Measuring IPv4 – IPv6 translation techniques

Technical Report 2012-001

Department of Computer Science

The University of Auckland

January 2012

Se-young Yu
syu051@aucklanduni.ac.nz

Brian E. Carpenter
brian@cs.auckland.ac.nz

ABSTRACT

This document reviews currently proposed IPv4-IPv6 translation techniques and describes a simple performance study of three open-source IPv4-IPv6 translators. The purpose of this document is to introduce the fundamental ideas behind NAT-PT, NAT64 and HTTP proxy and to measure the performance effect on round-trip time of using these translators in a simple network with up to 100 simultaneous connections.

Categories and Subject Descriptors

C.2.6 [**Computer-communication Networks**]: Internetworking – *routers.*

General Terms

Measurement, Performance, Experimentation.

Keywords

IPv6, NAT-PT, NAT64.

1. INTRODUCTION

Each device connected to the Internet needs at least one Internet Protocol (IP) address, used to route packets destined to the device or originated from the device.

The current IPv4 [2] uses a 32-bit address space which can hold up to four billion (2^{32}) addresses at most. Due to constant growth, the global IPv4 address pool is almost exhausted [1].

IPv6 [3] has slowly started to be deployed, replacing IPv4 with a 128-bit address space. It allows a theoretical maximum of 3.4×10^{38} addresses and it is expected to solve the address shortage problem. However, IPv6 is not backward compatible with IPv4.

We expect a long transition period, during which IPv4 and IPv6 must coexist in the Internet. During this time, various IPv4-IPv6 coexistence techniques such as tunnels will play a critical role. At some time, IPv6-only hosts will appear in large numbers, but many existing servers will support only IPv4. Most coexistence techniques will not help in this case: only some kind of translation of the packet stream will allow an IPv6-only client to communicate with an IPv4-only server. This paper is focused on these translation techniques.

We will present efficiency comparisons between three different IPv4-IPv6 translation techniques which have been defined and implemented. We are not aware of any other published measurement studies of this topic. In Section 2, we describe the fundamental algorithms of NAT-PT, Stateful NAT64 and HTTP Proxy. In section 3, we describe an experimental environment to measure simple translation efficiency and how it scales up with increasing numbers of connections. In sections 4 and 5, we present and discuss results from the experiments.

2. TRANSLATION TECHNIQUES

2.1 A. Network Address Translation – Port Translation

Basic Network Address Translation (NAT) translates an IPv4 address and a port number to a different IPv4 address and port number, in order to allow multiple client devices to share a single IPv4 address. Network Address Translation – Protocol Translation (NAT-PT) [4] similarly assigns a shared IPv4 address, and a translated port number, to an IPv6 client that uses NAT-PT to connect to a remote IPv4-only server. Each IPv6 client is assigned an IPv4 address and a translated port number when it starts a new flow through the NAT-PT. The IPv6 header from the client is translated into an IPv4 header, using the assigned address and port number, and the resulting packet is sent to the destination. Any IPv4 response packet from the server is also translated back to an IPv6 packet through NAT-PT, using the address and port information stored from the previous translation. If no IPv6 address information is found for such an IPv4 packet, the packet is silently discarded.

NAT-PT also includes dynamic translation of DNS queries and responses, but this does not affect the results given in this paper.

2.2 B. Stateful NAT64

Stateful NAT64 [5] formally replaces NAT-PT as an IETF Proposed Standard. Stateful NAT64 (often abbreviated as NAT64) uses a similar packet translation mechanism to NAT-PT, and requires a Binding Information Base and a Session table to maintain information about each session between IPv4 and IPv6 hosts (see Fig. 1 for an example). A Binding Information Base entry stores IPv6 address, port, shared IPv4 address and port of an IPv6 node. The session table entry stores two 5-tuples of packet headers, summarizing session information about the communicating nodes in both IPv4 and IPv6 forms.

