Performance Analysis and Comparison of the TAYGA and of the PF NAT64 Implementations

Gábor Lencse and Sándor Répás

***Abstract*—The transition mechanisms for the first phase of IPv6 deployment are surveyed and the most important NAT64 solutions are selected. The test environment and the testing method are described. As for the selected NAT64 implementations, the performance of the TAYGA running under Linux and of the Packet Filter (PF) of OpenBSD was measured and compared. The stability of the tested NAT64 solutions was analyzed under serious overload conditions to test if they may be used in production environments with strong response time requirements.**

***Keywords*—IPv6 deployment, IPv6 transition solutions, NAT64, performance analysis, PF, TAYGA.**

I. INTRODUCTION

The performance and stability of the different NAT64 [1] implementations will be an important topic for network operators in the following years because on the one hand the global IPv4 Address Pool is being depleted[1] and on the other hand the vast majority of the Internet still uses IPv4 only. Thus from the many issues of the co-existence of IPv4 and IPv6, the communication of an IPv6 only client with an IPv4 only server is the first practical task to solve in the upcoming phase of the IPv6 deployment because internet service providers (ISPs) can still supply the relatively small number of new servers with IPv4 addresses from their own pool but the huge number of new clients can get IPv6 addresses only. DNS64 [2] and NAT64 are the best available techniques that make it possible for an IPv6 only client to communicate with an IPv4 only server. (A brief description of their operation will be provided later.)

Different free NAT64 implementations were considered and two of them were selected for testing. The aim of our research was to compare the performance of the selected implementations running on different free operating systems and to analyze their behavior under heavy load conditions.

Note that the performance analysis and comparison of some selected DNS64 implementations under Linux, OpenBSD and FreeBSD was also a part of our research and our results were presented in [5].

The remainder of this paper is organized as follows: first, some possible techniques are mentioned for the communication of an IPv6 only client with an IPv4 only server, then the operation of the DNS64+NAT64 solution is introduced and a short survey of the results of the most current publications is given, second, the selection of the NAT64 implementations is discussed, third, our test environment is described, fourth, the performance measurement method of the NAT64 implementations is detailed, fifth, the NAT64 results are presented and discussed, and finally, our conclusions are given.

The volume of the IPv6 traffic of the Internet is low in many countries of the western world now and it will probably grow slowly for some years, but its volume may explode later on and the networks should be ready to cope with it. Thus our results are expected to give valuable information to many network administrators when selecting the appropriate IPv6 transition solution for their networks.

II. IPV6 TRANSITION MECHANISMS FOR THE FIRST PHASE OF IPV6 DEPLOYMENT

A. The Most Important Solutions

The authors conceive that the deployment of IPv6 will take place by a long co-existence of the two versions of the Internet Protocol and in the first phase of the IPv6 transition, the main issue will be the communication of an IPv6 only client with an IPv4 only server. Several mechanisms can be used for this task, of which the most notable ones are:

1. NAT-PT/NAPT-PT [6] started its life as a proposed standard in 2000 but due to several issues it was put to historic status in 2007 [7].
2. The use of an Application Level Gateway [8] is an operable alternative, however it is rather expensive as ALGs have to be both developed and operated for all the different applications.
3. The most general and flexible solution is the use of a DNS64 server and a NAT64 gateway.

B. The Operation of DNS64 and NAT64

To enable an IPv6 only client to connect to an IPv4 only server, one can use a *DNS64 server* and a *NAT64 gateway*. The DNS64 server should be set as the DNS server of the IPv6 only client. When the IPv6 only client tries to connect to any server, it sends a recursive query to the DNS64 server to find the IPv6 address of the given server. If the given server has an IPv4 address only then the DNS64 server constructs and returns a special IPv6 address called *IPv4-Embedded IPv6 Address* [9] containing the IPv4 address of the server in the last 32 bits. In the first 96 bits, it may contain the *NAT64 Well-known Prefix* or a *network specific prefix* from the network of the client.

Manuscript received February 11, 2013. This research was supported by the TÁMOP-4.2.2.C-11/1/KONV-2012-0012: "Smarter Transport" – IT for co-operative transport system – The Project is supported by the Hungarian Government and co-financed by the European Social Fund. This publication was supported by the TÁMOP-4.2.2/B-10/1-2010-0010 project and by the Széchenyi István University.

