Performance Analysis and Comparison of 6to4 Relay Implementations

Gábor Lencse
Department of Telecommunications
Széchenyi István University
Győr, Hungary

Sándor Répás
Department of Telecommunications
Széchenyi István University
Győr, Hungary

***Abstract*—the depletion of the public IPv4 address pool may speed up the deployment of IPv6. The coexistence of the two versions of IP requires some transition mechanisms. One of them is 6to4 which provides a solution for the problem of an IPv6 capable device in an IPv4 only environment. From among the several 6to4 relay implementations, the following ones were selected for testing: sit under Linux, stf under FreeBSD and stf under NetBSD. Their stability and performance were investigated in a test network. The increasing measure of the load of the 6to4 relay implementations was set by incrementing the number of the client computers that provided the traffic. The packet loss and the response time of the 6to4 relay as well as the CPU utilization and the memory consumption of the computer running the tested 6to4 relay implementations were measured. The implementations were tested also under very heavy load conditions to see if they are safe to be used in production systems.**

Keywords—IPv6 deployment; IPv6 transition solutions; 6to4; performance analysis

I. INTRODUCTION

The majority of the current Internet still uses the Internet Protocol version 4 (IPv4) for forwarding the packets of the communication of the applications. Even though IPv6 was defined in 1998 [1], it has not replaced IPv4 yet. As of Aug. 31, 2013, only 1.88% of the Internet traffic reaching Google used IPv6 [2]. The coexistence of the two versions of IP results in different issues (e.g. the two endpoints of the communication use different IP versions, or the endpoints use the same IP version but the communication path between the endpoints supports the other version only). Several *transition mechanisms* were developed to solve the different issues of the coexistence of the two versions of the Internet Protocol. These *theoretical solutions* are defined in different RFCs. There are a number of *implementations* for each solutions. When a network operator decides to support some of the IPv6 transition mechanisms, it can be a difficult task to choose the right implementations because there can be security, reliability and performance issues. Several papers were published in the topic of performance analysis of different IPv6 transition implementations.

One of the most important driving forces of the deployment of IPv6 is the depletion of the global IPv4 address pool. IANA delegated the last five "/8" IPv4 address blocks to the five Regional Internet Registries in 2011 [3]. Therefore an important upcoming coexistence issue is the problem of an IPv6 only client and an IPv4 only server, 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 [4] and NAT64 [5] are the best available techniques that make it possible for an IPv6 only client to communicate with an IPv4 only server. Another very important coexistence issue comes from the case when the ISP does not support IPv6 but the clients do and they would like to communicate with IPv6 servers. The most matured solution for this problem is called 6to4 [6]. The stability and performance analysis of different 6to4 implementations is the topic of this paper.

The remainder of this paper is organized as follows: first, a short survey of the results of the most current publications is given, second, 6to4 is introduced, third, the selection of the 6to4 relay implementations is discussed, fourth, our test environment is described, fifth, the performance measurement method of the 6to4 relay implementations is detailed, sixth, the results are presented and discussed, seventh, the validity of our results is considered and our plans for the future research are presented, finally, our conclusions are given.

This topic was identified as being of high importance for network administrators.

II. A SHORT SURVEY OF CURRENT RESEARCH RESULTS

A. The Problem of an IPv6 Only Client and an IPv4 Only Server

Several papers were published in the topic of the performance of DNS64 and NAT64 in 2012 and 2013. The performance of the TAYGA NAT64 implementation (and implicitly of the TOTD DNS64 implementation) is compared to the performance of NAT44 in [7]. 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 [8]. The performance of the Ecdysis NAT64 implementation is compared to the performance of both the NAT-PT and an HTTP ALG in [9]. All of these papers deal with the common performance of a given DNS64 implementation with a given NAT64 implementation. The performance of the BIND DNS64 implementation and performance of the TAYGA NAT64 implementation are

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.

analyzed separately and also their stability is tested in [10]. A good survey of the most recent DNS64 and NAT64 research results is given in [11]. They also demonstrated that the DNS64+NAT64 system is a viable solution for an internet service provider. Our results about the stability and performance of different DNS64 and NAT64 implementations were published in [12] and [13], respectively.

B. *The Problem of an IPv6 Capable Client in an IPv4 Only Environment*

A good survey of IPv6 transition mechanisms including both translation and tunneling solutions can be found in [14]. It discusses 6to4 among the tunneling mechanisms. Ref. [15] named 6to4 and Teredo [16] the two most widely used transition solutions on the bases of the IPv6 prefixes in use. The performance of 6to4 is addressed in [17]. They prepared a controlled environment and compared the performance characteristics (round trip time and throughput) of 6to4 to the native IPv4 and IPv6 using both TCP and UDP between the endpoints. They used Cisco routers in the test network. In contrast, we have chosen different free software [18] (also called open source [19]) implementations for stability testing and performance comparison.

III. Introduction To 6to4 In A Nutshell

The aim of 6to4 is to help those IPv6 capable devices that are residing in an IPv4 environment to connect to other devices being in the same situation and to the native IPv6 internet. The solution is an "automatic tunnel" that encapsulates the IPv6 packets into IPv4 packets (using protocol number 41, as the configured IPv6 over IPv4 tunnel [20]).

