Performance Analysis of DNS64 and NAT64 Solutions

Gábor Lencse, Gábor Takács

***Abstract*—The need for DNS64 and NAT64 solutions is introduced and their operation is presented. A test environment for the performance analysis of DNS64 and NAT64 implementations is described. The resource requirements of the implementations are measured. The performance of DNS64 and NAT64 solutions is measured under heavy load conditions to determine if they are safe to be used in a production environment, like the network of an internet service provider.**

***Index Terms*—IPv6 deployment, DNS64, NAT64, performance analysis.**

I. INTRODUCTION

As the Internet Assigned Numbers Authority (IANA) delegated the last five "/8" IPv4 address blocks to the Regional Internet Registries in 2011 [1], and the depletion of the IPv4 address pool of the RIPE NCC (which is responsible for the IPv4 address allocations in Europe) is expected to happen in 2012 [2], the deployment of the IPv6 became inevitable in Europe, too. Internet service providers (ISPs) must urgently take preparations for both providing IPv6 services and the co-existence of the two versions of IP. (Of course, not only ISPs, but also customers (including both private and business customers) have to manage this complex change carefully [10]; and because of its complexity they have to use some integrated approach for the evaluation of all aspects concerning their activities and networks [11].)

In the past years a lot of research was done in the field of IPv6 and important theoretical results were achieved. However, if an ISP plans to introduce IPv6, it is crucial to test the *performance* and the *stability* of the different published solutions and choose the ones that are proven to be suitable.

The co-existence of IPv4 and IPv6 raises many different issues. In the beginning of the deployment of IPv6, the following situation is found to be the most typical: there will be customers that have IPv6 addresses only, and they want to connect to servers still having IPv4 addresses only. (The case of the IPv4 only clients and the IPv6 only servers will be a typical situation in a later phase of the deployment of IPv6.)

Manuscript received May 11, 2012. This work was financed by Telenor Hungary.

Gábor Lencse is with the Department of Telecommunications, Széchenyi István University, 9026 Győr, Egyetem tér 1, Hungary. phone: +36-30-409-56-60; fax: +36-96-613-646, e-mail: lencse@sze.hu

Gábor Takács was with the Department of Telecommunications, Széchenyi István University

Even though the use of dual stack by any of the parties (client or server) would solve the problem, it is not a feasible solution for ISPs because, on the one hand, they will not be able to provide their customers with IPv4 addresses as they are running out of them soon, and on the other hand, they cannot force third party server operators to use dual stack instead of IPv4 only.

The techniques that an ISP can use for solving the problem of IPv6 only clients and IPv4 only servers are DNS64 [3] and NAT64 [4]. These well-known methods have several implementations and we have carefully chosen some of them to investigate their performance and stability.

The rest of this paper is organized as follows: first, the operation of DNS64 and NAT64 is introduced, second, the selection of the implementations is discussed, third, our test environment is described, fourth, the performance measurement method of DNS64 is detailed, fifth, the DNS64 results are presented and discussed, sixth, the performance measurement method of NAT64 is described, seventh, the NAT64 results are presented and discussed, and finally, our conclusions are given.

II. THE OPERATION OF DNS64 AND NAT64

To enable an IPv6 only client to connect to an IPv4 only server, one needs *DNS64 service* and a *NAT64 gateway*. The operation of the solution is introduced using the following network as an example.

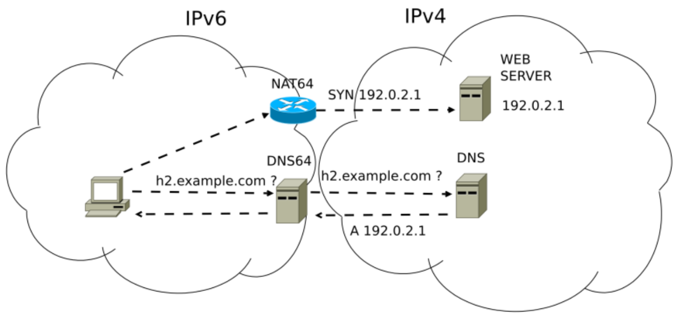

Fig. 1. DNS64 and NAT64 (source: [5], corrected by the authors)

The IPv6 only client (symbolized by a PC on the left side of Fig. 1) wants to connect to an IPv4 only server (symbolized by a web server on the right side of the figure). The *IPv4 only* server means that the DNS system has only an "A" record for the server and no "AAAA" records. A precondition for the operation of the method is that the DNS64 server should be set as the DNS server of the IPv6 only client. When the IPv6 only client tries to connect to the web server, it sends a *recursive query* to the DNS64 server to find the IPv6 address of the web server. The DNS64 server uses the normal DNS system to find out the IP address of the web server.