Gábor Lencse is with the Department of Telecommunications, Széchenyi István University, 1 Egyetem tér, Győr, H-9026, Hungary (phone: +36-30-409-56-60, fax: +36-30-96-613-636, e-mail: lencse@sze.hu).

Sándor Répás is with HunNet-Média Ltd. 18-22. Victor Hugo, Budapest, H-1132, Hungary (e-mail: RSandor@AHOL.co.hu)

[1] IANA delegated the last five "/8" IPv4 address blocks to the five Regional Internet Registries in 2011 [3], of which APNIC has already depleted its IPv4 address pool in 2011 and RIPE NCC did so during the writing of this paper on September 14, 2012 [4]. It means that RIPE NCC also uses a more strict allocation policy for its very last /8 block.

The route towards the network with the given IPv6 prefix should be set in the IPv6 only client (and in all of the routers along the route from the client to the NAT64 gateway) so that the packet go through the NAT64 gateway.

The IPv6 only client uses the received IPv6 address to set up a connection to the desired (IPv4 only) server. The client sends an IPv6 packet to the received IPv6 address. When its packet arrives to the NAT64 gateway, the gateway builds an IPv4 packet using the payload (and some header fields) of the IPv6 packet and it sets the destination address of the IPv4 packet according to the rightmost 32 bits of the destination address of the IPv6 packet. These 32 bits contain exactly the IPv4 address of the desired server. The source address of the IPv4 packet is set to be the IPv4 address of the NAT64 gateway. The NAT64 gateway sends out the IPv4 packet and the packet arrives to the IPv4 only server. The IPv4 only server responds the normal way using the source address of the received IPv4 packet as the destination address of its answer and in this way the server sends its response to the NAT64 gateway. The gateway receives the IPv4 packet and builds an IPv6 packet using the payload (and some header fields) of the IPv4 packet and its own data about the given connection. (The NAT64 gateway uses stateful NAT or some other method to track the IPv6 – IPv4 mapping.) Finally, the NAT64 gateway sends the IPv6 packet back to the client.

For a more detailed but still easy to follow introduction, see [10] and for the most accurate and detailed information, see the relating RFCs [1] and [2].

C. A Short Survey of the Current Research Results

Several papers were published in the topic of the performance of DNS64 and NAT64 in 2012. The performance of the TAYGA NAT64 implementation (and implicitly of the TOTD DNS64 implementation) is compared to the performance of NAT44 in [11]. The performance of the Ecdysis NAT64 implementation (that has its own DNS64 implementation) is compared to the performance of the authors' own HTTP ALG in [12]. The performance of the Ecdysis NAT64 implementation is compared to the performance of both the NAT-PT and an HTTP ALG in [13]. All of these papers deal with the performance of a given DNS64 implementation with a given NAT64 implementation. On the one hand this is natural, as both services are necessary for the operation, on the other hand this is a kind of "tie-in sale" that may hide the real performance of a given DNS64 or NAT64 implementation by itself. Even though both services are necessary for the complete operation, in a large network they are usually provided by separate, independent devices; DNS64 is provided by a name server and NAT64 is performed by a router. Thus the best implementation for the two services can be – and also should be – selected independently. The performance of the BIND DNS64 implementation and performance of the TAYGA NAT64 implementation are analyzed separately and also their stability is tested in [14]. However, only one implementation was considered for each service, so even if they were proved to be stable and fast enough, more research is needed for the comparison of the performance (and also the stability) of multiple DNS64 and NAT64 implementations.

A good survey of the most recent DNS64 and NAT64 research results is given in [15]. They also demonstrated that the DNS64+NAT64 system is a viable solution for an internet service provider. However the stability of the different DNS64 and NAT64 implementations under heavy load conditions and the comparison of their performance were not addressed there.

Due to the space limitations of our (before mentioned) previous paper about the stability and performance of different DNS64 implementations [5], our results concerning NAT64 implementations are now published here.

III. The Selection of NAT64 Implementations

Only free software [16] (also called open source [17]) implementations were considered.