The IPv6 capable device can be a single host having a 6to4 pseudo-interface that performs the encapsulation of the IPv6 packets into IPv4 packets and also the decapsulation in the opposite direction. This is called *6to4 host*. It is also possible that there are multiple IPv6 devices in an IPv6 network behind a so-called *6to4 (border) router* that performs the encapsulation of the IPv6 packets into IPv4 packets and the decapsulation in the opposite direction. These *6to4 IPv6 devices* can communicate with other 6to4 IPv6 devices or with IPv6 devices on the native IPv6 internet. In the latter case, they need a 6to4 relay at the border of the IPv4 internet and the IPv6 internet, see Fig. 1.

It is a precondition of the applicability of the 6to4 solution that a 6to4 host or a 6to4 router must have a public IPv4 address. The IPv6 addresses for the IPv6 capable devices will get IPv6 addresses from the 2002::/16 prefix. The next 32 bits of their IPv6 addresses are the 32 bits of the public IPv4 address of the 6to4 host or 6to4 router and still there are 16 bits for subnetting. (It can be filled with 0 or chosen randomly if no subnetting is needed.) The last 64 bits of the IPv6 address may be generated in the usual way from the MAC addresses of the hosts using the modified EUI-64 algorithm.

If the communication occurs between two IPv6 capable devices that both use 6to4 then the route of the encapsulated packet in the IPv4 internet is exactly determined by the public IPv4 addresses of the two 6to4 hosts/routers. If one of the communication endpoints resides in the native IPv6 internet then the route of the packet must go through a 6to4 relay. There are multiple 6to4 relays having the same 192.88.99.1 anycast address and the network will use the nearest one.

The forthcoming example scenario can be followed in Fig. 1. Let our client having the 2002:c000:208::2 6to4 IPv6 address communicate with the server having the 2001:db8::2 global IPv6 address. The client sends out its IPv6 packet containing its own IPv6 address as the source address and the IPv6 address of the server as the destination address. The packet arrives to the 6to4 router as the default gateway. The 6to4 router encapsulates the IPv6 packet in an IPv4 packet using its own IP address, 192.0.2.8 as the source address and the 192.88.99.1 anycast address as the destination address. The protocol type is set to 41, which indicates that an IPv6 packet was embedded. The packet arrives to the nearest 6to4 relay at the 192.88.99.1 anycast address. The relay recognizes the 41 protocol type and thus it decapsulates the IPv6 packet and sends towards its destination. The server receives the packet and replies in the normal way addressing its reply packet to the client. In the global public IPv6 internet, the 2002::/16 prefix is routed (using anycast addressing) towards the nearest 6to4 relay, which may be different from the one that was used by the packet travelling from the client to the server (asymmetric routing). The 6to4 relay receives the IPv6 packet and encapsulates it in an IPv4 packet. It determines the target IPv4 address from the destination IPv6 address as it contains the 192.0.2.8 IPv4 address of the 6to4 router next to its 2002::/16 prefix: 2002:**c000:208**::2. When the 6to4 router receives the packet it simply decapsulates the embedded IPv6 packet and sends it to the client.

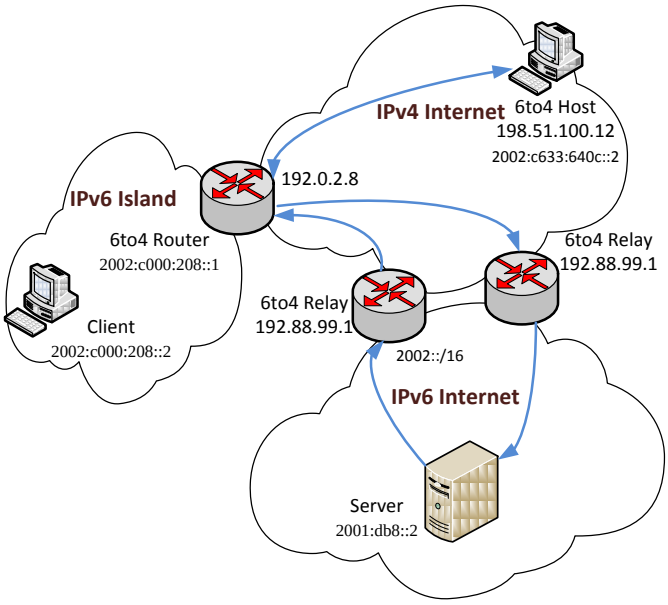

Fig. 1. Sample network for the demonstration of 6to4

Note that if two devices using both 6to4 IPv6 addresses communicate with each other then the 6to4 routers or hosts never send the IPv4 packet (containing the encapsulated IPv6 packet) to a 6to4 relay rather they always take the 32 bits next to the 2002::/16 prefix of the destination IPv6 address and use these 32 bits as the target IPv4 address. A 6to4 target address can be easily recognized from its 2002::/16 prefix. All other

target addresses are treated as global IPv6 addresses and the destination address of the IPv4 packet is set to the 192.88.99.1 anycast address of the 6to4 relays.

