Acta
Technica
Jaurinensis

Vol. 7., No. 4., pp. 404-427, 2014
DOI: 10.14513/actatechjaur.v7.n4.340
Available online at acta.sze.hu

Performance and Stability Analysis of Free NAT64 Implementations with Different Protocols

S. Répás, P. Farnadi, G. Lencse

Széchenyi István University, Department of Telecommunications
Egyetem tér 1, H-9026 Győr, Hungary
E-mail: repas.sandor@sze.hu, farnadi@tilb.sze.hu, lencse@sze.hu

Abstract: Due to its rapid spread, Internet has now outgrown the address space provided by IPv4. The transition to the new IPv6 protocol is extremely slow. The different transition techniques are intended to facilitate the transition to the new version of Internet Protocol. The NAT64 transition technique is one of the most suitable solutions. In this paper, both the performance and the stability of two NAT64 gateway implementations are examined by ICMP, TCP and UDP protocols. The tested two free implementations, the TAYGA on the Linux system and the PF of the OpenBSD system can be effectively used in a production environment, and can facilitate the deployment of the IPv6 protocol.

Keywords: NAT64, TAYGA, PF, performance, stability

1. Introduction

Due to the exhaustion of the IPv4 address pool [1], the internet service providers will not be able to provide IPv4 addresses to their customers in the near future. The application of the new version of the Internet Protocol, the IPv6 can provide practically infinite number of addresses for the huge number of Internet-enabled devices. The specification of the IPv6 protocol exists since 1998 [2], but the widespread application of it is still pending. While the IPv6 protocol can solve the address exhaustion problem, the application of the new protocol raises another problem. Because of the different IP header structure and addressing scheme, the direct communication between an IPv4 and IPv6 hosts is impossible. The most important hindering factor of the rapid deployment of the IPv6 protocol is the combination of this incompatibility and the huge number of the installed IPv4 only devices. To solve this problem and speed up the IPv6 widespread implementation one can use the so called IPv6 transition techniques. These techniques enable the communication between hosts in a mixed IPv4 and IPv6 network. Over the years several transition techniques have been developed. According to the authors' opinion, the combination of a DNS64 [3] server and a NAT64 [4] gateway is currently one of the most useful techniques to speed up the IPv6 deployment. The appropriate choice of high performance and very stable DNS64 and NAT64 implementations can be crucial for the service providers. This paper deals with the stability and performance of two different open source NAT64 implementations with the three most important protocols over: ICMP, UDP and TCP.

The remainder of this paper is organized as follows: first, the operation of the DNS64+NAT64 solution is described, second, TAYGA under Linux and Packet Filter of OpenBSD are introduced, third, the NAT64 performance and stability research results are surveyed, fourth, the description of the test network and the testing method of each protocols are given, fifth, the test results are introduced, sixth, our results are summarized and discussed, and finally, our conclusions are given.

2. The DNS64 and NAT64 transition techniques

In the current phase of the IPv6 deployment it is a very important task to provide connectivity between an IPv6 only client and an IPv4 only server. To solve this problem one can use the DNS64+NAT64 combination. DNS64 is an extension of the DNS server, and NAT64 is similar to the "normal" Network Address Translation [5] process, but with address family translation. The operation of DNS64+NAT64 is shown in Fig. 1.

The explanation of the communication process is the following:

- Step 1: The IPv6 only client sends a query to its name server about the IPv6 address of the destination server with the DNS name of the server (query the AAAA record [6] of the destined server).
- Step 2: The DNS64 server tries to resolve the DNS name.
- Step 3:
  - If the DNS server resolves the given name to an IPv4 address, but not IPv6: The DNS64 server generates an answer with an AAAA record with a synthesized IPv6 address. This IPv6 address contains the given IPv4 address of the server at the last 32 bits, while the first 96 bits can be a network specific prefix or the NAT64 well-known prefix. This special IPv6 address is called IPv4-Embedded IPv6 Address.
  - If the DNS server could resolve the given name to an IPv6 address (the given name has an AAAA resource record) then the DNS64 server acts as normal. (This case is not shown in the figure.)
- Step 4: The DNS64 server sends back as an answer for the query of its client.
- Step 5: The client sends the IPv6 packet through its (NAT64) gateway.
- Step 6:
  - If the destination address of the packet contains the special prefix, the gateway knows that the IPv6 packet is destined to an IPv4 server. The NAT64 gateway makes a stateful network address and address family translation between IPv6 and IPv4 and sends the resulted IPv4 packet to the IPv4 only server with its own public IPv4 address as source address. The NAT64 gateway needs to have both IPv4 and IPv6 addresses, to make this process happen.

   - If the destination address of the packet does not contain the prefix, the gateway forward the packet normally to its next hop. (This case is not shown in the figure.)

- Step 7: The IPv4 only server sends its answer to the IPv4 address of the NAT64 gateway.
- Step 8: The NAT64 gateway assembles the IPv6 version of the received packet using the payload of the IPv4 packet and its own state table and then sends the IPv6 packet to the originating client.

The whole process is intended to be transparent for the client computer. This IPv6 transition solution is compatible with the majority of the wide spread application layer protocols that work in client-server model (e.g. HTTP, SMTP, POP3, IMAP4, SSH, etc.) but it has issues with those protocols that transfer IP addresses (e.g. FTP, SIP) and usually does not work with peer-to-peer applications (e.g. BitTorrent), see: [7], [8] and [9].

For a more detailed but still easy to follow introduction to DNS64+NAT64, see [10] and for the most accurate and detailed information, see the relating RFCs [3] and [4].

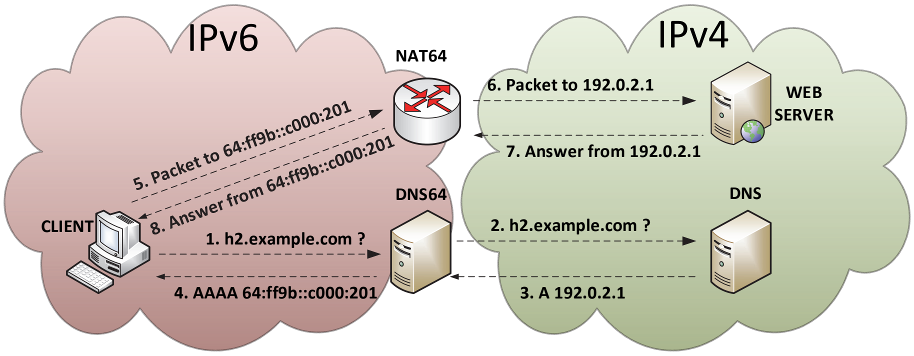

Figure 1. Operation of DNS64+NAT64 (Based on: [11])

3. The examined NAT64 implementations

3.1. TAYGA

TAYGA [12] is a free stateless NAT64 implementation for Linux under GPLv2 license. The main goal of its developers was to provide a production quality NAT64 service where a dedicated NAT64 device would be overkill. The latest release of TAYGA is 0.9.2. According to its authors, TAYGA could never come close to offering the power and flexibility available in the packet filter of Linux (iptables), so instead TAYGA turns IPv6 into IPv4 in the most transparent manner possible, allowing existing IPv4-only tools to be used to further manipulate sessions flowing through it. 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 the stateful NAT44 packet filter performs an SNAT (Source Network Address Translation) from the private IPv4 addresses to the public IPv4 address of the NAT64 gateway. 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. To work with TAYGA, a suitably large private IPv4 dynamic address pool should be provided.