Ecdysis [18] could have been a choice, as it was the first NAT64 implementation, and it also contains DNS64 support. However, Ecdysis contains a non-official Linux kernel module and it is unfortunately not stable. Ecdysis was tested with version 2.6.32, 2.6.35, 2.6.37 and 3.0.1 kernels and it froze many times. In addition to that, the home page of the project does not reflect any development since November 17, 2010. For these reasons, two other NAT64 implementations were selected for performance analysis: TAYGA and PF.

TAYGA [19] is a free software under GPLv2 license and according to its developers it was intended to provide *production quality NAT64 service*. TAYGA is a *stateless* NAT64 solution for Linux. It means that by itself it can create only a one-to-one mapping between IPv6 and IPv4 addresses. For this reason TAYGA is used together with a stateful NAT44 packet filter (**`iptables`** under Linux): TAYGA maps the source IPv6 addresses to different IPv4 addresses from a suitable size of private IPv4 address range, and from the private IPv4 addresses the stateful NAT44 packet filter performs an SNAT (Source Network Address Translation) to the public IPv4 address of the NAT64 gateway (sometimes also called Port Address Translation, PAT). In the reverse direction, the stateful NAT44 packet filter "knows" which private IPv4 address belongs to the reply packet arriving to the IPv4 interface of the NAT64 gateway. After the NAT44 translation TAYGA can determine the appropriate IPv6 address using its one-to-one address mapping and then it rewrites the packet to IPv6.

Note that TAYGA is able to store the one-to-one IPv6 – IPv4 address mappings on disk, therefore, in case of a system crash TAYGA can continue using these after restart. (On the basis of our experiences with TAYGA, we do not think this functionality would be much used.)

When configuring TAYGA, a suitably large private IPv4 address range should be provided.

The Packet Filter (PF) of OpenBSD 5.1 [20] includes NAT64 support that is based on the Ecdysis program code [21]. PF is a free software under the BSD license.

PF [22] was first released in OpenBSD v3.0 in 2001. PF is a very powerful tool to filter and manipulate IP packets in modern BSD systems. It can be configured by editing **`/etc/pf.conf`**, and packets can be manipulated by many attributes. PF supports NAT, load balancing, logging and it can also operate as stateless and stateful packet filter at the same time.

PF supports IPv4 and IPv6 stateful NAT for many years, and now it supports NAT64, too. This feature of PF is called

address family translation. PF in stateful mode can translate many IPv6 client addresses to one outgoing IPv4 address via address family translation. Because of stateful translation, PF needs to build a "states table" to find the correct IPv6 destination address of the incoming IPv4 packets. There is no need to use stateless and stateful NAT one after the other. In this case, a single state table is enough for perfect system operation against TAYGA's two-table solution. Therefore PF generates lower load for the NAT64 gateway.

IV. The Test Environment for NAT64 Performance Measurements

The aim of our tests was to examine and compare the performance of the selected NAT64 implementations. We were also interested in their stability and behavior under heavy load conditions. (For testing the software, some hardware had to be used, but our aim was not the performance analysis of any hardware.)

A. The Structure of the Test Network

The topology of the network is shown in Fig. 1. The central element of the network is the NAT64 gateway. The eight DELL IPv6 only workstations at the bottom of the figure played the role of the clients for the NAT64 measurements. The number i^{th} client (i=1..8) imitated to reach all the hosts in the 10.i.0.0/16 IPv4 only network. No DNS64 servers were used for the mapping, rather the IPv4-embedded IPv6 addresses were constructed 'manually' by the test script appending the 32 bits of the given IPv4 address to the 2001:738:2c01:8001:ffff:ffff::/96 network specific prefix.