Let us consider the performance requirements of the devices performing 6to4 operations.

- A 6to4 host performs encapsulation and decapsulation of the traffic from and to the local host.
- A 6to4 router performs encapsulation and decapsulation of the traffic from and to a limited number of hosts behind the router.
- A 6to4 relay may be responsible for the traffic of a huge number of hosts. Because of the anycast addressing, their number and thus the load of the 6to4 relay depend on the location of other 6to4 relays.

Therefore, if a 6to4 relay using the 192.88.99.1 anycast IPv4 address is set up then it may receive huge load. Consequently, it should be a stable system that does not collapse even in serious overload situation rather complies with the graceful degradation principles [21].

IV. Selection Of Implementations For Testing

As it was mentioned before, only free software [18] (also called open source [19]) implementations were considered for stability testing and performance comparison. We had multiple reasons for this decision:

- The licenses of certain vendors (e.g. [22] and [23]) do not allow the publication of benchmarking results.
- Free software can be used by anyone for any purposes thus our results can be helpful for anyone.
- Free software is free of charge for us, too.

In our previous research efforts of performance and stability analysis of IPv6 transition solutions (DNS64 [12] and NAT64 [13]), we used Linux, OpenBSD and FreeBSD as host operating systems. Unfortunately 6to4 is not implemented in OpenBSD for security concerns [24]. Thus, the following implementations were selected for testing: sit under Linux, stf under FreeBSD and stf under NetBSD.

V. Test Environment

The aim of our tests was to examine and compare the performance of the selected 6to4 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. 2. The central element of the network is the 6to4 relay. The ten Dell workstations at the bottom of the figure played the role of the 6to4 hosts for the 6to4 relay performance measurements. The two Dell computers at the top of the figure responded to all the 6to4 hosts.

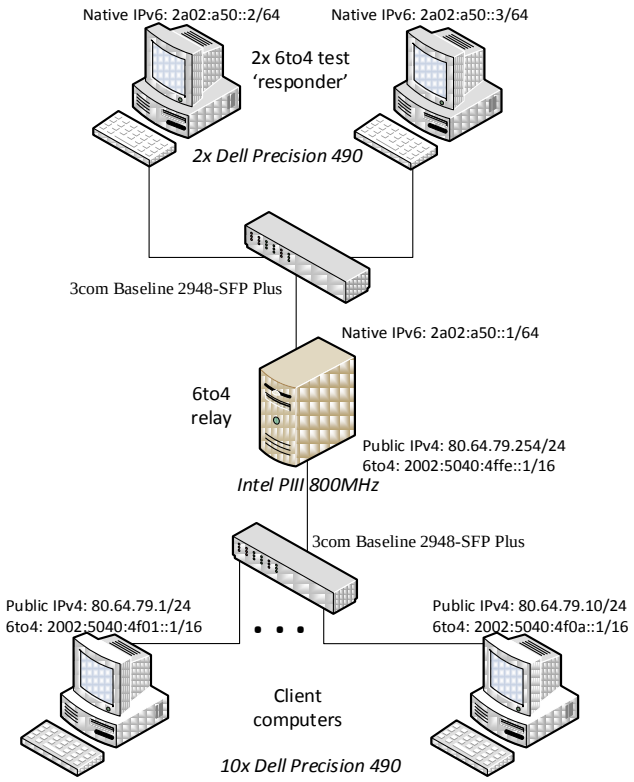

Fig. 2. Topology of the 6to4 test network.

Even though their hardware was much more powerful than that of the 6to4 relay (see details later), we used two of them to be sure that the responder part of the system was never a bottleneck during our tests.

B. The Hardware Configuration of the Computers

A test computer with special configuration was put together for the purposes of the 6to4 relay so that the 6to4 hosts 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 6to4 hosts, 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 the 6to4 host purposes, standard *DELL Precision Workstation 490* computers were used with the following configuration:

- DELL 0GU083 motherboard with Intel 5000X chipset
- Two Intel Xeon 5140 2.33GHz dual core processors
- 4x1GB 533MHz DDR2 FB-DIMM SDRAM (quad channel)

- Broadcom NetXtreme BCM5752 Gigabit Ethernet controller (PCI Express)

Note that these computers were the same as those used in the DNS64 and NAT64 tests ([12] and [13]) but with a little faster CPU and four identical RAM modules which were able to operate quad channel.

The responder computers were similar to the 6to4 computers but they had somewhat slower CPUs:

- Two Intel Xeon 5130 2GHz dual core processors

Debian Squeeze 6.0.3 GNU/Linux operating system was installed on all the computers including the Pentium III test computer acting as a 6to4 rely when it was used under Linux. The version number of FreeBSD and NetBSD were 9.0 and 6.0.1, respectively. The responder computers had the OpenBSD 5.1 operating system. OpenBSD was chosen because it supports NAT66, which was needed for our experiments.