Packet Filter, PF

The Packet Filter was introduced in the OpenBSD system in 2001. PF is a very popular firewall application in the BSD systems. The PF of OpenBSD supports stateful and stateless mode at the same time. It supports various types of packet manipulation, and it supports IPv4 and IPv6 stateful NAT for many years. Since OpenBSD 5.1 it supports stateful NAT64, too [13]. Stateful behaviour means that it does not need the help of a stateful NAT44 packet filter to work as a complete NAT64 gateway. This nature of PF can speed up the NAT64 translation.

4. A Short Survey of the Current Research Results

Though NAT64 is addressed in high number of current research papers, only a relatively few of them deals with the performance analysis of its different implementations. For example, [14] gives a good summary of the NAT64 papers until 2012 and it states that "two papers were found that deal with NAT64 performance issues". And also many of those that deal with the performance of NAT64 implementations do it in the way that they examine the performance of a given NAT64 implementation together with a given DNS64 implementation. For instance, they measure the performance of the TAYGA NAT64 implementation with the TOTD DNS64 implementation in [15]. The authors of two other papers [16] and [17] measure the performance of the Ecdysis NAT64 implementation, which uses its own DNS64 implementation. However, we have already shown that as DNS64 and NAT64 are two distinct services, they may be and thus should be analyzed separately to be able to choose the best suiting implementations for someone's purposes [18]. We have published our first results on the separate performance analysis of the TAYGA NAT64 implementation and of the BIND DNS64 implementation in [19]. Later on, we compared the performance of the BIND and TOTD DNS64 implementations under Linux, OpenBSD and FreeBSD in [20]. We tested the stability and the performance of the TAYGA and of the PF NAT64 implementations using ICMP in [18]. The aim of our current research it to test their stability and compare their performance using also TCP and UDP protocols, as they are used over IP in real life applications.

5. The test environment for the NAT64 performance measurements

5.1. The topology of the test network

The test network was built up in the Infocommunications Laboratory of the Department of Telecommunications, Széchenyi István University. The topology of the network is shown in Fig. 2. All of the network elements were connected with 1000Base-TX network connections. For this purpose, a 3Com Baseline 2948-SFP switch was used. The responder computer can be seen on the top of the figure. The role of the computer was to answer all of the measurement traffic destined to it. For this reason, a high performance workstation computer was used for this role. The central element of the network is the NAT64 server. This gateway was built from the lowest performance computer in the laboratory, because serious overload situation was necessary during the measurement process. In the bottom of the picture eight pieces of high performance test clients are shown. These workstation computers played the role of the high number of clients during the different test scenarios.

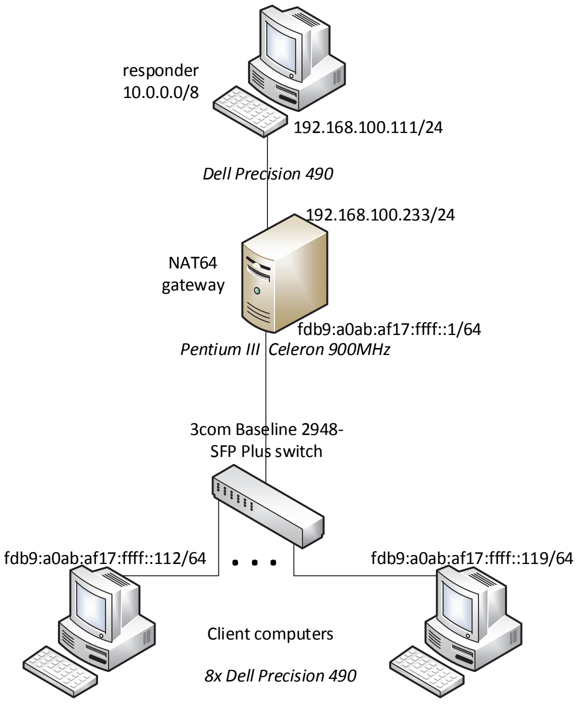

Figure 2. Topology of the test network

5.2. The hardware configuration of the computers

The configuration of the responder computer was the following:

- DELL 0GU083, Intel 5000X chipset mainboard
- Two Dual Core Intel(R) Xeon(R) CPU 5160 3.00GHz dual core microprocessors
- 4x1 GB 533 MHz DDR2 SDRAM (quad channel)
- Broadcom NetXtreme BCM5752 Gigabit Ethernet PCI Express network card
- Debian 6.0.7

- KDE 4.4.5 graphic environment

The configuration of the NAT64 gateway computer was the following:

- Intel D815EEA2 mainboard
- 900 Intel Pentium III (Coppermine) microprocessor
- 256 MB, 133 MHz SDRAM
- Two 3Com 3c940 Gigabit Ethernet PCI network cards

The configuration of all of the client computers was the following:

- DELL 0GU083, Intel 5000X chipset mainboard
- Two Dual Core Intel(R) Xeon(R) CPU 5140 2.33GHz dual core microprocessors
- 4x1 GB 533 MHz DDR2 SDRAM (quad channel)
- Broadcom NetXtreme BCM5752 Gigabit Ethernet PCI Express network card
- Debian 6.0.7

5.3. The software configuration of the computers

The NAT64 gateway performance was monitored with the commands as seen in Figures 3 and 4. The settings of the network interfaces are shown in Figures 5, 6 and 7.

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

Figure 3. NAT64 gateway computer performance monitoring on Linux system

```
vmstat –w 1 > load.txt
```

Figure 4. NAT64 gateway computer performance monitoring on OpenBSD system

The settings of the TAYGA on Linux are shown in Fig. 8. The starting of TAYGA was automatized with a shell script. The script is shown in Fig. 9. The NAT64 function was enabled on the OpenBSD system in a modification in the `/etc/pf.conf` file as shown in Fig. 10.

```
#Internal
auto eth0
iface eth0 inet6 static
address fdb9:a0ab:af17:ffff::1
netmask 64
up sleep 1
up echo 0 > /proc/sys/net/ipv6/conf/eth1/autoconf
up echo 0 > /proc/sys/net/ipv6/conf/eth1/accept_ra
post-up sleep 1
post-up /root/nat64-config.sh

#External
auto eth1
iface eth1 inet static
address 192.16.100.233
netmask 255.255.255.240
gateway 192.168.100.1

pre-up echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
```