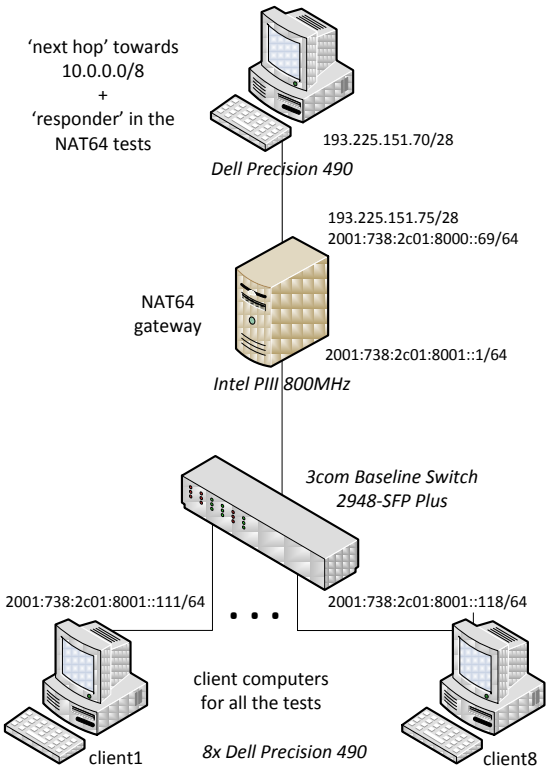

Figure 1. Topology of the NAT64 test network.

At the NAT64 gateway, the address of the next hop router towards the 10.0.0.0/8 network was set to 193.225.151.70. The DELL workstation with this IP address responded instead of all of the hosts with IP addresses from the 10.0.0.0/8 network, see more details later on.

B. The Configuration of the Computers

A test computer with special configuration was put together for the purposes of the NAT64 gateway in order that the clients will be able to produce high enough load for overloading it. The CPU and memory parameters were chosen to be as little as possible from our available hardware base in order to be able to create an overload situation with a finite number of clients, and only the network cards were chosen to be fast enough. The configuration of the test computer was:

- Intel D815EE2U motherboard
- 800MHz Intel Pentium III (Coppermine) processor
- 256MB, 133MHz SDRAM
- Two 3Com 3c940 Gigabit Ethernet NICs

For all the other purposes (the 8 client computers and the next hop router towards the 10.0.0.0/8 network) standard *DELL Precision Workstation 490* computers were used with the following configuration:

- DELL 0GU083 motherboard with Intel 5000X chipset
- Two Intel Xeon 5130 2GHz dual core processors
- 2x2GB + 2x1GB 533MHz DDR2 SDRAM (accessed dual channel)
- Broadcom NetXtreme BCM5752 Gigabit Ethernet controller (PCI Express)

Debian Squeeze 6.0.3 GNU/Linux operating system was installed on all the computers (including the Pentium III test computer when it was used under Linux). The version of the OpenBSD operating system installed on the test computer was 5.1.

V. NAT64 Performance Measurement Method

A. NAT64 Gateway Settings

1) Preparation of the TAYGA system

The network interfaces of the freshly installed Debian Squeeze Linux operating system on the Pentium III computer were set according to Fig. 1.

In order to facilitate the IPv6 SLAAC (*Stateless Address Autoconfiguration*) of the clients, **radvd** (*Router Advertisement Daemon*) was installed on the test computer.

The settings in the file **/etc/radvd.conf** were the following ones:

```
interface eth2
{
        AdvSendAdvert on;
        AdvManagedFlag off;
        AdvSendAdvert on;
        prefix 2001:738:2c01:8001::/64
        {
                        AdvOnLink off;
        };
        RDNSS 2001:738:2c01:8001::1 {};
};
```

Then the TAYGA system was installed on the Pentium III test computer and it was configured as a NAT64 gateway. The

following modifications were done in the **/etc/tayga.conf** file:

```
tun-device nat64
ipv4-addr 172.16.0.1
dynamic-pool 172.16.0.0/12
prefix 2001:738:2c01:8001:ffff:ffff::/96
```

The further settings were done by the following script:

```
#!/bin/bash
tayga --mktun
ip link set nat64 up
ip addr add 172.16.0.1 dev nat64
ip addr add 2001:738:2c01:8001::2 dev nat64
ip route add 172.16.0.0/12 dev nat64
ip route add 2001:738:2c01:8001:ffff:ffff:/96 dev nat64
tayga
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
iptables -t nat -A POSTROUTING -o eth2 -j MASQUERADE
ip route add 10.0.0.0/8 via 193.225.151.70
```

During the preliminary tests, the kernel of the NAT64 gateway sent "Neighbour table overflow" messages. (The *neighbour table* is the IPv6 counterpart of the IPv4 ARP cache.) The different limits for the size of the neighbour table were raised as follows:

```
cd /proc/sys/net/ipv6/neigh/default/
echo 4096 > gc_thresh1
echo 8196 > gc_thresh2
echo 16384 > gc_thresh3
```

2) Preparation of the PF system

The same PIII computer was used to test the PF NAT64 solution as with the TAYGA system.

