An empirical evaluation of a Shim6 implementation

John Ronan and John McLaughlin

Telecommunications Software & Systems Group,
Waterford Institute of Technology,
Cork Road,
Waterford,
Ireland
{jronan,jmclaughlin}@tssg.org

Abstract. Several solutions are proposed to enable scalable multihoming over IPv6. One of these proposals is Shim6, a host-based multihoming solution based on the modification of the Internet Protocol stack of the host. This modification adds a layer below the transport protocols but above the forwarding layer. As this approach makes the modifications to the network stack transparent, existing applications automatically benefit from Shim6 functionality.

In this paper we investigated aspects of the performance of the LinShim6 implementation from Université Catholique de Louvain. We also outline our modifications of the LinShim6 implementation to allow external software to control the locators used between hosts.

Key words: Shim6, multihoming, ECN

1 Introduction

For a number of years now, the IETF has been working on IPv6, a successor to IPv4. More recently, work has been done to address the scalability of the current internet architecture. The Internet Architecture Board (IAB) has identified several limitations of the current internet architecture[1]. These issues impact the scalability of inter-domain routing systems, which is reflected in the growth of Border Gateway Protocol (BGP)[2] routing tables, and also in the number of routes in the Default Free Zone (DFZ) Routing Information Base (RIB) processed by BGP routers. Several factors which influence the growth of BGP routing tables include, multihoming, traffic engineering, IP address allocation policy, and business events, such as large mergers and acquisitions. All of these factors can lead to an increased number of unique routing prefixes that cannot be aggregated within the DFZ RIB and hence cause routing table growth.

Several years ago, after examining the multihoming issue[3], the IETF chartered the Site Multihoming by IPv6 Intermediation (Shim6) working group to develop a host-based IPv6 multihoming solution, [4] presents a good overview of the requirements, constraints, and the process that led to the emergence of

Shim6 as a multihoming solution. The Shim6 specification documents are now published[5, 6, 7], and in this paper we report on our experiences with Université Catholique de Louvain's (UCL) publicly available Shim6 implementation LinShim6[8, 9] for the Linux kernel.

This paper is organised as follows: Section 2 provides a brief description of the capabilities of Shim6 for the benefit of those readers unfamiliar with the protocol. Section 3 describes both our overall goal and experiences in creating a Shim6 testbed. Section 4 presents the baseline performance measurements, with a brief description of the results. In closing, section 5 discusses the work presented here within the broader context of the EU FP7 EFIPSANS[1] research project.

2 Shim6 host-based IPv6 multihoming

The Shim6 protocol[6], has been designed to add multihoming capabilities to IPv6 end-hosts. Potentially, this allows for far more IPv6 enabled sites to protect their upstream connections, without having to go to the trouble of implementing BGP peering. This means that entities can retain control within their own site without incurring the overhead of deploying BGP.

Along with the Shim6 protocol, the IETF Site Multihoming by IPv6 Intermediation (Shim6) working group designed a failure detection and repair mechanism, called the REAchability protocol (REAP)[7] which allows hosts to detect and recover from failures.

Today, in the current (IPv4) Internet, a multihomed site is obliged to have a network connection with each of its upstream providers, and the site has to use IP addresses independent from those providers. These addresses come from what is called Provider Independent or PI address space (as opposed to Provider Aggregatable (PA) address space).

With Shim6 however, multihoming functionality is made available to the end host using Provider Aggregatable addresses — removing the need to involve BGP or any other protocol. At present, the default IPv6 address selection algorithm [10] defines how the address pair for a communication session is selected, this address pair does not change for the duration of the session. Shim6 offers the ability to change the address pair used (and thus the path) during the session, transparent to the application. The Shim6 approach uses routable IP addresses (locators) as the identifiers visible to the transport layer. This also provides the facility to change the locator pair in use should REAP detect that the currently used pair of addresses (or interfaces) between two communication nodes has failed. REAP will search for a working pair of locators and pick another working pair (if available) when this occurs[7]. This change is performed at the network layer, which means that applications and transport protocols do not need any changes to benefit from this new capability.