NAT64 also uses a shared IPv4 address to proxy for an IPv6 host when establishing a session with an IPv4 host. The translation procedure is very similar to NAT-PT in Section A above. In place of a DNS translator, NAT64 requires an associated DNS64 server, which synthesizes DNS responses rather than translating them;

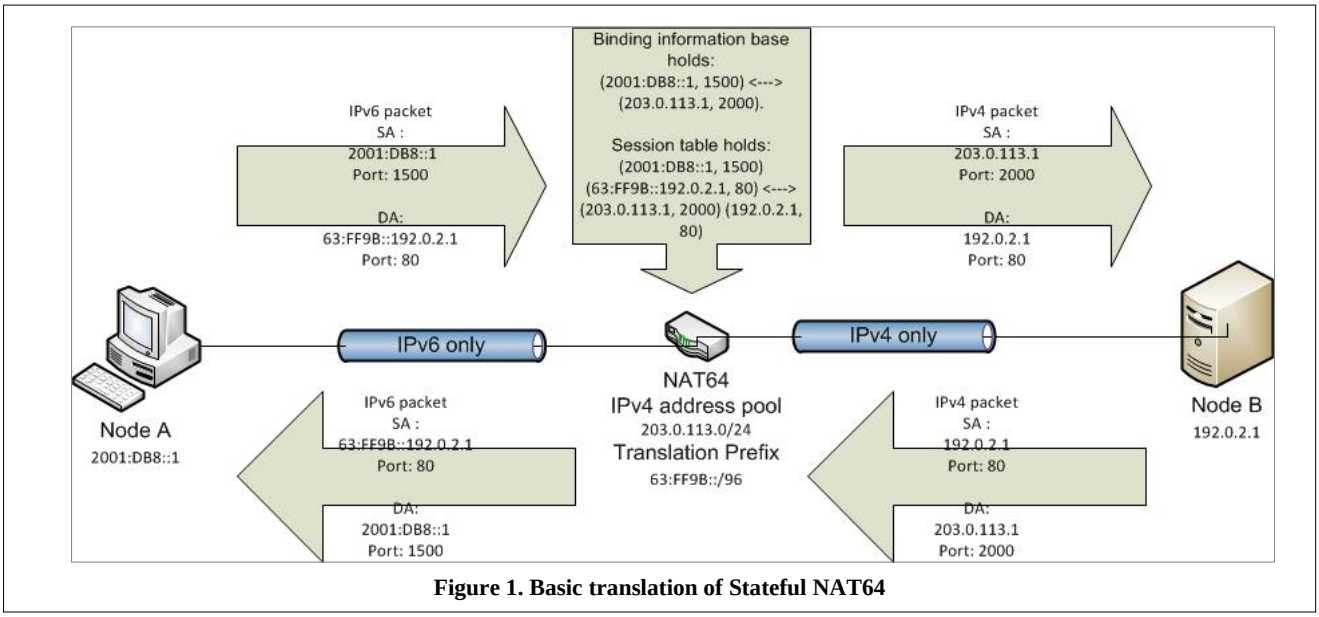

Figure 1. Basic translation of Stateful NAT64

this too does not affect the results in this paper. The other differences from NAT-PT are clarifications based on practical experience. They give no grounds to anticipate substantial changes in performance.

2.3 C. HTTP Proxy

An HTTP proxy [6] running in a dual-stack (IPv4 plus IPv6) host can be used to convert an IPv4 HTTP message into IPv6 or vice versa. The HTTP proxy mediates an HTTP session between IPv4 and IPv6 nodes by establishing a session to each node. When it receives an HTTP packet from either host destined to the other, it rebuilds an HTTP packet with the same HTTP content but using a different IP version. Logically, the original packet is not translated but recreated as a different packet with the same HTTP content.

3. EXPERIMENT ENVIRONMENT

The purpose of this experiment is to understand performance differences between the three techniques described above. We built a simple client – server network with IPv4-IPv6 translators deployed in a dual-stack host in the middle (Fig. 2). We sent HTTP packets from the client to the server and back through the translators, with various numbers of simultaneous connections, and measured the Round Trip Time of the packets returned. The packets themselves had two different sizes, referred to as “small” and “large”.

3.1 Machines involved

There are three nodes involved in this experiment, an IPv6 client, an IPv4-IPv6 translator and a simple web server. All three were PCs running Linux and appropriate open source software.

The IPv6 client is set to send various sizes of HTTP packets to the simple web server and measures the Round-Trip Time (RTT) between an HTTP message sent and an HTTP reply received from the web server; this is of course distinct from the underlying TCP RTT. The client normally has only IPv6 enabled and is connected to our IPv6-only network.

The IPv4-IPv6 translator sits in the middle between the client and web server. It is normally connected with the client via an IPv6 interface and the server via an IPv4 interface, so it is able to communicate with both nodes. The translator performs IPv4-IPv6 packet translation by intercepting all the packets that enter one interface and sending translated packets to the other interface. In the tests using native IPv4 or IPv6, this device acts as a regular IP router.