The network interfaces of the OpenBSD 5.1 operating system on the test computer were set according to Fig. 1.

The following modifications were performed in the **/etc/pf.conf** file:

```
set limit states 40000
pass in on sk1 inet6 from any to \
  2001:738:2c01:8001:ffff:ffff::/96 af-to inet from \
  193.225.151.75
```

The first command line generated such a large state table which is able to handle all the states of NAT64. The second line (actually broken into three lines in the paper) enabled NAT64 function.

The IPv4 and IPv6 packet forwarding were enabled in the **/etc/sysctl.conf** file:

```
net.inet.ip.forwarding=1
net.inet6.ip6.forwarding=1
```

B. The Settings of the 'Responder' Computer

For the testing of the NAT64 gateway, "someone" had to answer in the name of the IPv4 only hosts. A DELL computer (the same configuration as the clients) was used for this purpose. This host had the 193.225.151.70 IP address. At the DELL computer, the packets towards the 10.0.0.0/8 network were redirected to the computer itself by the following **iptables** rule:

```
iptables -t nat -A PREROUTING -d 10.0.0.0/8 \
  -j DNAT --to-destination 193.225.151.70
```

As the DELL computer had much more computing power than that of the Pentium III test computer, it was able to answer instead of the IPv4 only computers easily. E.g. in the case of the below described performance measurements with 8 clients, the CPU utilization of the DELL computer was under 5%.

C. Client Settings

The DELL computers were used as clients. For the simplicity of the accounting of their addresses, they were configured manually from the 2001:738:2c01:8001::111-118 range but they received all the other settings from the **radvd**. The next hop towards the 2001:738:2c01:8001:ffff:ffff::/96 network was set to the IPv6 address of the NAT64 gateway in the client computers:

```
route add -A inet6 2001:738:2c01:8001:ffff:ffff::/96 \
  gw 2001:738:2c01:8001::1
```

D. NAT64 Performance Measurements

The script below was executed by 1, 2, 4 and 8 clients to produce an increasing load for the performance measurements.

```
#!/bin/bash
i=`cat /etc/hostname | grep -o .$`
for b in {0..255}
do
  rm -r $b
  mkdir $b
  for c in {0..63..4}
  do
    ping6 -c11 -i0 -q \
      2001:738:2c01:8001:ffff:ffff:10.$i.$b.$c \
      >> $b/nat64p-10-$i-$b-$c &
    ping6 -c11 -i0 -q \
      2001:738:2c01:8001:ffff:ffff:10.$i.$b.$((c+1)) \
      >> $b/nat64p-10-$i-$b-$((c+1)) &
    ping6 -c11 -i0 -q \
      2001:738:2c01:8001:ffff:ffff:10.$i.$b.$((c+2)) \
      >> $b/nat64p-10-$i-$b-$((c+2)) &
    ping6 -c11 -i0 -q \
      2001:738:2c01:8001:ffff:ffff:10.$i.$b.$((c+3)) \
      >> $b/nat64p-10-$i-$b-$((c+3))
  done
done
```

Using the **ping6 -c11** command, eleven *echo request* ICMPv6 messages were sent to all of the generated IPv6 addresses. To provide a high enough load, four **ping6** commands were started in (quasi) parallel[2] utilizing the capabilities of the two dual core CPUs of the DELL computers.

The *synchronized start* of the client scripts was done by using the "Send Input to All Sessions" function of the terminal program of KDE (called **Konsole**).

Every single client was used to simulate the load of a high number of clients and in every series of measurements, the number of clients was increased from one to eight using the values for the number of clients: 1, 2, 4 and 8 to be able to double the load of the clients.