[1] **E**xposing the **F**eatures in **IP** version **S**ix protocols that can be exploited/extended for the purposes of designing/building **A**utonomic **N**etworks and **S**ervices

3 Testbed

Our primary goal in this work was to get a baseline performance metric for an existing Shim6 implementation, and then to integrate Shim6 into the overall EFIPSANS architecture[11]. While Shim6 is already somewhat autonomous, in that it can detect and recover from link failures, we augmented LinShim6 with functionality to allow third party code to directly inform the Shim6 implementation which locators it should use. This facility could be used in the case of a scheduled downtime, for example. As a proof-of-concept for the EFIPSANS project, to demonstrate monitoring functionality, we developed a small daemon coupled with a Linux Netfilter[12] module to detect congestion or loss in a network through the Explicit Congestion Notification (ECN)[13] mechanism. This information could then be acted upon by third-party code to instruct LinShim6 to change the locator set in use, based on congestion detected and other variables such as, network load, jitter, delay etc.
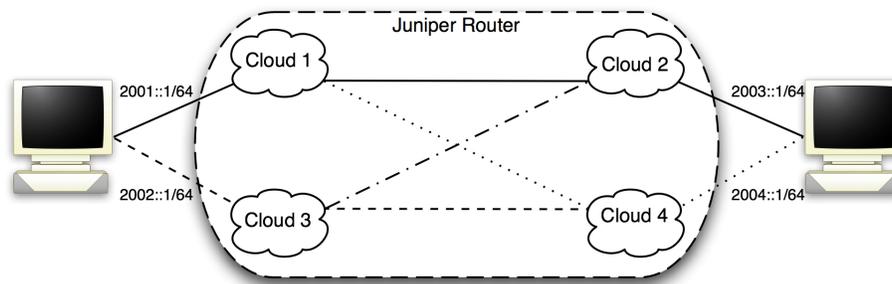
3.1 Shim6 testbed

Fig. 1: Shim6 testbed in TSSG

This testbed (figure 1) was set up to replicate the scenario in "Performance Analysis of REAchability Protocol for IPv6 Multihoming"[14]. We installed and configured the UCL Shim6 implementation, LinShim6, and proceeded to generate a set of results in order to ascertain what differences (if any) were present in the behaviour of this implementation versus the simulated results available in[14]. The testbed consisted of:

– Two Dual PII blade servers, each with 3 Network Interface Cards
– One Juniper M10i running JunOS

From our initial work on constructing this testbed, a number of issues arose. Initially we uncovered various bugs in the LinShim6 implementation, that became apparent due to our deployment being on real hardware. This led to much

work being done both by ourselves and Sébastien Barré, the LinShim6 author, to diagnose and fix these issues.

When deploying the testbed, it quickly became clear that the authors in[14] did not use any routing protocols in the simulation. As our testbed was deployed on real equipment we felt that is was important to be as realistic as possible so we used the Open Shortest Path First (OSPF)[15] protocol on the inter virtual-router links[2]. Obviously this meant that OSPF was able to recover from any single link failure by itself. Consequently, in order that a path failure could be simulated, the link between *Cloud 2* and *Cloud 3*, was manually disabled, thus restricting the available redundant paths between the two hosts.

When generating traffic, we originated all sessions from 2003::1 to 2001::1 through *Cloud 1* and *Cloud 2*. Then, to simulate failure, at a certain point in time, the link between *Cloud 1* and *Cloud 2* is disabled, this failure is detected by REAP, and, after path exploration, the session should continue between the locator pair of 2004::1 and 2002::1.

The tests performed involved the TCP[16, 17, 13] and UDP[18] protocols. For TCP tests, we evaluated the TCP behaviour using the FTP[19] protocol. The traffic used to evaluate the UDP behaviour corresponded, as close as possible to a Voice over IP (VoIP) application using a G.729[20] codec both with a uni-directional and bi-directional packet flow. To emulate G.729 the Iperf[21] tool was configured to generate 8 kilobits of data per second (50 packets per second, 20 bytes per packet).