Figure 5. Network interface settings in the `/etc/network/interfaces` file on the Linux system

```
inet 192.168.100.233 255.255.255.0
!route add –inet 10.0.0.0/8 192.168.100.111
```

Figure 6. External network interface settings in the `/etc/hostname.sk0` file on the OpenBSD system

```
inet6 fdb9:a0ab:af17:ffff::1 64
```

Figure 7. Internal network interface settings in the `/etc/hostname.sk1` file on the OpenBSD system

```
tun-device nat64
ipv4-addr 172.16.0.1
prefix fdb9:a0ab:af17:ffff:ffff:ffff::/96
dynamic-pool 172.16.0.0/12
data-dir /var/db/tayga
```

Figure 8. TAYGA settings in the `/usr/local/etc/tayga.conf` file

On the responder computer, all of the IP traffic destined to the 10.0.0.0/8 network were redirected to the local address of the Ethernet interface of the computer with iptables command as seen on Fig. 11.

```
#!/bin/bash
tayga --mktun
ip link set nat64 up

#Private IPv4 address space for stateless NAT
ip addr add 172.16.0.1 dev nat64
ip route add 172.16.0.0/12 dev nat64

#For NAT64 unique local IPv6 address space
ip addr add fdb9:a0ab:af17:ffff::2 dev nat64
ip route add fdb9:a0ab:af17:ffff:ffff:ffff::/96 dev nat64
tayga

#Enable packet forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding

#Enable stateful NAT44
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

#The used addresses are routed to the responder
ip route add 10.0.0.0/8 via 192.168.100.111
```

Figure 9. The `nat64-config.sh` shell script on Linux system

```
set limit states 40000
pass in on sk1 inet6 from any to fdb9:a0ab:af17:ffff:ffff:ffff::/96 \
  af-to inet from 192.168.10.233
```

Figure 10. The `/etc/pf.conf` settings on OpenBSD system

```
iptables -t nat -A PREROUTING -d 10.0.0.0/8 -j DNAT \
  --to-destination 192.168.100.111
```

Figure 11. iptables settings on the responder computer

6. The measurement process and scripts

6.1. ICMP

Responder

During the preliminary measurements, some fluctuations were experienced in the load of the NAT64 gateway. This behavior was caused by the limited size of the connection tracking table on the responder computer. This problem could be solved with two methods. First, the size of the table could be drastically increased. Second, the timeout value of a record in the table could be decrease to a low value. For the measurements, the second solution was chosen. It is shown on Figure 12.

```
echo 1 > /proc/sys/net/netfilter/nf_conntrack_icmp_timeout
```

Figure 12. Setting the timeout value of ICMP packets to 1s in the conntrack table

Client computers

All of the measurements were made with 1, 2, 4 and 8 client computers simultaneously. The measurement of the execution time and the starting of the experiment was initiated with the command line on the Fig. 13. The synchronized start of the scripts on the client computers were done by using the "Send Input to All Sessions" function of the Konsole terminal program of the KDE graphical environment on a separated "controller" computer.

```
~# /usr/bin/time –f "%" –o runtime –a ./icmptest.sh experiment_ID
```

Figure 13. Starting the measurement

The test script is shown in figure 14.

```
#!/bin/bash
mkdir $1
i=`cat /etc/hostname | grep -o .$`
for b in {0..255}
do
  mkdir $1/$b
  for c in {0..248..8}
  do
    ping6 -c10 -i0  fdb9:a0ab:af17:ffff:ffff:ffff:10.$i.$b.$c \
      >> $1/$b/nat64p-10-$i-$b-$c &
    ping6 -c10 -i0  fdb9:a0ab:af17:ffff:ffff:ffff:10.$i.$b.$((c+1)) \
      >> $1/$b/nat64p-10-$i-$b-$((c+1)) &
    ping6 -c10 -i0  fdb9:a0ab:af17:ffff:ffff:ffff:10.$i.$b.$((c+2)) \
      >> $1/$b/nat64p-10-$i-$b-$((c+2)) &
    ping6 -c10 -i0  fdb9:a0ab:af17:ffff:ffff:ffff:10.$i.$b.$((c+3)) \
      >> $1/$b/nat64p-10-$i-$b-$((c+3)) &
    ping6 -c10 -i0  fdb9:a0ab:af17:ffff:ffff:ffff:10.$i.$b.$((c+4)) \
      >> $1/$b/nat64p-10-$i-$b-$((c+4)) &
    ping6 -c10 -i0  fdb9:a0ab:af17:ffff:ffff:ffff:10.$i.$b.$((c+5)) \
      >> $1/$b/nat64p-10-$i-$b-$((c+5)) &
    ping6 -c10 -i0  fdb9:a0ab:af17:ffff:ffff:ffff:10.$i.$b.$((c+6)) \
      >> $1/$b/nat64p-10-$i-$b-$((c+6)) &
    ping6 -c10 -i0  fdb9:a0ab:af17:ffff:ffff:ffff:10.$i.$b.$((c+7)) \
      >> $1/$b/nat64p-10-$i-$b-$((c+7))
  done
done
```

Figure 14. icmp-test.sh shell script

Each client sent 256*256*10=655360 ICMP Echo request packets to 256*256=65536 different destination IP addresses during one experiment. The body of the cycle contains 8 ping commands which were started concurrent. With the 8 simultaneous commands, we were able to provide sufficiently large load on the NAT64 gateway.

6.2. TCP

Responder

On the responder computer, the Apache 2 web server was used to respond the queries sent over TCP. The Apache 2 web server was installed from the Debian repository by the `apt-get` command. The preparation of the files with different sizes was made by the dd command. For example to generate a 100 byte long file, the following command line was used:

```
dd if=/dev/zero of=/var/www/file bs=100 count=1
```

The timeout value of the TCP records in the connection tracking table was decreased to 1s by issuing the following command:

```
 echo 1 > /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_time_wait
```

Client computers

In this case the `tcp256.sh` shell script was started synchronously, similarly to the ICMP measurement. The `tcp256.sh` script is shown in Fig. 15.

The `tcp.sh` script was invoked 256 times by the `tcp256.sh` script. The `tcp.sh` script was responsible for downloading the files from the responder computer. The two scripts downloaded altogether 256*256=65536 files from 65536 IP addresses. The `tcp.sh` script is shown in Fig. 16.

```
#!/bin/bash
i=`cat /etc/hostname | grep -o .$`
for b in {0..255}
do
    mkdir $1/$b
    /usr/bin/time -f "%e" -o output.txt -a ./tcp.sh $i $b $1
done
```

Figure 15. `tcp256.sh` shell script

```
#!/bin/bash
for c in {0..255}
do
  wget -m http://[fdb9:a0ab:af17:ffff:ffff:ffff:10.$1.$2.$c]/file \
    -P $3/$2
done
```

Figure 16. `tcp.sh` shell script

6.3. UDP

6.4. Network

For the UDP measurements, the OpenVPN software with UDP VPN tunnel was selected. The logical network diagram is shown in Fig. 17. The Responder computer played the role of the OpenVPN server, while all of the Client computers were the clients of the OpenVPN server.