The simple web server receives each HTTP packet sent from the client through the translator and sends an HTTP reply packet back to the client.

The client, translator and HTTP server each has Ubuntu 9.1 karmic (kernel 2.6.31-16) as its operating system. The client has an Intel Core 2 Duo E8400 CPU with 3GHz, 3.2GB DDR2 memory and Intel 82567-LM-3 Gigabit network connection controller card. The translator and HTTP server both have an Intel Atom 330 CPU with 1.60GHz, 2GB DDR2 memory and a Realtek RTL8111/8168B PCI Gigabit Ethernet controller card.

3.2 Measurements

• The client sends two different packets for each set of experiments. One is a small HTTP request packet with size 107 bytes and another is a large HTTP request using an HTTP POST header with 1200 bytes of random characters in the contents, which makes an overall packet size of 1382 bytes.

• The client establishes 1 to 100 simultaneous connections with the server for each set of experiments. Each connection is used to send 10000 identical HTTP request packets and receive 10000 HTTP responses each time.

• The translators used in this experiment are naptd [7] from Lukasz Tomicki, Ecdysis from Viagenie [8] and Apache HTTP proxy server [9]. These translators implemented NAT-PT [4], NAT64 [5] and HTTP proxy [6] respectively and each was used for a set of experiments. The performance of these translators was measured and compared both with each other and with native IPv4 and IPv6 connections.

• The web server serves two different HTTP responses. One is a small HTTP response containing a single-character string “a” and another is a large HTTP response with a random string of length 1083. Both types of response are used for each experiment.

3.3 Domain name service handling

We deployed a DNS-ALG translator for NAT-PT, a DNS64 synthesizing resolver [10] for NAT64, and a BIND9 DNS server for the Apache HTTP proxy server. The client sends a DNS lookup for the web server when it needs to discover its IP address. The DNS request is sent to the central machine containing the translator, and the relevant DNS record is hard coded in the DNS server in this machine, with no further lookup required. The DNS response packet is captured by DNS-ALG or DNS64 as applicable and an IPv6 response is created by translation (DNS-ALG) or synthesis (DNS64). In the case of the HTTP proxy, DNS lookup is performed from the proxy server, not the client. This can make a small difference in the performance measurement of the translator, since the other two translation techniques do not require the machine where the translator is installed to initiate DNS lookup. However, the DNS response is always cached after the initial lookup, so the single DNS lookup has negligible impact on the average RTT measured over 10000 tests.

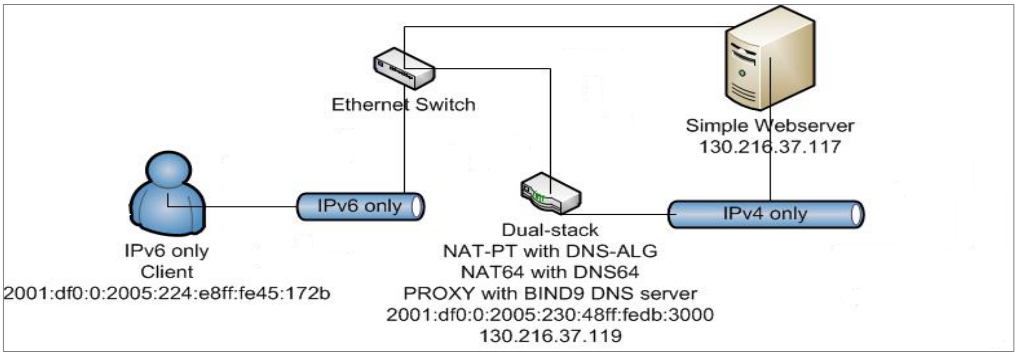

Figure 2. Network structure of the experiment

4. EXPERIMENT AND RESULTS

During the experiment we have measured performance of the three IPv4-IPv6 translators, as well as of native IPv4 and IPv6 connections, with all other conditions held constant, giving a total of five sets of results, each ranging up to 100 simultaneous connections carrying 10000 HTTP transactions.

Note that we use the terms NAT-PT and NAT64 to represent the naptd and Ecdysis NAT64 implementations. However, we did not compare the intrinsic efficiency of the NAT-PT and NAT64 specifications, but only of the selected open source implementations.