The Round Trip Time (RTT) in both paths was configured to be identical. The Netem tool[22] was used to implement a normal distribution with a mean of 80ms and a 20ms variance. The "failure" event occured at a random interval between 75 and 125 seconds after the test run commences. All test runs were terminated 60 seconds after the "failure" event. These choices were dictated by those used in[14].

To run the tests, scripts were written to automate every run. For each value of T_{send} from 1 to 15, 45 test runs were completed. This gave a total of 675 test runs. This was done for each of TCP (the FTP protocol), bidirectional UDP, and unidirectional UDP. Giving over 2000 total runs or over 4000 unique log files.

3.2 Explicit Congestion Notification

Congestion is a perpetual problem in networks and can have a detrimental effect on user experience in situations where a high QoS is required (video streaming, VOIP etc). The Explicit Congestion Notification (ECN) protocol provides a means to detect congestion in IPv4 and IPv6 networks. Although it has been standardised for over a decade, it has suffered from slow uptake. This appears to be as a result of packet loss from intermediate routers rigidly enforcing earlier RFCs, and hence dropping packets as "invalid".

---

[2] OSPF is used internally in our site, and was a logical choice.

Briefly, when two endpoints have negotiated use of ECN, the sender of data packets will mark the outgoing packets with an ECN code point (2 bits in the IPv6 Traffic Class octet). An intermediate router approaching the point of congestion which comes across one of these packets will update it to signal that it is about to become congested, by setting the Congestion Encountered (CE) code point. Upon receipt of such a packet, the receiver will notify the sender via the ECN Echo (ECE) bit in the TCP header of the next TCP ACK, that the data packet experienced congestion. The sender then will take steps to "back-off" in an attempt to alleviate the congestion problem. Also, when an ECN-Capable TCP sender reduces its congestion window for any reason, the TCP sender sets the CWR bit in the TCP header of the first new data packet sent after the window reduction. This means there are two indicators available for use to use as congestion (or loss) indicators.

In order to facilitate some control over the chosen Shim6 network pair, we have extended the LinShim6 user space daemon *shim6d* with a "put" command. This command allows one to request Shim6 to use a specific locator pair at any time. However, the selected pair is still subject to the normal Shim6 rules in that if it should fail for whatever reason, Shim6 will automatically start the process to select a valid locator pair. For the EFIPSANS project, we have added functionality to LinShim6 to use the information presented by the ECN implementation when congestion (or loss) is detected and this information could potentially be used to migrate any affected Shim6 sessions to a clear path. The decision to migrate could be based on information such as prior knowledge of clear paths or other knowledge that could be supplied to the host from another service[23, 24].

The code consists of a Netfilter module and a user space daemon. The Netfilter module intercepts any ECE or CWR marked packets for the user space daemon to examine. If the daemon determines that the packet is, indeed of interest. The daemon will output the source and destination addresses of the effected stream via a network socket such that a listening application could make decisions based on this data. The output from the network socket is shown in listing 1.

```
labadmin@sam:~$ telnet localhost 2223
Trying ::1...
Connected to localhost.
Escape character is '^]'.
<?xml version="1.0" encoding="UTF-8"?>
<ecn src="2001:770:20:4::2" dest="2001:770:20:e::2" congested="true"/>
<?xml version="1.0" encoding="UTF-8"?>
<ecn src="2001:770:20:4::3" dest="2001:770:20:e::2" congested="true"/>
```

Listing 1: Output from network socket

4 Results

4.1 LinShim6

As mentioned already, our Shim6 testbed was set up to replicate the scenario depicted in[14]. In that paper the metric the authors used was "Application