The response times of the **ping6** commands were measured. The *CPU and memory utilization* were also measured on the test computer running NAT64.

The following command line was used under Linux:

```
dstat -t -c -m -l -p --unix --output load.csv
```

The command line was the following under OpenBSD:

```
vmstat -w 1 >load.txt
```

[2] The first three ones were started in a so-called asynchronous way using the **&** sign at the end of the command lines thus the four commands were running nearly parallel.

VI. NAT64 PERFORMANCE RESULTS

The outer **for** cycle of the test script using variable **b** was executed 256 times and average and standard deviation of the measured values were calculated. The results are displayed in similar tables for both test cases, and only the first table is explained in detail.

A. NAT64 Performance Results of TAYGA

The results can be found in Table I. Row 1 shows the number of clients that executed the test script. (The load of the clients is proportional with the number of the clients.) The packet loss ratio is displayed in the second row. Rows 3, 4 and 5 show the average, the standard deviation and the maximum values of the response time (expressed in milliseconds), respectively. The following two rows show the average and the standard deviation of the CPU utilization of the test computer. Row 8 shows the number of forwarded packets per seconds that was calculated from the average traffic arriving to the test computer from the direction of the clients measured in bytes. (In the calculations, the size of the messages carrying the ICMPv6 echo requests was always 100 bytes.) The last row shows the memory consumption measured at the test computer.

Evaluation of the results:

- Though packet loss occurred even for a single client, the packet loss ratio was always very low (under 0.03 percent, which means that 3 packets were lost from 10,000 packets).
- The response time showed nearly linear increase in the function of the load: the doubling of the load approximately doubled the response time as well.
- The number of packets served per second could increase in a certain extent until there was free CPU capacity. For four clients, the CPU was nearly fully utilized and for eight clients, TAYGA could not serve more packets due to the lack of CPU capacity; the number of packets served per second even decreased from 7,085 to 6,890 by less than 3%.
- The memory consumption was always low and it was only very slightly increasing in the function of the load.

To sum up the findings above, we can lay down that TAYGA performed well, its memory consumption was found to be low and its response time increased approximately linearly with the load, that is, TAYGA complied with the *graceful degradation* principle [23].

B. NAT64 Performance Results of PF

The results can be found in Table II. Evaluation of the results:

- The packet loss rate was very low (0.02 percent).
- The response time showed less then linear increase in the function of the load while there was free CPU capacity. It increased approximately linearly with the load when the number of clients was increased from four to eight.
- The number of packets served per second could seriously increase with the number of clients. From one to two clients, it was nearly doubled, and for eight clients, it likely come close to saturation, however the CPU utilization was still about 90% for eight clients, so PF may forward even a little more than 22,800 packets per second.
- The memory consumption was very low and increased less than linearly with the load.

As far as we could test, PF complied with the graceful degradation principle, but even with eight clients it had some free CPU capacity. It had also very low memory consumption.

C. Comparison of the performance of TAYGA and PF

Comparing the performance results of TAYGA and PF, we can state that their response time and the number of forwarded packets per seconds were similar at moderate load. However at serious overload conditions, PF could handle 22,886 packets per seconds with an average response time of 1.2 milliseconds while TAYGA could do only 6,890 packets per second with an average response time of 4.4 milliseconds.

TABLE I. NAT64 PERFORMANCE: TAYGA

1	**number of clients**		**1**	**2**	**4**	**8**
2	**packet loss (%)**		0.01	0.01	0.03	0.03
3	**response time of `ping6` (ms)**	average	0.447	0.957	1.986	4.406
4		std. deviation	0.103	0.158	0.321	0.474
5		maximum	5.202	5.438	8.057	13.965
6	**CPU utilization (%)**	average	69.8	84.3	97.8	100.0
7		std. deviation	3.3	1.7	2.2	0.1
8	**traffic volume (packets/s)**		5,721	6,614	7,085	6,890
9	**memory consumption (MB)**		10.8	11.4	11.9	12.9