C. The Software Configuration of the Computers

Fig. 2 shows the IP addresses of the Ethernet interfaces. The 6to4 hosts had IPv4 and 6to4 IPv6 addresses. The 6to4 relay had two Gigabit Ethernet interfaces: eth1 was used for communication with the 6to4 hosts and it had the 80.64.79.254 public IPv4 address and the 2002:5040:4ffe::1 6to4 address; eth2 was used for communication with the responder computers and it had the 2a02:a50::1 global IPv6 address. The responder computers had the 2a02:a50::2 and 2a02:a50::3 global IPv6 addresses.

To make the results comparable, the same Pentium III computer was used for 6to4 purposes under Linux, FreeBSD and NetBSD. The network settings were also identical. Fig. 3 shows the exact settings of the interfaces under Linux containing the starting of the sit 6to4 pseudo interface and also the routing. Note that some of the tests were also performed with one responder only. Fig. 3 contains the settings for both cases. The network settings and the routing of FreeBSD are shown in Fig. 4 and the FreeBSD stf tunnel was started by the commands shown in Fig 5. The configuration files for the network interfaces under NetBSD are shown in Fig. 6 and Fig 7. The routing under NetBSD was set and the stf tunnel was started by the script in Fig. 8.

As it can be seen in Fig. 3 and Fig. 4 the packets to the 2001:0738:2c01:8001:ffff:0:a00::/104 network were directed to the responder computer(s). On the responder computers, NAT66 was used to redirect these packets to the computer itself (see Fig. 9), to be able to respond to them.

Note that the CPU utilization was monitored during the measurements, and even when using only one responder, its maximum CPU utilization was about 10%.

The network setting of the 6to4 hosts were done as shown in Fig. 10.

```
#to the 6to4 hosts
auto eth1
iface eth1 inet static
 address 80.64.79.254
 netmask 255.255.255.0

#to the responders
auto eth2
iface eth2 inet6 static
 address 2a02:a50::1
 netmask 64
pre-up echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
pre-up echo 0 > /proc/sys/net/ipv6/conf/eth2/autoconf
pre-up echo 0 > /proc/sys/net/ipv6/conf/eth2/accept_ra
#In the case of TWO responders:
post-up ip -f inet6 route add 2001:0738:2c01:8001:ffff:0000:0a00::/105 via 2a02:a50::2
post-up ip -f inet6 route add 2001:0738:2c01:8001:ffff:0000:0a80::/105 via 2a02:a50::3
# In the case of ONE responder:
post-up ip -f inet6 route add 2001:0738:2c01:8001:ffff:0000:0a00::/104 via 2a02:a50::2

#6to4
auto sit0
iface sit0 inet6 static
 address 2002:5040:4ffe::1
 netmask 16
```

Fig. 3. Configuration of the Linux interfaces, routing and the sit tunnel (/etc/network/interfaces).

```
ifconfig_sk0_ipv6="inet6 2a02:a50::1/64"
ifconfig_sk1="inet 80.64.79.254 netmask 255.255.255.0"
ipv6_static_routes="respondernet respondernet2"
ipv6_route_respondernet="2001:0738:2c01:8001:ffff:0000:0a00::/105 2a02:a50::2"
ipv6_route_respondernet2="2001:0738:2c01:8001:ffff:0000:0a80::/105 2a02:a50::3"
ipv6_gateway_enable="YES"
```

Fig. 4. Configuration of the FreeBSD interfaces and routing when using two responders (/etc/rc.conf).

```
ifconfig stf0 create
ifconfig stf0 inet6 2002:5040:4ffe::1 prefixlen 16 alias
```

Fig. 5. Starting script of the FreeBSD 6to4 tunnel (start6to4).

```
up
media autoselect
inet6 2a02:a50::1 prefixlen 64 alias
```

Fig. 6. Configuration of the interface towards the clients, NetBSD (/etc/ifconfig.sk0).

```
up
media autoselect
80.64.79.254 netmask 0xffffff00 media autoselect
```

Fig. 7. Configuration of the interface towards the responders, NetBSD (/etc/ifconfig.sk1).

```
route add -inet6 2001:0738:2c01:8001:ffff:0000:0a00:: -prefixlen 105 2a02:a50::2
route add -inet6 2001:0738:2c01:8001:ffff:0000:0a80:: -prefixlen 105 2a02:a50::3
ifconfig stf0 create
ifconfig stf0 inet6 2002:5040:4ffe::1 prefixlen 16 alias
```

Fig. 8. Configuration script under NetBSD (start6to4).

```
set timeout interval 2
set limit states 400000
match in on em1 inet6 to 2001:0738:2c01::/48 rdr-to em1
```

Fig. 9. Redirection on the responder computers under OpenBSD (/etc/pf.conf).

```
auto eth0
iface eth0 inet static
address 80.64.79.{1..10} # that is 80.64.79.1 on client1, 80.64.79.2 on client2, etc.
netmask 255.255.255.0

auto tun6to4
iface tun6to4 inet6 v4tunnel
 address 2002:5040:4f0{1..a}::1 # {} is to be interpreted as above
 netmask 16
 gateway ::80.64.79.254
 endpoint any
 local 80.64.79.{1..10} # {} is to be interpreted as above
```

Fig. 10. Configuration of the network interfaces of the 6to4 hosts under Linux (/etc/network/interfaces).