4.1 Experiment Procedure

1) For each set, the client sends one DNS lookup for the web server to the DNS server, which is the dual-stack machine with translator. In case of the HTTP proxy server, this DNS lookup is performed by the proxy server, therefore no DNS lookup is performed by the client nor is the DNS answer sent to the client; otherwise the DNS answer is sent back to the client from the DNS server’s local record.

2) The client establishes a single TCP connection to the web server through the IPv4-IPv6 translator after it receives the DNS answer. (In the native IP tests, the TCP connection is via the router.)

3) When the connection is established, the client sends a small HTTP request for a web page to the web server using IPv6. This packet is translated in the middle by IPv4-IPv6 translator and sent to the web server.

4) A small HTTP response from the web server is also translated and sent back to the client and the time between the HTTP request sent and the HTTP response received is recorded.

5) For each connection, 10000 HTTP requests are sent and the time is recorded for each HTTP request and response.

6) After a single connection, two simultaneous connections are established and do the same procedure from 2) to 5).

7) Increase number of simultaneous connections to 3, 4, 5, 10, 20, 30, 50, and 100 and do the same procedure from 2) to 5).

8) Keep the HTTP request size small while changing the HTTP response size to large, and otherwise do the same procedure from 2) to 7).

9) Change the HTTP request size to large and HTTP response size to small, and otherwise do the same procedure from 2) to 7).

10) Disable the IPv4-IPv6 translators and enable IPv4 only in all three machines involved. Do the same procedure from 2) to 9) with native IPv4 connection.

11) Disable IPv4 and enable IPv6 only in all three machines involved and do the same procedure from 2) to 9) with native IPv6 connection.

12) Collect RTT data based on the procedure above for each IPv4-IPv6 translator and for the native connections.

13) Classify the results based on the request/response sizes.

Fig. 3, 4 and 5 show the results for a range of conditions. Each figure shows boxplots, indicating in μs the median RTT as well as its statistical range, for native IP and the three translation methods. For each boxplot, dots represent each RTT measured from the translator on the bottom of X-axis. The upper quartile, median and lower quartile for each translator is represented as a solid box. We report the observed variations in RTT in percentage terms; the significance of these changes in practice is discussed in section 5.