TABLE II. NAT64 PERFORMANCE: PF

1	**number of clients**		**1**	**2**	**4**	**8**
2	**packet loss (%)**		0.02	0.02	0.02	0.02
3	**response time of `ping6` (ms)**	average	0.405	0.486	0.606	1.194
4		std. deviation	0.050	0.073	0.127	0.250
5		maximum	1.770	1.433	3.904	7.055
6	**CPU utilization (%)**	average	31.7	50.1	80.1	90.3
7		std. deviation	5.4	5.0	5.1	4.7
8	**traffic volume (packets/s)**		5,909	11,091	18,367	22,886
9	**memory consumption (MB)**		2.4	3.5	5.5	7.9

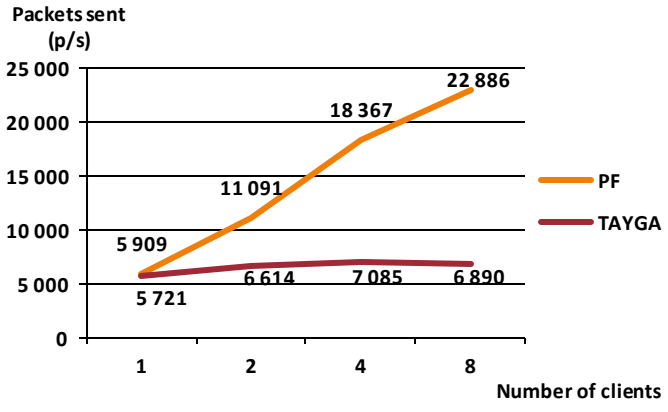

Figure 2. Number of forwarded pacekts per second in the function of the number of clients

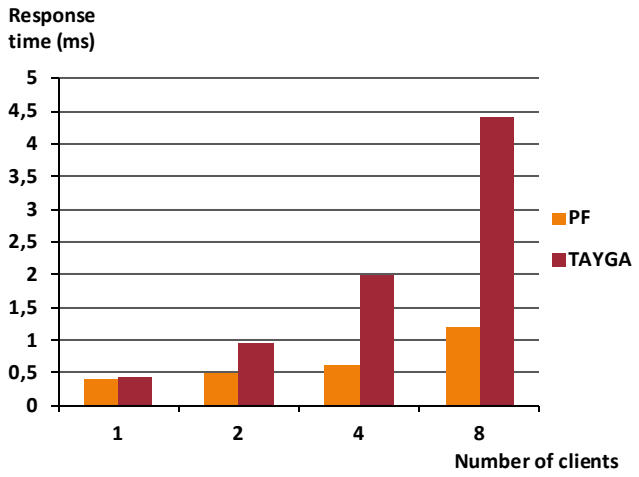

Figure 3. Response time in the function of the number of clients

This is a very large difference in their performance. At least two reasons can be identified.

1. As TAYGA is a stateless NAT64 solution and another stateful NAT44 solution had to be used thus every packet were touched twice.
2. As TAYGA runs in user space, every packet had to be copied first from kernel space to user space and then back from user space to kernel space.

In spite of this difference of their performance under heavy load conditions, we can lay down that both TAYGA and PF are stable and can be used as a NAT64 solution – probably for many years. System administrators anticipating high traffic volume from their IPv6 only clients towards the IPv4 Internet are encouraged to use PF.

VII. CONSLUSIONS

Due to the exhaustion of the IPv4 address pool, the internet service providers will not be able to provide IPv4 addresses to an increasing number of clients. The application of a DNS64 server and a NAT64 gateway is found to be the best solution for the IPv6 only clients to access the IPv4 Internet. The appropriate choice of a stable and high performance NAT64 implementation can be crucial for an ISP. Two free NAT64 implementations were tested concerning their stability and performance. Both TAYGA under Linux and PF under OpenBSD proved to be stable thus they both can be used in a production environment like the network of an ISP. Their performance was found to be similar under moderate load conditions but PF 3.3 times outperformed TAYGA under heavy overload conditions. The PF of OpenBSD is our number one recommendation for a NAT64 solution.