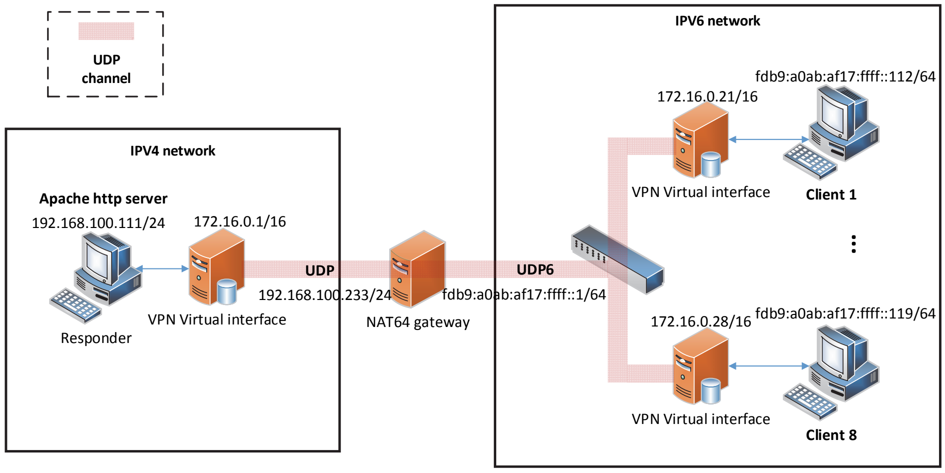

Figure 17. Network diagram of the UDP measurement

6.5. Responder

The OpenVPN server was installed from the Debian repository by the `apt-get` command. For the working OpenVPN environment, the following tasks were performed:

- Step 1: Editing the `/usr/share/doc/openvpn/examples/easy-rsa/2.0/vars` file, according to Fig. 18.
- Step 2: Creation of the `/etc/openvpn/keys` directory with the `mkdir` command.
- Step 3: Copying the `openssl.cnf`, `whochopensslcnf`, and `pkitool` files from the `/usr/share/doc/openvpn/examples/easy-rsa/2.0/` to the newly created `/etc/openvpn/keys` directory by `cp` command.
- Step 4: Building the certificate authority by issuing the commands on Fig. 19.
- Step 5: Creating the certificates by issuing the commands on Fig. 20.
- Step 6: Copying the `ca.crt`, `dh1024.pem`, `server.crt` and `server.key` files to the `/etc/openvpn/` directory.
- Step 7: Extraction of the `/usr/share/doc/openvpn/examples/sample-configfiles/server.conf.gz` file into the `/etc/openvpn` directory and modifying the `server.conf`, according to Fig. 21.

```
export KEY_COUNTRY=”HU”
export KEY_PPROVINCE=”GYMS”
export KEY_CITY=”GYOR”
export KEY_ORG=”TILB”
export KEY_EMAIL=openvpnadmin@tilb.sze.hu
```

Figure 18. Modification of the `/usr/share/doc/openvpn/examples/easy-rsa/2.0/vars` *file*

```
. /usr/share/doc/openvpn/examples/easy-rsa/2.0/vars
. /usr/share/doc/openvpn/examples/easy-rsa/2.0/clean-all
. /usr/share/doc/openvpn/examples/easy-rsa/2.0/build-ca
. /usr/share/doc/openvpn/examples/easy-rsa/2.0/build-dh
```

Figure XVIX. Creating the Certificate Authority

```
cd /usr/share/doc/openvpn/examples/easy-rsa/2.0/
./build-key-server server
./build-key client
```

Figure 20. Creating the certificates

```
proto udp
dev tun
server 172.16.0.0 255.255.0.0
duplicate cn
group
nogroup
```

Figure 21. The modifications of the `/etc/openvpn/server.conf` *file*

6.6. Clients

The OpenVPN clients were installed from the Debian repository by the `apt-get` command. For the working OpenVPN environment, the following tasks were performed:

- Step 1: Copying the `client.crt`, `ca.crt`, `client.key` files from the `/etc/openvpn/keys/` directory of the responder computer into the `/etc/openvpn/` directory of the client computers.
- Step 2: Copying the `/usr/share/doc/openvpn/examples/sample-configfiles/client.conf` file into the `/etc/openvpn/` directory and modifying it, according to Fig. 22.

```
dev tun
proto udp6
remote fdb9:a0ab:af17:ffff:ffff:ffff:192.168.100.111
```

Figure 22. The modifications of the `/etc/openvpn/client.conf` *file*

In this case the `udp256.sh` shell script was started synchronously on the client computers, similarly to the ICMP and TCP measurements. The `udp256.sh` script is shown in Fig. 23.

The `udp.sh` script was invoked 64 times by the `udp256.sh` script. The `udp.sh` script was responsible the downloading of the files from the responder computer, see Fig. 24. The core of the "`for`" cycle downloaded four files parallel. The two scripts downloaded altogether 64*256=16384 files from the same IP address.

```
#!/bin/bash
mkdir $1
for b in {0..63}
do
  /usr/bin/time -f "%e" -o output.txt -a ./udp.sh $b $1
done
```

Figure 23. udp256.sh shell script

```
#!/bin/bash
for c in {0..252..4}
do
  wget http://172.16.0.1/file -P $2/$1/$c &
  wget http://172.16.0.1/file -P $2/$1/$((c+1)) &
  wget http://172.16.0.1/file -P $2/$1/$((c+2)) &
  wget http://172.16.0.1/file -P $2/$1/$((c+3))
done
```

Figure 24. udp.sh shell subscript

7. NAT64 performance results

7.1. ICMP

TAYGA

The results can be found in Table 1. Row 1 shows the number of clients that executed the test script. (The load of the NAT64 gateway was proportional to 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. The last row shows the number of forwarded packets per seconds.

Table 1. TAYGA, ICMP NAT64 performance

1	***Number of clients***		***1***	***2***	***4***	***8***
2	***Packet loss (%)***		0.002	0.003	0.005	0.007
3	***Response time (ms)***	***average***	1.67	3.65	7.42	28.47
4		***std. deviation***	0.56	0.94	1.70	9.21
5		***maximum***	30.00	34.70	52.40	87.24
6	***CPU utilization (%)***	***average***	88.89	95.64	99.55	100.00
7		***std. deviation***	2.16	1.77	0.89	0.00
8	***Traffic volume (packets/s)***		3825	3928	4097	4128

Evaluation of the results:

- The packet loss ratio was slightly increased with higher traffic. With one client the NAT64 gateway produced 0.002% packet loss, whereas with 8 clients it was 0.007% packet loss.
- The CPU utilization was very high on the gateway even with one client. With 2 and 4 clients the behavior of the system remained predictable and the averages of the response times were only slightly more than doubled (3.65/1.67=2.19; 7.42/3.65=2.03). With 8 clients, the system remained stable, but the average CPU usage reached the 100%, while the average of the response times was almost four fold (28.47/7.42=3.84).
- The number of served queries were increased while the gateway computer had free CPU capacity. When the CPU usage reached the 100%, the response time increased unexpectedly.