VI. Performance Measurement Method

In order to be able to exactly tune the measure of the load from moderate to serious overload in controlled steps, the number of the clients (6to4 hosts) was tuned. First, a series of measurements was conducted by a single client. Second, the test system was restarted and two clients were used, etc. The measurements were done by the execution of the script in Fig. 11. Different destination IPv6 addresses were used to simulate real-life situation. In a given series of experiments, each active client sent 256*64*8*11=1,441,792 ICPMv6 echo requests (sending eleven of them to each one of the different 256*64*8=131.072 IPv6 addresses). Using the "**&**" sign for asynchronous command execution, eight `ping6` commands were executed quasi-parallel utilizing the computing power of the two dual core CPUs. The target address range, 2001:0738:2c01:8001:ffff:0000:0a00::/104 was cut into two halves. The lower half side of the address range, 2001:0738:2c01:8001:ffff:0000:0a**0**0::/105 was used by the commands containing the variable **i**, and the higher half side of the address range, 2001:0738:2c01:8001:ffff:0000:0a**8**0::/105 was used by the commands containing the variable **j**. Thus the application of two responders could be easily done by using only two lines in the routing table of the 6to4 relay.

```
#!/bin/bash
i=`cat /etc/hostname|grep -o .$`
j=$((i+128))
for b in {0..255}
    do
    rm -r $b
    mkdir $b
    for c in {0..252..4}
        do
        ping6 -c11 -i0 -q
2001:738:2c01:8001:ffff:0000:10.$i.$b.$c
>$b/6to4p-10-$i-$b-$c &
        ping6 -c11 -i0 -q
2001:738:2c01:8001:ffff:0000:10.$j.$b.$c
>$b/6to4p-10-$j-$b-$c &
        ping6 -c11 -i0 -q
2001:738:2c01:8001:ffff:0000:10.$i.$b.$((c+1))
>$b/6to4p-10-$i-$b-$((c+1)) &
        ping6 -c11 -i0 -q
2001:738:2c01:8001:ffff:0000:10.$j.$b.$((c+1))
>$b/6to4p-10-$j-$b-$((c+1)) &
        ping6 -c11 -i0 -q
2001:738:2c01:8001:ffff:0000:10.$i.$b.$((c+2))
>$b/6to4p-10-$i-$b-$((c+2)) &
        ping6 -c11 -i0 -q
2001:738:2c01:8001:ffff:0000:10.$j.$b.$((c+2))
>$b/6to4p-10-$j-$b-$((c+2)) &
        ping6 -c11 -i0 -q
2001:738:2c01:8001:ffff:0000:10.$i.$b.$((c+3))
>$b/6to4p-10-$i-$b-$((c+3)) &
        ping6 -c11 -i0 -q
2001:738:2c01:8001:ffff:0000:10.$j.$b.$((c+3))
>$b/6to4p-10-$j-$b-$((c+3))
    done
done
```

Fig. 11. 6to4 relay performace test script (ping-test-8d.sh).

The CPU utilization and the memory consumption of the 6to4 relay was measured under BSD with the following command:

```
vmstat –w 1 > load.txt
```

Under Linux, the command line was the following:

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

VII. Measurement Results

A. Linux sit tunnel with two responders

The results can be found in Table 1. (The tables were put on the same page for the synoptic view and easy comparison of the results of the 6to4 implementations.) This table and all the other tables with the results are to be interpreted as follows. Row 1 shows the number of clients that executed the test script. (The load of the 6to4 relay was proportional with the number of the clients.) The packet loss ratio is displayed in the second row. Rows 3 and 4 show the average and the standard deviation 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 7 shows the number of forwarded packets per seconds. The last row shows the estimated memory consumption measured at the test computer. Note that this parameter can be measured with high uncertainty, as its value is very low and other processes than the 6to4 relay implementation may also influence the size of free/used memory of the test computer.

The number of forwarded packets per seconds and the CPU utilization are graphically displayed in Fig. 12.

Evaluation of the results:

- Though packet loss occurred for eight or more clients, the packet loss ratio was always very low (under 0.03 percent, which means that 3 packets were lost from 10,000 packets).
- The average response time (given in milliseconds) was low even at 10 clients. Its measure showed nearly linear increase in the function of the load from 4 to 10 clients.
- The number of forwarded packets per second could increase nearly linearly in the function of the number of clients for small number of clients (1-3). It showed less than linear growth for 4-6 clients, and saturation for 7-10 clients, where there was not enough free CPU capacity.
- The CPU utilization showed a linear increase in the function of the number of clients in the case of 1-3 clients (3.0%, 6.1%, 10.1%), as expected. However it showed a radical increase in the case of 4-7 clients (16.9%, 28.1%, 51.0%, 86.4%) which was unexpected and seems to be groundless.
- The memory consumption was always very low (note: it was measured in kB) and it was only slightly increasing in the function of the load with some fluctuations.