This is the author's version of the paper. For personal use only, not for redistribution. The definitive version can be found in the *Proceedings of the 36th International Conference on Telecommunications and Signal Processing (TSP 2013)*, (Rome, Italy, 2013. July, 2-4.) pp. 71-76. © IEEE

REFERENCES

[1] M. Bagnulo, P. Matthews and I. Beijnum, "Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers", IETF, April 2011. ISSN: 2070-1721 (RFC 6146)

[2] M. Bagnulo, A Sullivan, P. Matthews and I. Beijnum, "DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers", IETF, April 2011. ISSN: 2070-1721 (RFC 6147)

[3] The Number Resource Organization, "Free pool of IPv4 address space depleted" http://www.nro.net/news/ipv4-free-pool-depleted

[4] RIPE NCC, "RIPE NCC begins to allocate IPv4 address space from the last /8", http://www.ripe.net/internet-coordination/news/ripe-ncc-begins-to-allocate-ipv4-address-space-from-the-last-8

[5] G. Lencse and S. Répás, "Performance analysis and comparison of different DNS64 implementations for Linux, OpenBSD and FreeBSD", *Proc. 27th IEEE International Conference on Advanced Information Networking and Applications* (AINA-2013, March 25-28, 2013) Barcelona, Spain, pp. 877-884.

[6] G. Tsirtsis and P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)", IETF, February 2000. (RFC 2766)

[7] C. Aoun and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status", IETF, July 2007. (RFC 4966)

[8] P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", IETF, August 1999. (RFC 2663)

[9] C. Bao, C. Huitema, M. Bagnulo, M Boucadair and X. Li, "IPv6 addressing of IPv4/IPv6 translators", IETF, October 2010. ISSN: 2070-1721 (RFC 6052)

[10] M. Bagnulo, A. Garcia-Martinez and I. Van Beijnum, "The NAT64/DNS64 tool suite for IPv6 transition", IEEE Communications Magazine, vol. 50, no. 7, July 2012, pp. 177-183. doi:10.1109/MCOM.2012.6231295

[11] K. J. O. Llanto and W. E. S. Yu, "Performance of NAT64 versus NAT44 in the Context of IPv6 Migration", in *Proc. International MultiConference of Engineers and Compuer Scientists* 2012 Vol I. (IMECS 2012, March 14-16, 2012), Hong Kong, pp. 638-645

[12] C. P. Monte et al, "Implementation and evaluation of protocols translating methods for IPv4 to IPv6 transition", *Journal of Computer Science & Technology*, vol. 12, no. 2, pp. 64-70

[13] S. Yu, B. E. Carpenter, "Measuring IPv4 – IPv6 translation techniques", Technical Report 2012-001, Department of Computer Science, The University of Auckland, January 2012

[14] G. Lencse and G. Takács, "Performance Analysis of DNS64 and NAT64 Solutions", *Infocommunications Journal*, vol. 4, no 2, June, 2012. pp. 29-36.

[15] E. Hodzic, S. Mrdovic, "IPv4/IPv6 Transition Using DNS64/NAT64: Deployment Issues", *Proc. 2012 IX International Symposium on Telecommunications* (BIHTEL, Oct. 25-27, 2012), Sarajevo, Bosnia and Herzegovina

[16] Free Software Fundation, "The Free Software Definition", http://www.gnu.org/philosophy/free-sw.en.html

[17] Open Source Initiative, "The Open Source Definition", http://opensource.org/docs/osd

[18] "Ecdysis: open-source implementation of a NAT64 gateway" http://ecdysis.viagenie.ca

[19] "TAYGA: Simple, no-fuss NAT64 for Linux" http://www.litech.org/tayga/

[20] Theo de Raadt, "The OpenBSD 5.1 Release", May 1, 2012, ISBN 978-0-9784475-9-5, http://www.openbsd.org/51.html

[21] Simon Perreault, "[Ecdysis-discuss] NAT64 in OpenBSD", http://www.viagenie.ca/pipermail/ecdysis-discuss/2011-October/000173.html

[22] P. N. M. Hansteen, *The Book of PF: A No-Nonsense Guide to the OpenBSD Firewall*, 2nd ed., San Francisco: No Starch Press, 2010. ISBN: 978-1593272746

[23] NTIA ITS, "Definition of 'graceful degradation'" http://www.its.bldrdoc.gov/fs-1037/dir-017/_2479.htm