Packet Filter

The results can be found in Table 2. Evaluation of the results:

- The packet loss rate was always very low.
- Due to the doubling of the load on the NAT64 gateway, the response time ratios were as follows: 0.7/0.48=1.45; 1.43/0.7=2.04; 3.19/1.42=2.23. The CPU utilization increased with more clients, but the system remained stable in the serious overload situation with 8 clients.
- The number of served queries increased while the gateway computer had free CPU capacity. From 1 to 2 clients, there was 64.13% increase in the number of answered queries, whereas from 2 to 4 clients there was 22.73% increase. With 8 clients, the NAT64 gateway reached its maximum capacity, the CPU usage was close to 100%, and the number of served queries showed only a slight increase (4.35%).

Table 2. PF, ICMP NAT64 performance

1	***Number of clients***		***1***	***2***	***4***	***8***
2	***Packet loss (%)***		0.001	0.002	0.007	0.008
3	***Response time (ms)***	***average***	0.48	0.70	1.43	3.19
4		***std. deviation***	0.22	0.46	0.77	1.15
5		***maximum***	21.30	30.50	44.00	43.40
6	***CPU utilization (%)***	***average***	49.86	77.23	91.17	96.47
7		***std. deviation***	6.64	3.91	1.82	1.51
8	***Traffic volume (packets/s)***		8536	14010	17195	17944

Comparison of the ICMP results

Comparing the performance results with ICMP packets of TAYGA and PF, we can state the followings:

- With one client, the throughput of PF is significantly higher than that of TAYGA: 8536/3825=223.16%, in addition to that, PF reaches this performance with significantly lower usage of the CPU. With 2, 4 and 8 clients the ratios of the throughputs of the two implementations are: 14010/3928=356.67%; 17195/4097=419.69%; 17944/4128=436.69%. (Fig. 25.)
- A similar phenomenon could be seen with the average response times: 1.67/0.48=3.48; 3.65/0.7=5.21; 7.42/1.42=5.23; 28.47/3.19=8.92 (Fig. 26.).
- In addition, the maximum values of the response times of PF are better, too.

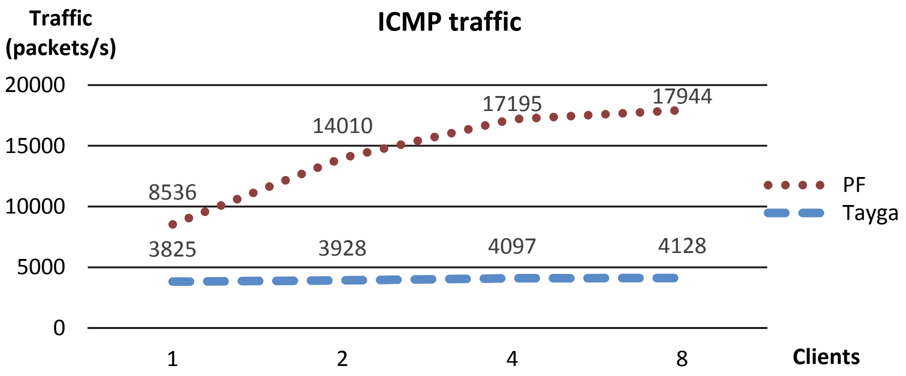

Figure 25. ICMP throughput comparison (higher is better)

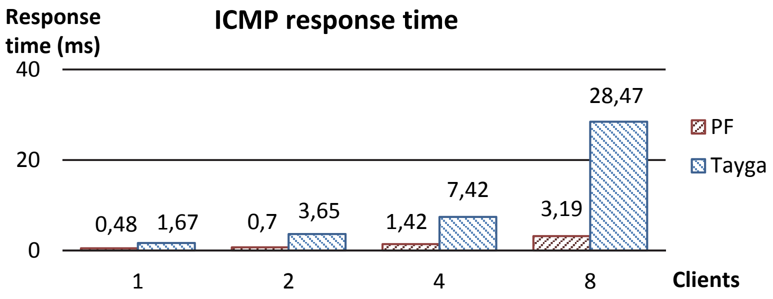

Figure 26. ICMP response time comparison (lower is better)

7.2. TCP

The performance measurements of the two NAT64 gateway implementations with TCP protocol were done by using 8 different file sizes. This method generated 8 times more data. It is more practical to interpret these results with graphs.

TAYGA

The average number of served queries can be seen on Fig. 27. The horizontal axis (x) shows the size of the files. The sizes were doubled during the measurements from 100 to 12800 bytes. The vertical axis (y) shows the throughput of the NAT64 gateway (files/sec), whereas the graph depth (z) corresponds to the number of clients. In the intersection points of different values of the x and z axes, the y values show that how many files with a given size and at a given number of clients the system was able to serve. The average of the response times and the average values of the CPU utilization are shown on Figures 28 and 29, respectively. Evaluation of the results:

- With one client, the CPU utilization is strictly increasing from 55% to 75% in the 100 – 12800 bytes file size range. The system was able to serve nearly the same number of files of size from 100 to 3200 bytes. Then the curve started to break down slowly, with 6400 bytes it served by 21.51% less and with 12800 bytes it served by 25.8% less than with the previous size (see Fig. 27).
- With two clients, the CPU still had free capacity, thus the response time just slightly increased, the system remained stable. The throughput of the system with 100 bytes files increased by 683/405=68.64% compared to one client. But with 12800 byte files, this difference is only 292/233=25.32%.
- With 4 and 8 clients, the system does not have free capacity, thus the response times are increasing continuously.

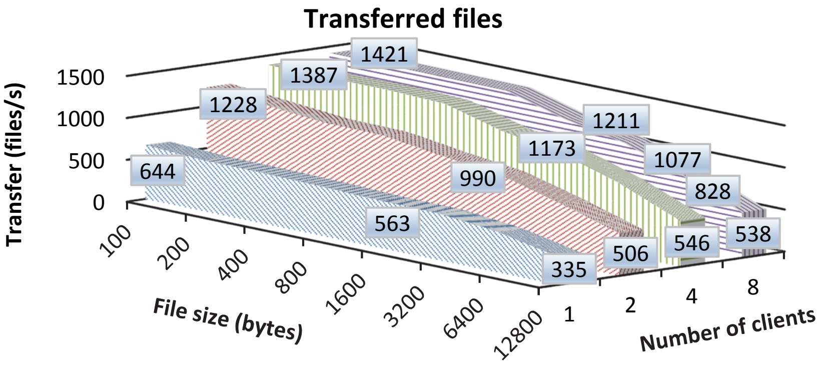

Figure 27. Number of transferred files per second by TAYGA NAT64 gateway with TCP protocol