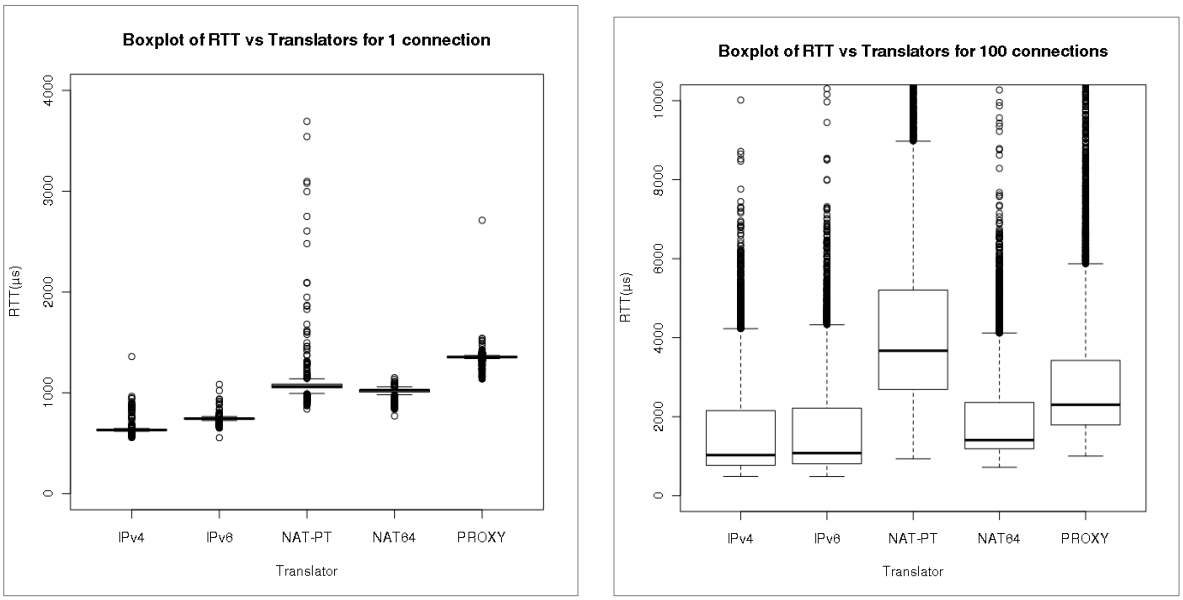

Figure 3. Boxplot of RTT vs Translator with small request / small response

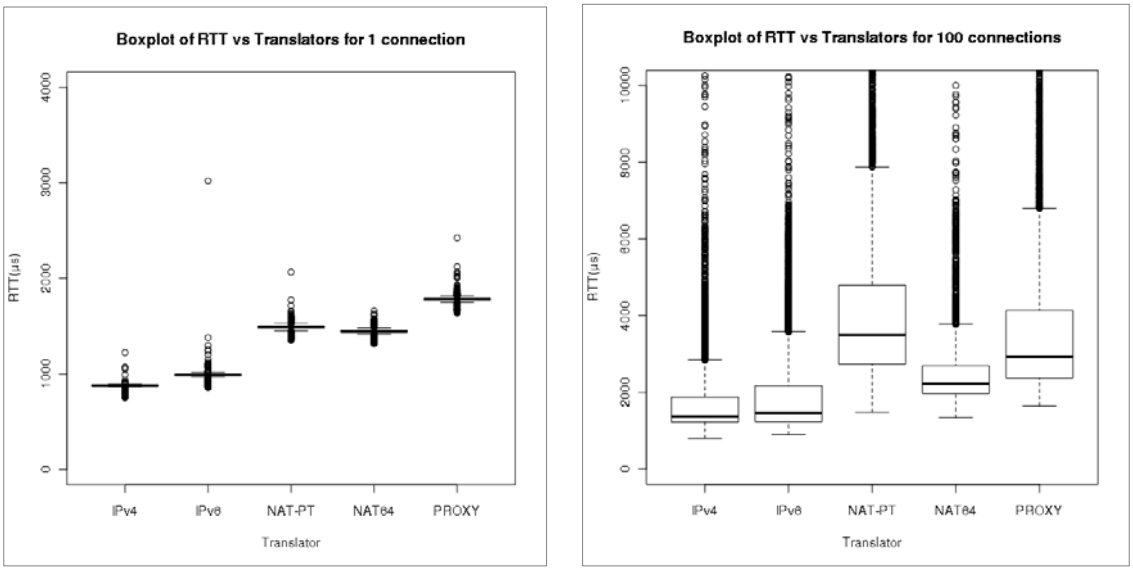

Figure 4. Boxplot of RTT vs Translator with small request /large response

4.2 Small HTTP request, Small HTTP response (Fig. 3)

There appear to be noticeable differences between the average RTT of IPv4-IPv6 translators and native connections.

Among the translators, NAT64 gave the shortest median RTT for both single connections and any number of simultaneous connections. It is, however, 46% slower than the median RTT of native IPv6 with a single connection. The median RTT of NAT64 increased steadily with the number of simultaneous connections, up to 20 simultaneous connections. The median RTT of NAT-PT increased rapidly with the number of simultaneous connections, even though the RTT of NAT-PT with few connections is similar to the RTT of NAT64. The variability of the RTT of NAT-PT also increased rapidly with the number of simultaneous connections. With 100 simultaneous connections, the median RTT of NAT-PT is 140% more than the median RTT of a native IPv6 connection. The median RTT of the HTTP proxy is also similar to NAT64 with a small number of simultaneous connections, and increases steadily as the number of connections increases. Above 50 simultaneous connections the median RTT of HTTP proxy settles down. The median RTT of the HTTP proxy with 100 simultaneous connections is 99% more than the median RTT of native IPv6 connection.

The median RTTs of native IPv6 and IPv4 connection do not differ much for any number of simultaneous connections. The median RTT of native IPv4 is only slightly faster than IPv6.