To sum up the findings above, we can lay down that the Linux sit 6to4 relay performed quite well, its memory consumption was found to be very low and its average response time increased approximately linearly with the load at high load conditions, that is, it seems to comply with the *graceful degradation* principle [21], however from 4 to 7 clients, the CPU utilization increased higher than linearly in the function of the number of clients. This is a strange phenomenon and it should be investigated before Linux sit 6to4 relay is being actually used in environments with strong response time requirements.

TABLE I. LINUX 6TO4 RELAY PERFORMANCE RESULTS USING TWO RESPONDERS

1	Number of Clients		1	2	3	4	5	6	7	8	9	10
2	Packet loss (%)		0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.03	0.03
3	Response Time of ping6 (ms)	Average	0.274	0.308	0.241	0.480	0.576	0.7	0.867	0.998	1.168	1.311
4		Std. dev.	0.017	0.029	0.053	0.082	0.105	0.126	0.148	0.176	0.192	0.196
5	CPU Utilization (%)	Average	3.0	6.1	10.1	16.9	28.1	51.0	84.6	94.1	92.5	94.3
6		Std. dev.	1.0	1.3	2.5	5.6	10.3	22.2	23.8	13.5	12.8	8.9
7	Traffic Volume (packets/sec)		11177	21360	29424	35166	39829	42615	43502	45411	45691	46812
8	Memory Consumption (kB)		72	96	120	124	148	124	108	164	212	292

TABLE II. FREEBSD 6TO4 RELAY PERFORMANCE RESULTS USING TWO RESPONDERS

1	Number of Clients		1	2	3	4	5	6	7	8	9	10
2	Packet loss (%)		0.02	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	Response Time of ping6 (ms)	Average	0.491	0.936	1.454	2.050	2.635	3.234	3.866	4.456	5.059	5.629
4		Std. dev.	0.034	0.093	0.143	0.179	0.181	0.139	0.144	0.115	0.156	0.178
5	CPU Utilization (%)	Average	69.0	91.4	97.8	98.7	99.7	100.0	99.9	100.0	100.0	100.0
6		Std. dev.	3.9	2.9	1.4	1.1	0.6	0.0	0.4	0.0	0.0	0.0
7	Traffic Volume (packets/sec)		8900	11916	12873	13018	13179	13248	13176	13212	13214	13288
8	Memory Consumption (kB)		536	892	1548	892	892	1300	2320	2060	2012	2648

TABLE III. NETBSD 6TO4 RELAY PERFORMANCE RESULTS USING TWO RESPONDERS

1	Number of Clients		1	2	3	4	5	6	7	8	9	10
2	Packet loss (%)		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	Response Time of ping6 (ms)	Average	2.191	2.270	2.668	2.514	2.543	3.033	2.786	3.320	3.267	3.398
4		Std. dev.	0.574	0.628	0.809	0.718	0.719	0.768	0.686	0.634	0.617	0.557
5	CPU Utilization (%)	Average	9.9	18.9	25.0	34.9	42.9	44.5	55.8	55.0	62.5	67.4
6		Std. dev.	2.0	4.0	6.3	7.9	9.5	10.3	11.6	10.1	11.3	10.7
7	Traffic Volume (packets/sec)		3068	5958	7836	11006	13628	14021	17644	17241	19721	21079
8	Memory Consumption (kB)		936	144	156	160	160	176	240	228	236	592

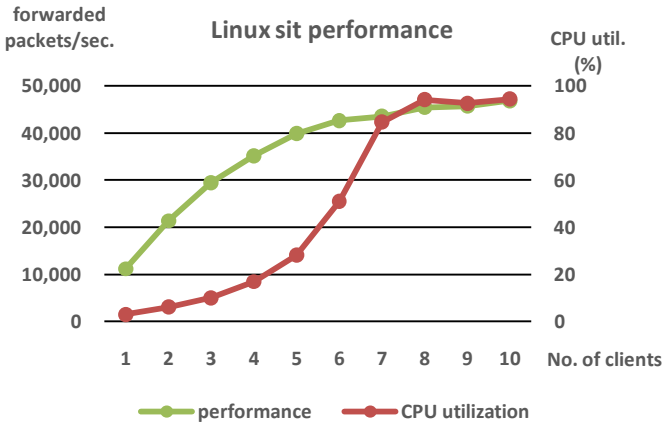

Fig. 12. Linux sit performance and CPU utilization

B. Linux sit tunnel with one responder

The aim of this series of measurements was to see if there are significant differences in the results compared to the case with two responders therefore the measurements were taken only for certain number of clients (1, 4, 5, 8, 10). The comparison can be seen in Fig. 13. The results of the measurements with the one responder were very close to that of the measurements with two responders. For this reason, we do not include measurement results with one responder for the other two implementations.