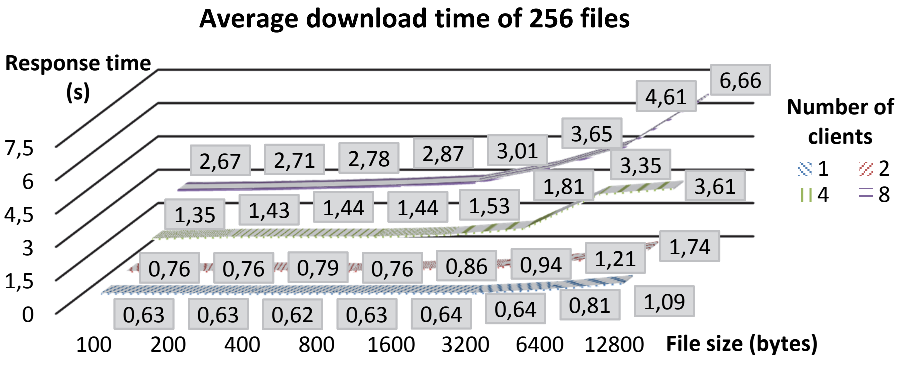

Figure 28. Average download time of 256 files with TAYGA and TCP protocol

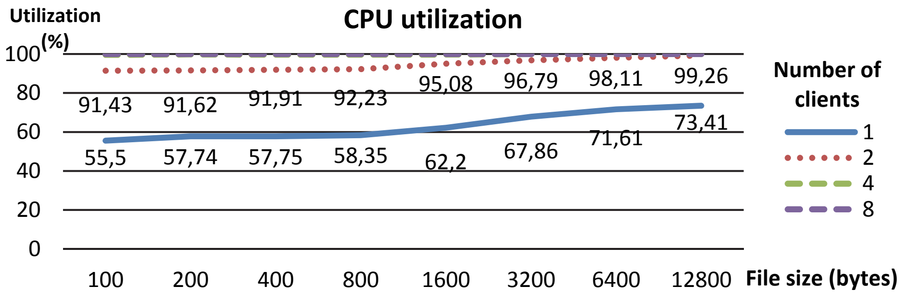

Figure 29. Average CPU utilization with TAYGA

PF

The number of served queries, the average of the download times and of the CPU utilization are shown on Figures 30, 31 and 32, respectively. Evaluation of the results:

- With one client, the utilization of the CPU is 18.82%, and with the maximum file size it is 32.02%. This is the reason of the low value of the average and maximum of the transfer time. With larger size of the transferred files the number of them just slightly decreases.
- With two clients, the CPU utilization is 33.88% in the beginning of the range, whereas it is 57.81% at the end of the range. Thus the average and maximum values of the response times only slightly increase. The throughput of the system is almost doubled compared to one client (~190%).
- With four clients, the CPU utilization is 59.79% at the beginning of the range, whereas it reaches 88.91% at the end of the range. The number of the transferred files strictly decreases from the value of 192% at the beginning of the range until 162% at the end of it – compared to the values of the measurement with two clients.
- With eight clients, the system reaches the end of its capacity. At the beginning of the range it uses 93.04% of its CPU, while transfers about 70% more files compared to the case with 4 clients. With the increasing of the size of the transferred files to 1600 bytes, the CPU utilization increases, too. From this point the system starts to behave unpredictable, the response time sharply increases, while the CPU utilization starts to fluctuate. Thus, the number of the transferred files of the system decreases to 1055 files/seconds at 12800 bytes from 2875 files at 100 bytes.

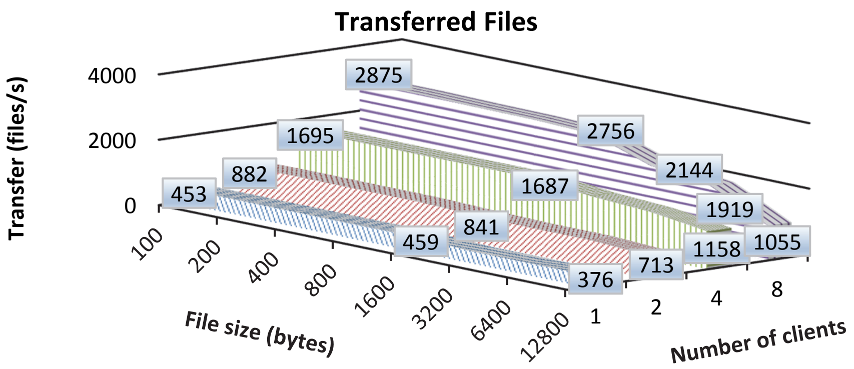

Figure 30. Number of transferred files per second by PF NAT64 gateway with TCP protocol

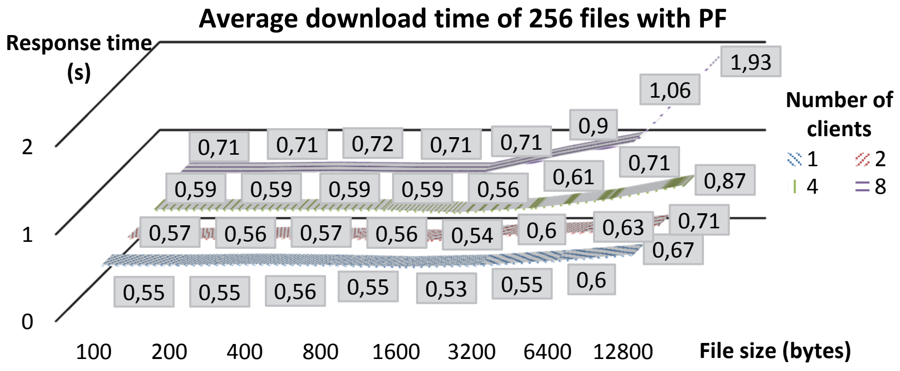

Figure 31. Average download time of 256 files with PF

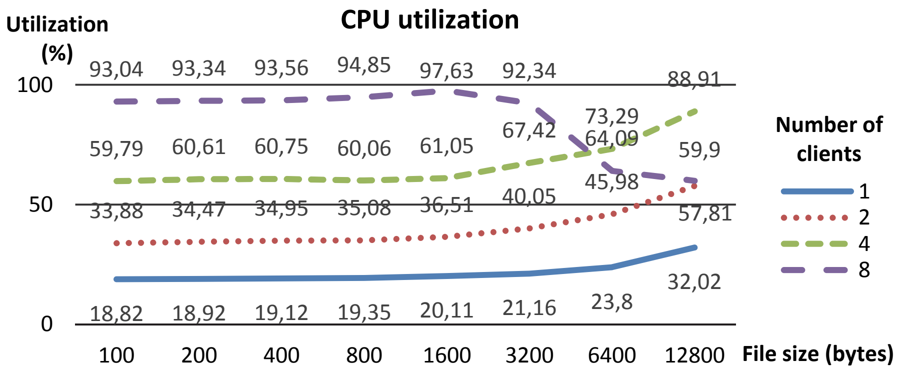

Figure 32. Average CPU utilization with PF

Comparison of the TCP results

Comparing the performance results of TAYGA and PF with transfer via TCP protocol, we can state the followings:

- With one, two and four clients the CPU utilization of PF of OpenBSD was much lower than that of TAYGA on Linux system. To serve one client with 100 bytes files, PF needs 18.82% CPU time, whereas the TAYGA needs 55.5%. Thus the average and maximum values of the transfer times are also better with PF.
- In download time with 12800 bytes files the advantage of the PF is about 3.5 times.
- In serious overload situation, the TAYGA remains stable, where the behavior of the PF is unpredictable but its performance is still higher than that of TAYGA.