- If the answer contains an IPv6 address (also) then the DNS64 server returns the IPv6 address as its answer to the recursive query.
- If the answer contains only an IPv4 address then the DNS64 server returns a special IPv6 address; in our example this is the 64:ff9b::/96 prefix plus the 32 bits of the IPv4 address of the web server.

The route towards the network with given IPv6 prefix (in our example, it is 64:ff9b::/96) should be set in the IPv6 only client (and in all of the routers along the route from the client to the NAT64 gateway) to go through the NAT64 gateway.

The IPv6 only client uses the received IPv6 address to set up a connection to the desired (IPv4 only) web server. The client sends a SYN packet to the received IPv6 address. When its SYN 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 web 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 it arrives to the IPv4 only server. The IPv4 only server responds the normal way using the source address of the IPv4 packet, that is, 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. The NAT64 gateway sends the IPv6 packet back to the client. (To be able to do this, the NAT64 gateway uses stateful NAT or some other method to track the IPv6 – IPv4 mapping.)

The short example above used the *well-known prefix* described in [6]. In practice, the worldwide use of this prefix has a number of hindrances, see points 3.1 and 3.2 of [6]. For this reason, when implementing a NAT64 gateway, a given size of the subnet is reserved from the actually used IPv6 network. This solution is called *network specific prefix*. In this way, the *Infocommunications Laboratory* of the Department of Telecommunications, Széchenyi István University has got the 2001:738:2c01:8001::/64 network out of which we have reserved 2001:738:2c01:8001:ffff:ffff::/96 as the network specific prefix.

Note that for NAT64, we embed IPv4 addresses into IPv6 addresses: the last 32 bits of the IPv6 address hold the embedded IPv4 address. These kinds of IPv6 addresses are called *IPv4-embedded IPv6 addresses* [6]. There is a further naming convention. Even though their structure and the way of their generation is identical, the IPv4-embedded IPv6 addressed have two further subgroups distinguished on the basis of the purpose of their usage: IPv6 addresses used to represent IPv4 hosts in the IPv6 network are called *IPv4-converted IPv6 addresses*. (They are used in this paper.) The term *IPv4-translatable IPv6 address* is used for an IPv6 address that belongs to an IPv6 host and the purpose of the address translation is to be able to connect to an IPv4 only host. (We are not dealing with this case in this paper.)

III. THE SELECTION OF DNS64 AND NAT64 IMPLEMENTATIONS

As BIND, the most widely used DNS implementation, contains native DNS64 support from version 9.8, there was no reason to consider anything else.

As for NAT64 gateways, there are a number of implementations [5]:

- TAYGA is a stateless NAT64 implementation for Linux
- Ecdysis is a NAT64 gateway containing also DNS64
- Microsoft Forefront Unified Access Gateway is a reverse proxy and VPN solution that implements DNS64 and NAT64
- Stateless Network Address Translation 64 runs on Cisco ASR 1000 router
- Stateful NAT64 feature on Juniper MX Series 3D Universal Edge router
- OpenBSD PF packet filter is promised to be NAT64 capable in OpenBSD 5.1

From this seemingly wide selection, finally we were able to test the stability and performance of TAYGA only. Why?

- 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 in the last 15 months [7].
- The Microsoft solution is a small part of a multi function product; the whole product is not at all needed and would be too expensive and resource consuming.
- The solutions running on Cisco or Juniper routers require special hardware that we did not have.
- At the time of our measurements the current OpenBSD release was 5.0.

TAYGA is a free software under GPLv2 license and according to its developers it was intended to provide *production quality NAT64 service* [8]. TAYGA is a *stateless* NAT64 solution. 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 to the 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.

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.

IV. THE TEST SYSTEM FOR DNS64 AND NAT64 PERFORMANCE MEASUREMENTS

The aim of our tests was to examine the selected programs regarding stability and behaviour 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 System*

A test network was set up in the *Infocommunications Laboratory* of the Department of Telecommunications, Széchenyi István University. The logical topology of the network is shown in Fig. 2. The central element of the network is the DNS64/NAT64 computer. This Linux box played the role of both the DNS64 server and the NAT64 gateway but not simultaneously, but rather one after the other, as we measured the performance of the two systems separately.