C. FreeBSD stf tunnel with two responders

The results can be found in Table 2. The number of forwarded packets per seconds and the CPU utilization are graphically displayed in Fig. 14. Evaluation of the results:

- Though packet loss occurred for 1-3 clients, but the packet loss ratio was always very low (under 0.02 percent, which means that 2 packets were lost from 10,000 packets).
- The average response time (given in milliseconds) was acceptable even at 10 clients. Its measure showed nearly linear increase in the function of the load.
- The number of forwarded packets per second in the function of the number of clients showed saturation from 3 clients because there was not enough free CPU capacity.
- The CPU was practically fully utilized from 3 clients.
- The memory consumption was always low (note: it was measured in kB) and it was only slightly increasing in the function of the load with some fluctuations.

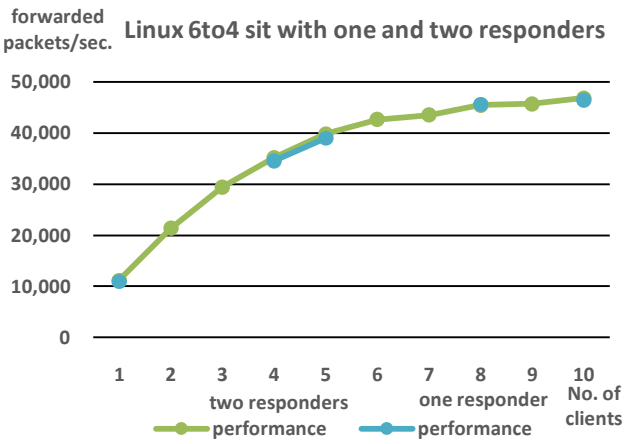

Fig. 13. Comparison of the Linux sit performance with one and two responders.

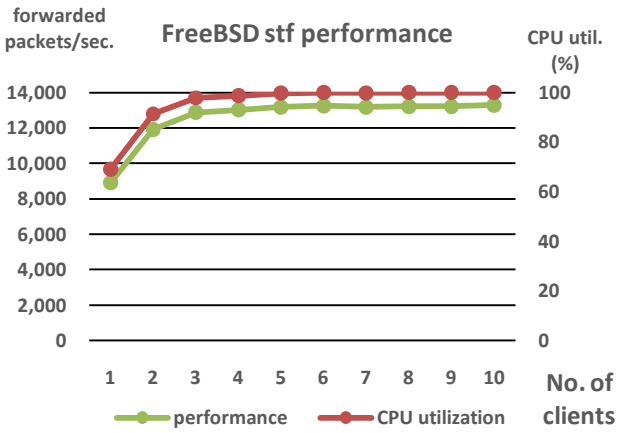

Fig. 14. FreeBSD stf performance and CPU utilization

To sum up the findings above, we can lay down that the FreeBSD stf 6to4 relay performed quite well, its memory consumption was found to be low and its average response time increased approximately linearly with the load, that is, it complies with the *graceful degradation* principle [21].

D. NetBSD stf tunnel with two responders

The results can be found in Table 3. Evaluation of the results:

- The packet loss ratio was always less than 0.005% (thus it was rounded to 0.00%).
- The average response time (given in milliseconds) was acceptable even at 10 clients. Its measure showed only a very slight increase in the function of the load.
- The number of forwarded packets per second in the function of the number of clients did not show saturation even in the case of 10 clients.
- The CPU was not fully utilized even in the case of 10 clients.
- The memory consumption was always low (note: it was measured in kB), but is showed serious fluctuations.

To sum up the findings above, we can lay down that the NetBSD stf 6to4 relay performed quite well, its memory consumption was found to be low and its average response time increased less then linearly with the load. As the CPU was never fully utilized we can state only that as far as we could test it, NetBSD stf complied with the *graceful degradation* principle [21].

E. Comparison of the Resuls and Final Evaluation

As for the number of forwarded packets per second, Linux sit showed much better performance than FreeBSD/NetBSD stf at any investigated load conditions. As for 10 clients, Linux sit processed 46812/13288=3.52 times more packets per seconds than FreeBSD stf and 46812/21079=2.22 times more packets per seconds than NetBSD stf. As for the average response time, Linux sit was also much better than FreeBSD/NetBSD stf. The average response times of Linux sit, FreeBSD stf and NetBSD stf at 10 clients were: 1.3ms, 5.6ms and 3.4ms, respectively. Thus we can say that as for their measured performance, Linux sit seems to be much better than FreeBSD/NetBSD stf. However, Linux sit showed the phenomenon of a super linear CPU consumption in the function of the load at certain conditions. This makes us cautious and we advise to test Linux sit further before using it in mission critical environments with strong response time requirements.

The comparison of the performance results of the FreeBSD stf and of the NetBSD stf is also very interesting. On the one hand, FreeBSD stf performed much better than NetBSD stf both in the number of packets processed (FreeBSD: 8900, NetBSD: 3068) and in the average response time (FreeBSD: 0.5ms, NetBSD: 2.2ms) with one client. Note that its CPU utilization was much higher, too. On the other hand, NetBSD stf performed significantly better than FreeBSD stf both in the number of packets processed (NetBSD: 21079, FreeBSD: 13288) and in the average response time (NetBSD: 3.4, FreeBSD: 5.6) with ten clients. NetBSD had even free CPU capacity. Therefore if one prefers the BSD platform (e.g. for security reasons) FreeBSD can be a good choice if low traffic is expected (and ensured) because of its shorter response time. If high traffic is expected (or if there is a possibility of high traffic), than the more robust NetBSD is our recommendation.