7.3. UDP

TAYGA

The number of the transferred files, the average of the download times and of the CPU utilization are shown on Figure 33, 34, 35, respectively. Evaluation of the results:

- With one client, the utilization of the CPU is 46.04% and it starts to increase at 800 bytes long files, and it grows until 70.41% at the end of the range. The average values of the download times doubled to the end of the range. With larger size of the transferred files the number of them slowly decreases from 644 to 335, which means -48%.
- With two clients, the CPU utilization is between 77.85% and 96.28%. Thus the average and maximum values of the response times increase from 1600 bytes. With larger size of the transferred files the number of them slowly decreases from 1228 to 506, which means -59%.
- With four and eight clients, the response times greatly increase compared to the two measurement with two clients. The main cause of this phenomenon is the high processor utilization, which is about 100% at the whole range. With four and eight clients the number of the transferred files decreases with 61% and 62% to the end of the range.

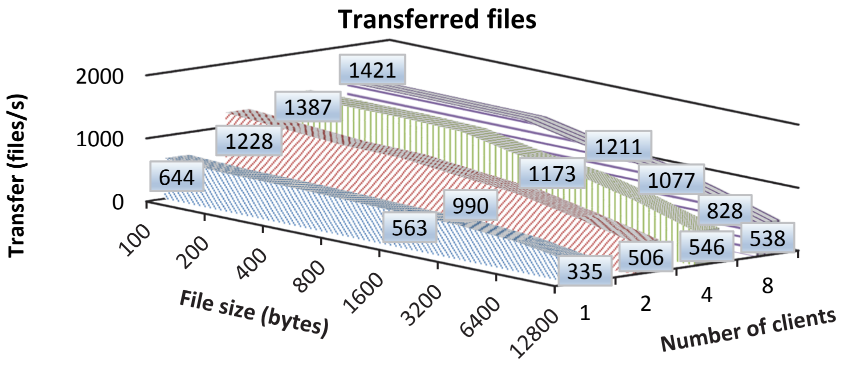

Figure 33. Number of transferred files per second by TAYGA NAT64 gateway with UDP protocol

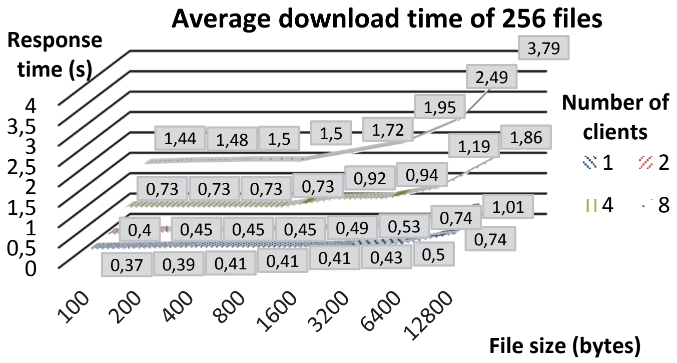

Figure 34. Average download time of 256 files with TAYGA

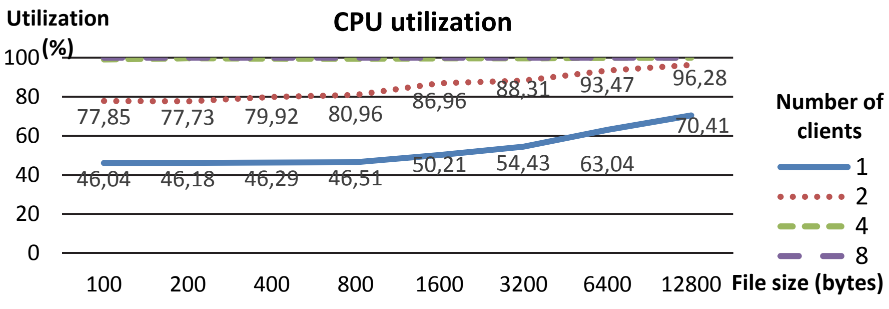

Figure 35. Average CPU utilization with TAYGA

PF

The number of the transferred files, the average of the download times and of the CPU utilization are shown on Figure 36, 37, 38, respectively. Evaluation of the results:

- With one client the utilization of the CPU is 19.47% and it starts to increase at 800 bytes long files, until 32.35% at the end of the range. The average values of the response times increase by 46.87% to the end of the range. With larger size of the transferred files the number of them slowly decreases from 747 to 446, which means -40%.
- With two clients, the CPU utilization is between 32.67% and 48.87%. The average values of the response times increase by 47% to the end of the range. The response times increase by 3.13% through 15% at the whole range compared to the one client case, while the number of the transferred files increases by 92.5% through 95%.
- With four clients, the utilization of the CPU starts at 59.41% and starts to increase between 1600 and 3200 bytes long files, until 75.59% at the end of the range. The response times are almost doubled in the range. The response times increase by 6.1% at 100 bytes, whereas 26% at 12800 bytes, compared to the measurement with two clients, while the number of transferred files increase to 195.91% with 100 bytes files and 167% with 12800 bytes files.
- With eight clients, the average utilization of the processor starts at 76.31%, and finishes at 82.41%. Comparing with the 4 clients case, the average of the response times increases by 37% at 100 bytes, and by 75% at the end of the range, while the throughput of the NAT64 gateway increases by 43% and 18%.

