to send actually more probes in burst, so as to obtain a better chance of getting a good path in terms of delay. We need to find a tradeoff in order to avoid a signalling storm in the case for example of the main access link of a campus network becoming broken. Solutions to mitigate this kind of problem are part of the future work as explained in the next section.

6. FURTHER WORK ON SHIM6

Based on our experience with the first `shim6` implementation, we have identified several problems that need to be solved to efficiently deploy `shim6` in the IPv6 Internet.

First, the `shim6` specification states that hosts may either start immediately a locator exchange or use some heuristic to defer context establishment. Implementations will need to include heuristics to decide when to establish a `shim6` context. For example, if the hosts prefer a path with a low delay for a given flow, a context may be established as soon as the flow is started, so that all locators are known and a lower delay path can be selected. It could also be useful to prevent `shim6` from being enabled in some cases. This may be decided by the application layer. For example, it would not be useful to pay the cost of a context establishment for a DNS query. Our further work will be to study, with help of real traffic traces, which kinds of flows would benefit from `shim6` support.

A second issue is the selection of paths. There is a path choice to make at the beginning of an exchange with a new peer, normally not under `shim6` control, as previously explained. There is also a similar decision upon failure. Additionally, a `shim6` host may want to change the currently used path if a better one becomes available. A possible approach for this path selection would be to use an Internet coordinate system such as the one proposed in [8]. Such a system could provide an ordering of locator pairs, so that, while exploring, probes can be sent first on low delay paths. However, proposed Internet coordinate systems are still mainly a research proposal. Security issues such as those discussed in [10] will need to be solved before all `shim6` hosts can depend on a stable Internet coordinate system. Another possible approach to find low delay paths is to use the REAP protocol itself. The previous section showed that REAP can prefer low delay paths if probes are sent in burst. We have observed that sending several initial probes in burst helps in discovering a new path faster, and also increases the probability of choosing a low delay path. But there is a trade off between the benefit of having fast discovery of low delay paths and the packet overhead of REAP. Delays may be learned also from the round trip times of probes. The problem then is that we need to explore all paths before knowing the best one. A solution could be

to send probes from time to time also when we are not exploring. This would allow to maintain a dynamic ordering of address pairs, adapted each time a new probe is received. The accuracy of such an ordering would be directly dependent on the frequency of probing. At the cost of (tunable) signalling load, we would get the benefit of switching not only upon failure detection, but also upon better path detection. Moreover, address pairs would be already ordered when a failure occurs, so that the heavy exploration could be made shorter.

A third aspect that should be studied is the behaviour of `shim6` in a campus or enterprise network. Most of the work on `shim6` has been based on the hypothesis of independent hosts. This hypothesis is valid for end-user hosts having two different active physical interfaces (e.g. an Ethernet and a WiFi interface), but in enterprise networks this is less clear. When a link between an enterprise router and a provider fails, all paths using the provider prefix will fail at the same time. If thousands of `shim6` flows are using this link at that time, they will all start a path exploration by using the REAP protocol. This is probably not the most efficient mechanism. In that case, it should be possible to coordinate the behaviour of all hosts. In the above example, it should be possible for the access router to inform all enterprise hosts about the failure of its link. This concern could be addressed by the use of a middle box that would take the responsibility of finding a best path, detecting and recovering from failures. Such a middle box could be a `shim6` proxy as proposed in [3], where the proxy manages the whole `shim6` protocol on behalf of end hosts.

7. CONCLUSION

The `shim6` and REAP protocols have been developped within the IETF to solve the multihoming problem in a scalable host-based manner. In this paper, we have briefly described our implementation of `shim6` and REAP on Linux. To our knowledge, this is the first publicly available implementation of `shim6`.

We have evaluated its behaviour in a lab environment. We have first shown that `shim6` allows TCP connections to survive partial path failures with a limited impact. We have then studied the parameters of the REAP protocol that affect the path recovery performance. When a small number of paths are broken, exploration always lasts less than 2 seconds and usually less than 1 second. When a large number of paths are broken, the duration of the exploration increases due to the exponential backoff mechanism used by REAP.

Our second evaluation has shown that, without information about the quality of the paths, every working path has an equal probability of being selected. Paths with lower delays may be chosen by simply allowing REAP to send several probes in burst. Our further work will be to perform measurements with real failures in the IPv6 Internet and develop techniques to improve the performance of shim6 in enterprise networks.

Acknowledgements

We would like to thank Bruno Quoitin and the anonymous reviewers for the careful comments provided on this paper.

8. REFERENCES

[1] J. Arkko and I. van Beijnum. Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming. Internet draft, IETF, December 2006. draft-ietf-shim6-failure-detection-07.txt, work in progress.

[2] M. Bagnulo. Hashed Based Adresses (HBA). Internet draft, IETF, October 2006. draft-ietf-shim6-hba-02.txt, work in progress.

[3] M. Bagnulo. Proxy Shim6 (P-Shim6). Internet draft, draft-bagnulo-pshim6-01.txt, work in progress, January 2007.

[4] M. Bagnulo, A. García-Martínez, and A. Azcorra. Efficient security for IPv6 multihoming. *SIGCOMM Comput. Commun. Rev.*, 35(2):61–68, 2005.

[5] T. Bates, P. Smith, and G. Huston. CIDR Report. available from `http://www.cidr-report.org/`, June 2007.

[6] A. de la Oliva, B. Donnet, I. Soto, and T. Friedman. Multihoming Architecture Document. OneLab deliverable D4C.1, February 2007.

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

[8] C. de Launois, S. Uhlig, and O. Bonaventure. Scalable route selection for IPv6 multihomed sites. In *Proceedings of Networking 2005*, Waterloo, Ontario, Canada, May 2-6th 2005.

[9] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Trans. Netw.*, 2(2):122–136, 1994.

[10] M. Kaafar, L. Mathy, and T. Turletti. Real attacks on virtual networks: Vivaldi out of tune. In *SIGCOMM Workshop on Large Scale Attack Defence (LSAD)*, Pisa, September 2006.

[11] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[12] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. Internet Draft, IETF, April 2007. <draft-iab-raws-report-02.txt>, work in progress.

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

[14] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. Internet draft, draft-ietf-shim6-proto-08.txt, work in progress, May 2007.

[15] S. Barré. Développement d'extensions au Kernel Linux pour supporter le multihoming IPv6. Master's thesis, UCL, 2006. (written in French).