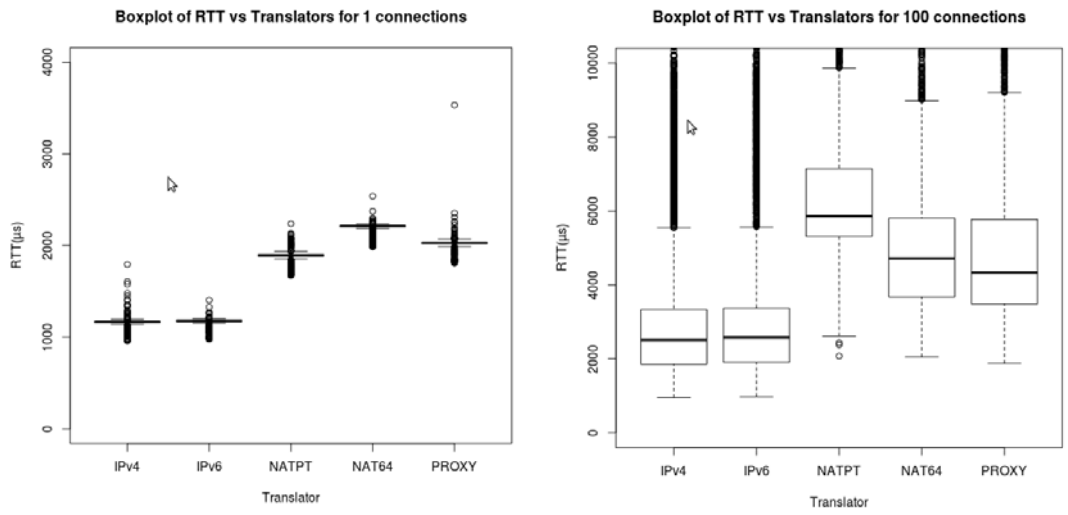

Figure 5. Boxplot of RTT vs Translator with large request / small response

4.3 Small HTTP request, Large HTTP response (Fig. 4)

With a large response size, we find a similar pattern of RTT values to section IV.B. We observe a noticeable difference of median RTT between native connections and translators.

Among the translators, NAT64 gives the lowest median RTT for any number of simultaneous connections. Compared to native IPv6 connection, NAT64 was only 46% slower when there was a single connection and 50% slower when there were 100 simultaneous connections. For NAT-PT, increasing the number of simultaneous connections caused the median RTT and variation to increase considerably compared to other translators, even though the median RTT of a single connection was similar to that for NAT64. Compared to a native IPv6 connection, the median RTT of NAT-PT was 140% greater when there were 100 simultaneous connections. The HTTP proxy showed a similar median RTT as in section IV.B. It showed 99% greater median RTT compared to the native IPv6 connection when there were 100 simultaneous connections.

4.4 Large HTTP request, small HTTP response (Fig. 5)

With a large request size we have observed an interesting pattern of the median RTT of NAT64. With a small number of simultaneous connections, the median RTT of NAT64 is the largest among the three translators. The median RTT of NAT64 with a single connection was 50% greater than the median RTT of native IPv6 connection, while NAT-PT and HTTP proxy were only 28% and 37% greater respectively (Fig. 5, left). After studying the code, and even with the assistance of the developers of Ecdysis NAT64, we were unable to explain its anomalous performance in this case. There is no theoretical reason to expect this anomaly, so we concluded that it was due to an unidentified implementation detail.

However, as the number of simultaneous connections increased, the median RTT of NAT-PT increased rapidly, while the median RTT of NAT64 did not increase much above 30 simultaneous connections. With more than 10 simultaneous connections, the median RTT of NAT-PT was greater than the median RTT of NAT64. With 100 simultaneous connections, the median RTT of NAT64 was 83% slower than the native IPv6 connection, while NAT-PT was 130% slower. The median RTT of the HTTP proxy increased rapidly until the number of simultaneous connection reached 50, but with more than 50 simultaneous connections the median RTT of HTTP proxy did not increase much more. The median RTT of the HTTP proxy was 69% slower than the native IPv6 connection with 100 simultaneous connections.

5. DISCUSSION

When we analyzed the specifications of NAT-PT, NAT64 and HTTP proxy, we developed a hypothesis about efficiency of translation based on the algorithmic structure of each translator. Firstly, we expected both NAT mechanisms to be more efficient than the HTTP proxy, since it has to rebuild every packet it receives from each node, executing full TCP processing as well as some application layer copying.

Secondly, we expected NAT-PT to be most efficient when there are a small number of simultaneous connections across the translator, because it has a somewhat simpler algorithm than NAT64, implying less load on the translator. With many simultaneous connections, we expected NAT64 to be most efficient, since it has a well-defined binding information management algorithm, which should more effectively manage a large amount of session state.