Recovery Time". This is defined as the difference in time between the last packet arriving through the old locator set (addresses), and the first packet arriving through the new one, after the the path between the locator set has failed. This metric accurately measures the time taken to recover from a path failure when there is a continuous flow of traffic. The same metric was used for our measurements.
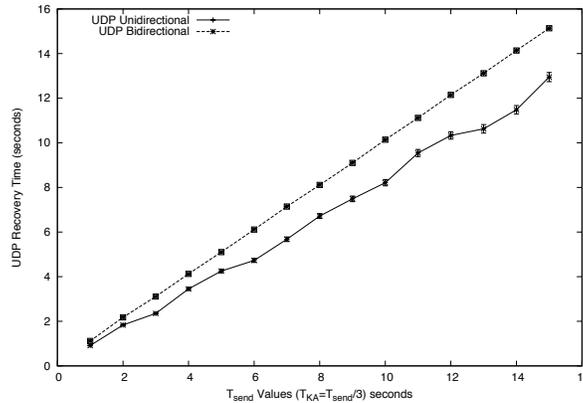

Fig. 2: UDP Recovery Time

UDP behaviour Figure 2 shows both bidirectional and unidirectional UDP recovery times. Two traffic profiles have been emulated. The Iperf network testing tool was used to generate a bidirectional UDP stream (VoIP conversation) and a unidirectional UDP stream (audio stream), with similar characteristics. Comparison with [14] reveals a marked similarity in results.

TCP behaviour TCP has several characteristics that UDP does not, such as reliability and congestion control. The authors used the FTP protocol to reliably generate high-bandwidth traffic. Also, as the LinShim6 implementation is capable of resetting TCPs retransmission timeout (RTO), we also performed this test. Figure 3 shows TCP recovery times for TCP both with and without the retransmission timers reset. Figure 4 compares bidirectional UDP and TCP. As can be seen from figure 3 and figure 4. The results obtained validates the proposals in §4.2 of [14]. In this work, the authors proposed that after a new path is chosen for a communication, that the TCP retransmission timer value should be reset. They argue that this is both more efficient and more appropriate as the timer values are dependent on the path in use now, not previously. Their simulation results showed that the relation between the TCP recovery time and T_{send} was was linear, and the the modified TCP behaviour was also very similar to that of UDP.
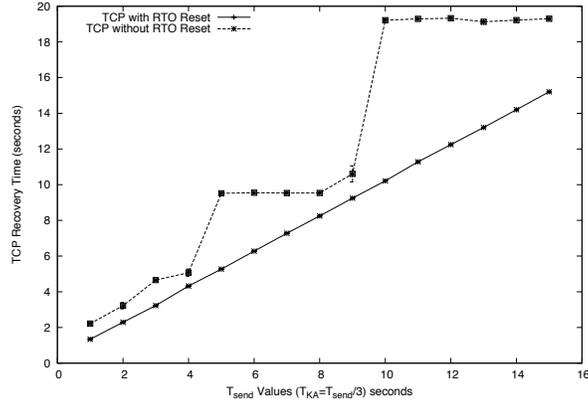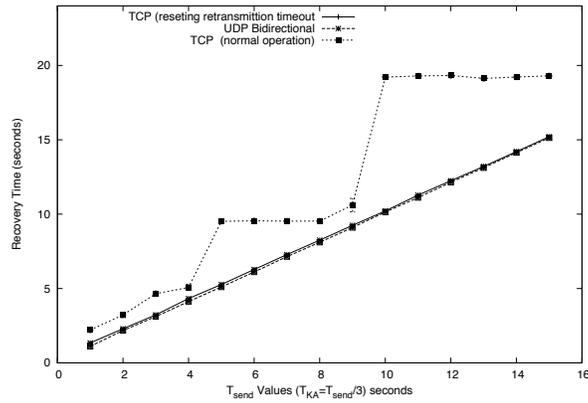
Fig. 3: TCP Recovery Time

Fig. 4: Here we can see that for TCP with RTO reset enabled, the behaviour is almost identical to that of bi-directional UDP. While unmodified TCP behaviour is clearly visible, showing the staircase like graph.

4.2 Comments on ECN

We were unable to successfully enable IPv6 ECN in the router equipment in our testbed. It appears that none of our routers (several Cisco, Juniper, Linux, FreeBSD) supported full IPv6 ECN functionality. However, we were able to test that our code functioned correctly. We were able to generate packet loss, and thus detect when the TCP sender set the CWR bit of the TCP header, as shown listing 2.