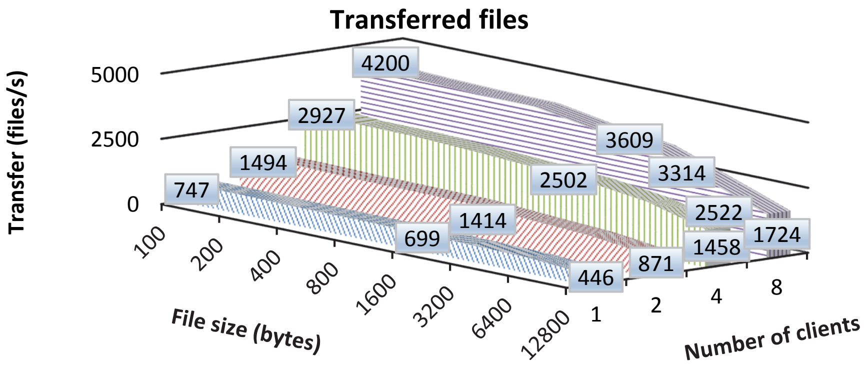

Figure 36. Number of transferred files per second by PF NAT64 gateway with UDP protocol

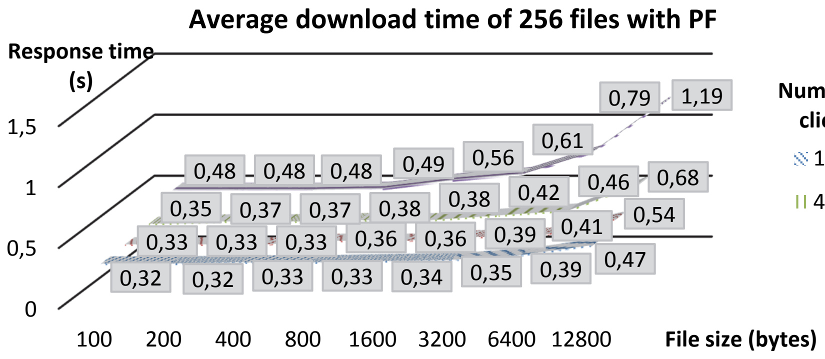

Figure 37. Average download time of 256 files with PF

Comparison of the UDP results

With one client, the PF based NAT64 gateway can transfer by about 20% more files then TAYGA at the whole range, with less than the half of the CPU usage of the TAYGA. With more clients PF gains even greater superiority over TAYGA. The advantage of PF with eight clients in the number of transferred 100 bytes long files is 295% and it is 320% with 12800 bytes files. TAYGA used all of its computing power with 4 clients, whereas PF cannot reach the 83% with eight clients. Both of the implementations proved their stability during the measurements.

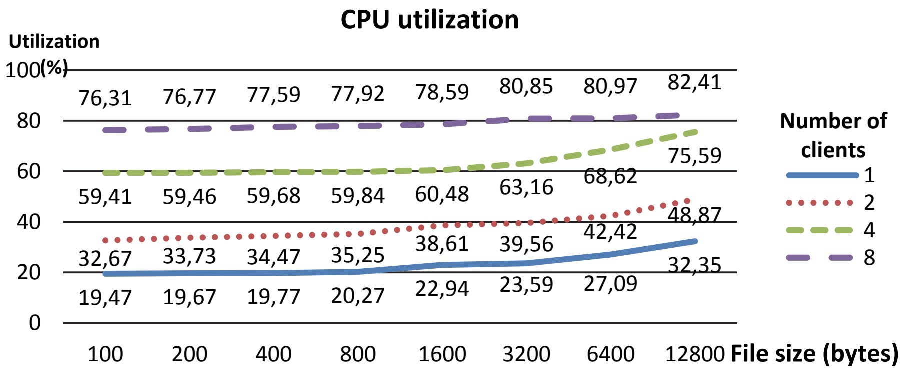

Figure 38. Average CPU utilization with PF

8. Conclusions

Both of the two NAT64 implementations were found stable enough to be used in a production environment, whereas PF of OpenBSD showed the best overall performance characteristics.

With ICMP traffic, both implementations behaved stable, and the throughput of PF was significantly better than that of TAYGA. In the serious overload situation PF outperformed TAYGA more than 4 times by means of the number of forwarded packets per second. With the TCP protocol, the dominance of PF is reduced to 3.5 times, which is still a significant value. However, PF became unpredictable under very high volume of TCP traffic. PF produced its relatively lowest values with UDP traffic, "only" 3.2 times outperformed TAYGA on Linux, and remained stable all of the time.

As for their response times, PF was 8.9 times faster than TAYGA with ICMP protocol at the highest load, whereas this proportion was 3.45 with TCP and 3.18 with UDP protocol.

Acknowledgement

The work of Sándor Répás was supported in the framework of TÁMOP 4.2.4. A/2-11-1-2012-0001 "National Excellence Program – Elaborating and operating an inland student and researcher personal support system convergence program" The project was subsidized by the European Union and co-financed by the European Social Fund.

The work of Gábor Lencse 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.

References

[1] Geoff, H.: *IPv4 Address Report*, http://www.potaroo.net/tools/ipv4/

[2] Deering, S., Hinden, R.: *Internet Protocol, Version 6 (IPv6) Specification*, (RFC 2460), 1998

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

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

[5] Egevang, K., Francis, P.: *The IP Network Address Translator (NAT)*, IETF, RFC 1631, 1994

[6] Thomson, S., Huitema, C., Ksinant, 5., Souissi, M.: *DNS Extensions to Support IP Version 6*, IETF, RFC 3596, 2003

[7] Škoberne, N., Ciglarič, M.: *Practical Evaluation of Stateful NAT64/DNS64 Translation,* Advances in Electrical and Computer Engineering, vol. 11, no. 3, pp. 49-54, 2011
DOI: 10.4316/AECE.2011.03008

[8] Bajpai, 5., Melnikov, N., Sehgal, A., Schönwälder, J.: *Flow-based Identification of Failures Caused by IPv6 Transition Mechanisms,* in Dependable Networks and Services, Springer LNCS, vol. 7279, pp. 139-150, 2012,
DOI: 10.1007/978-3-642-30633-4_19

[9] Répás, S., Hajas, T., Lencse, G.: *Application Compatibility of the NAT64 IPv6 Transition Technology* in 37th International Conference on Telecommunications and Signal Processing (TSP 2014), Berlin, Germany, pp. 49-55, 2014

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

[11] MRO: *IPv6 transition mechanisms: NAT64*, http://en.wikipedia.org/wiki/IPv6_transition_mechanisms#mediaviewer/File:NAT64.svg

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

[13] Theo de Raadt: *OpenBSD 5.1*, 2012, ISBN 978-0-9784475-9-5, http://www.openbsd.org/51.html

[14] Hodzic, E., Mrdovic, S.: *IPv4/IPv6 Transition Using DNS64/NAT64: Deployment Issues*, in IX International Symposium on Telecommunications (BIHTEL), Sarajevo, Bosnia and Herzegovina, 2012

[15] Llanto, K. J. O., Yu, W. E. S.: *Performance of NAT64 versus NAT44 in the Context of IPv6 Migration*, in International MultiConference of Engineers and Compuer Scientists (IMECS 2012), Hong Kong, pp. 638-645, 2012

[16] Monte, C. P. 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, 2012

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

[18] Lencse, G., Répás, S.: *Performance analysis and comparison of the TAYGA and of the PF NAT64 implementations*, in 36th International Conference on Telecommunications and Signal Processing (TSP 2013), Rome, Italy, pp. 71-76, 2013
DOI: 10.1109/TSP.2013.6613894

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

[20] Lencse G., Répás, S.: *Performance analysis and comparison of different DNS64 implementations for Linux, OpenBSD and FreeBSD*, in 27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013) Barcelona, Spain, pp. 877-884, 2013
DOI: 10.1109/AINA.2013.80