During the experiment, NAT-PT was less efficient than we expected. The median RTT of NAT-PT was greater than that of NAT64 with a single connection for all three sets of experiments. As the number of simultaneous connections increased, the median RTT of NAT-PT increased most rapidly among the translators in all conditions, and with 100 connections the median RTT of NAT-PT was the slowest among the translators, as expected.

NAT64 was the most efficient IPv4-IPv6 translator during the experiment, except in the set with large HTTP requests. Also, the

rate of increase of the median RTT with more connections was the least among the three translators. However, with large HTTP requests and few simultaneous connections, NAT64 had relatively inefficient translation. As noted above, this is most likely an implementation effect.

The HTTP proxy was more efficient than we expected. The median RTT with the HTTP proxy was slowest among the three translators with few simultaneous connections, except in the set with large HTTP requests where NAT64 was the slowest. However, with a large number of simultaneous connections, the HTTP proxy showed relatively efficient translation, always beating NAT-PT and usually close to NAT64. It appears that the processing cost of TCP and application layer copying scales almost as well as that of IP header translation.

The question arises whether the differences in RTT that we observed in laboratory conditions would matter in practice. The RTT for a simple HTTP command and response over the Internet will typically be in the range 10ms to 200ms, depending on circumstances. The worst case *increase* in median RTT due to translation that we observed was approximately 3ms. This will not significantly affect the user experience in most cases. On the other hand, the increased RTT reflects processing time in the translation device. A site operating a translator must ensure that it does not become a bottleneck.

6. CONCLUSION

In this paper we have compared translation efficiency between IPv4-IPv6 translators and native connections. Our study is based on our understanding of various IPv4-IPv6 packet translation techniques and we focused on measuring realistic performance metrics on a simple network. We have tested three open-source software packages: naptd (implementation of NAT-PT), Ecdysis (implementation of NAT64), and Apache HTTP proxy, by sending an HTTP over TCP over IPv6 packet to travel through each translator to be translated to IPv4, with the reply coming back from the IPv4 network through the translator.

The study showed that Ecdysis NAT64 is reasonably efficient in practice, except perhaps for a network which has a significant amount of large outbound packets and few simultaneous connections. With a small network, NAT64 works relatively efficiently compared to other translation techniques. If only native IPv6 connection is available and no other IPv4-IPv6 coexistence technique can be used, we recommend deploying NAT64 in order to communicate with IPv4 servers via an IPv6 connection. If only HTTP traffic is required, a dual stack Apache HTTP proxy is a reasonable alternative.

Our results and conclusions apply only to the particular implementations we have tested. As commercial implementations of NAT64 appear, they should be tested in a similar way to investigate their scaling behavior. It would also be of interest to test high-performance HTTP proxies, since our results show that it is not a foregone conclusion that NAT64 is faster or scales better than a proxy in all circumstances.

7. ACKNOWLEDGMENTS

Nevil Brownlee provided advice and support throughout the work reported here. We also thank Lukasz Tomicki for making NAT-PT source code available, Viagenie for making Ecdysis NAT64 source code available and Apache Foundation for making HTTP proxy source code available. Comments from anonymous reviewers have significantly improved this paper.

8. REFERENCES

[1] Hurricane Electric IPv4 Exhaustion Counters. http://ipv6.he.net/statistics/

[2] J. Postel. Internet protocol: DARPA internet program protocol specification. *Internet RFCs*. RFC 791, 1981

[3] S. Deering. R. Hinden. Internet protocol version 6 (IPv6) specification. *Internet RFCs*. RFC 2460, 1998.

[4] G. Tsirtsis. P. Srisuresh. Network address translation - protocol translation (NAT-PT). *Internet RFCs*. RFC 2766, 2000.

[5] M. Bagnulo. P. Matthews. I.V. Beijnum. Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers. *Internet RFCs*. RFC 6146, 2010.

[6] R. Fielding. J. Gettys. J.Mogul. Hypertext transfer protocol -- HTTP/1.1. *Internet RFCs*. RFC 2616, 1999.

[7] Network Address Translation, Protocol Translation IPv4/IPv6. http://tomicki.net/naptd.php

[8] Ecdysis: open-source implementation of a NAT64 gateway. http://Ecdysis.viagenie.ca/index.html.

[9] Apache HTTP server. http://httpd.apache.org/.

[10] M. Bagnulo. A. Sullivan. P. Matthews. I. van Beijnum. DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 Servers. *Internet RFCs*. RFC6147, 2011.