Consequently, we are confident that should we get to work with an IPv6 capable, ECN enabled router that we will be able to detect congestion and feed that information up to a management entity.

```
root@sam:~/src/tmp/ecnd# ./ecnd
[Signal] Creating signal handlers
[[ECN Queue]] Binding nfnetlink_queue as nf_queue handler for AF_INET6
[[ECN Queue]] Binding this socket to queue '0'
[Client] (client socket handle=5)
[ECN Monitor:parse()]
(seq=916108003, ack_seq=-1562873524) CWR = 1, ECE=0, SYN=0, ACK=1
[Process] Added path :
(src='2001:770:20:84::2',
dest='2001:770:20:84:250:c2ff:fe07:92db:) ECN=Yes
[ECN Monitor:parse()]
(seq=575124451, ack_seq=-1562873524) CWR = 1, ECE=0, SYN=0, ACK=1
[ECN Monitor:parse()]
(seq=-234932923, ack_seq=-1011743955) CWR = 1, ECE=0, SYN=0, ACK=1
```

Listing 2: Output from ecnd daemon

5 Conclusion

This paper presents details of work done in evaluating Université Catholique de Louvain's (UCL) publicly available Shim6 implementation. Its performance in our test network was compared against prior work, where the the behaviour of the Shim6 was simulated. The actual results obtained compare favorably with the simulation results. We then proceeded to implement a feedback mechanism based on the the Explicit Congestion Notification (ECN) protocol. This could allow for the re-balancing of traffic between hosts on clear (not experiencing congestion) paths should the hosts desire this functionality. Or indeed, in our case just act as a mechanism for reporting network congestion or loss to an EFIPSANS Managed Entity, which, monitors network performance.

We are also interested in testing our work across the Internet itself and gaining more relevant information as to the behaviour of Shim6 in larger deployments.

Acknowledgements

References

1. D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984 (Informational), September 2007.
2. Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006.
3. G. Huston. Architectural Approaches to Multi-homing for IPv6. RFC 4177 (Informational), September 2005.

4. C. de Launois and M. Bagnulo. The paths towards IPv6 multihoming. *IEEE Communications Surveys and Tutorials*, 8(2), 2006.
5. M. Bagnulo. Hash-Based Addresses (HBA). RFC 5535 (Proposed Standard), June 2009.
6. E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533 (Proposed Standard), June 2009.
7. J. Arkko and I. van Beijnum. Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming. RFC 5534 (Proposed Standard), June 2009.
8. S. Barré. Linshim6 - implementation of the shim6 protocol. Technical report, Université catholique de Louvain, Feb 2008.
9. S. Barré and O. Bonaventure. Shim6 implementation report : Linshim6. Internet draft, draft-barre-shim6-impl-03.txt, work in progress, September 2009.
10. R. Draves. Default Address Selection for Internet Protocol version 6 (IPv6). RFC 3484 (Proposed Standard), February 2003.
11. Nikolay Tcholtchev, Monika Grajzer, and Bruno Vidalenc. Towards a unified architecture for resilience, survivability and autonomic fault-management for self-managing networks. In *2nd Workshop on Monitoring, Adaptation and Beyond (MONA+)*, Stockholm, Sweden, November 23-24th 2009.
12. The Netfilter Project. Netfilter - firewalling, nat and packet mangling for linux. `http://www.netfilter.org`.
13. K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001.
14. 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.
15. J. Moy. OSPF Standardization Report. RFC 2329 (Informational), April 1998.
16. J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.
17. R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379.
18. J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.
19. J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), October 1985. Updated by RFCs 2228, 2640, 2773, 3659.
20. International Telecommunication Union. G.729, 2007. `http://www.itu.int/rec/T-REC-G.729/e`.
21. National Laboratory for Applied Network Research. Iperf. `http://iperf.sourceforge.net/`.
22. Stephen Hemminger. Network emulator. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`.
23. Damien Saucez, Benoit Donnet, and Olivier Bonaventure. Idips : Isp-driven informed path selection. IETF Draft, February 2008.
24. Olivier Bonaventure, Damien Saucez, and Benoit Donnet. The case for an informed path selection service. IETF Draft, February 2008.