VIII. Discussion Of The Validity Of The Results And Future Work

As 6to4 operates at network level and IP can carry the data units of both TCP and UDP, no care was taken to the transport layer protocol or the type of applications. This approach can be justified by the fact that 6to4 operates on the basis of the IP header and does not take care to the payload of the datagram. The payload was actually ICMPv6, because it was simple to generate by using the `ping6` Unix command.

On the one hand, an IP datagram is just like another, thus our results characterize the investigated 6to4 relay implementations in general with no regard to the transport layer protocol, applications or even network topology.

On the other hand, the length of the used ICMPv6 `echo request / echo reply` packets was 100 bytes, which is quite short, thus our tests were a kind of worst case tests for the implementations in the sense that much more payload bytes may be carried by the same amount of work of the 6to4 relay when using longer packets. (Our purpose was to provide the highest possible workload for the tested 6to4 relay implementations to be able to test their behavior under serious overload situations.)

As real life applications use longer packets up to 1500 bytes – the MTU (Maximum Transmission Unit) size of the Ethernet link layer protocol – we plan to perform our tests using different (higher) packet sizes, too.

It is also our plan to check the possible interference between the 6to4 relay implementations and the transport layer protocols. Unlike ICMP and UDP, TCP provides two reliable streams (one per direction) between the endpoints. Therefore TCP resends the lost packets and thus generates higher load for unreliable channels. The packet loss ratio was low in our experiments and thus we do not expect significant differences, but we plan to perform the tests both using UDP and TCP, too.

IX. Conclusions

The 6to4 IPv6 transition method was introduced. Linux sit, FreeBSD stf and NetBSD stf 6to4 relay implementations were selected for performance and stability testing. The test environment and the measurement method were described.

It was found that Linux sit gave the best performance results in both the number of forwarded packets per second and the average response time under all investigated load conditions. However Linux sit showed super linear CPU consumption in the function of the load under certain conditions. Therefore we advise systems administrators to be cautious and to test Linux sit further before using it in mission critical environments with strong response time requirements.

Within the BSD platform, which can be a choice for security reasons, FreeBSD gave shorter response time at low load conditions and NetBSD could process more packets per second at high load conditions.

References

[1] S. Deering and R. Hinden, “Internet protocol, version 6 (IPv6) specification”, IETF, December 1998. (RFC 2460)

[2] Google, “IPv6 statistics”, http://www.google.com/ipv6/statistics.html

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

[4] 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)

[5] 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)

[6] B. Carpenter, K. Moore, “Connection of IPv6 domains via IPv4 clouds”, IETF, February 2001. (RFC 3056)

[7] 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

[8] 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

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

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

[11] 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

[12] 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.

[13] G. Lencse and S. Répás, "Performance analysis and comparison of the TAYGA and of the PF NAT64 implementations" *Proc. 36th International Conference on Telecommunications and Signal Processing* (TSP-2013, July 2-4, 2013) Rome, Italy, pp. 71-76.

[14] P. Wu, Y. Cui, J. Wu, J. Liu, and C. Metz, “Transition from IPv4 to IPv6: a state-of-the-art survey” *Communications Surveys & Tutorials*, IEEE, vol. 15, no. 3, 2013, pp. 1407 - 1424

[15] E. Karpilovsky, A. Gerber, D. Pei, J. Rexford, and A. Shaikh A, “Quantifying the extent of IPv6 deployment,” in *Passive and Active Network Measurement*. vol. 5448/2009: Springer Berlin/Heidelberg, 2009, pp. 13-22.

[16] C. Huitema, “Teredo: tunneling IPv6 over UDP through network address translations (NATs)”, IETF, February 2006. (RFC 4380)

[17] N. Bahaman, E. Hamid, A.S Prabuwono, “Network performance evaluation of 6to4 tunneling” *Proc. 2012 International Conference on Innovation Management and Technology Research* (ICIMTR, May 21-22, 2012) Malacca, Malaysia, pp. 263-268

[18] Free Software Fundation, “The free software definition”, http://www.gnu.org/philosophy/free-sw.en.html

[19] Open Source Initiative, “The open source definition”, http://opensource.org/docs/osd

[20] E. Normark, R. Gillian, “Basic transition mechanisms for IPv6 hosts and routers”, IETF, October 2005. (RFC 4213)

[21] NTIA ITS, “Definition of ‘graceful degradation’” http://www.its.bldrdoc.gov/fs-1037/dir-017/_2479.htm

[22] Cisco, “End user licence agreement”, http://www.cisco.com/en/US/docs/general/warranty/English/EU1KEN_.html

[23] Juniper Networks, “End user licence agreement”, http://www.juniper.net/support/eula.html

[24] Federico Biancuzzi, “The man in the machine”, http://www.securityfocus.com/columnists/459