For the measurements, we needed a namespace that:

- can be described systematically
- can be resolved to IPv4 only
- can be resolved without delay

The 10-{0..10}-{0..255}-{0..255}.zonat.tilb.sze.hu namespace was used for this purpose. This name space was mapped to the 10.0.0.0 – 10.10.255.255 IPv4 address by the name server at 192.168.100.105. The DNS64 server mapped these IPv4 addresses to the 2001:738:2c01:8001:ffff:ffff:0a00:0000 – 2001:738:2c01:8001:ffff:ffff:0a0a:ffff IPv6 address range.

The IPv6 only workstations at the bottom left corner of the figure played the role of the clients for both the DNS64 and the NAT64 measurements.

During the NAT64 measurements, the TAYGA NAT64 gateway used the 172.16.0.0/12 private IPv4 address range that was SNAT-ed to the 193.224.129.170 IPv4 address.

At the NAT64 gateway, the address of the next hop router towards the 10.0.0.0/8 network was set to 193.224.129.172. (The PC 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.)

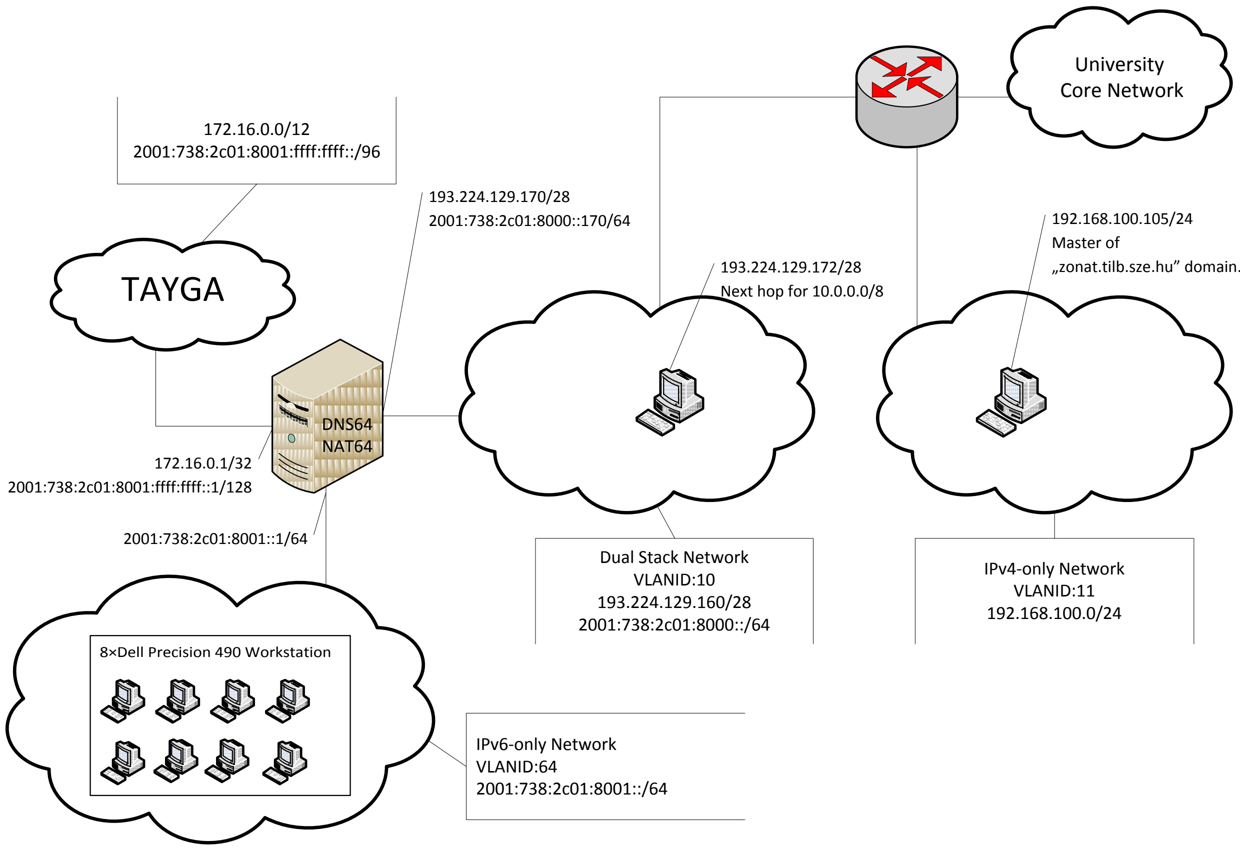

Fig. 2. Logical Topology of the DNS64 and NAT64 Test Network

The physical topology of the system is shown in Fig. 3. It is provided for the purpose that our measurements can be reconstructed and verified. Due to the infrastructural reasons that the client workstations and the DNS64/NAT64 Linux box were in two neighbouring rooms, they were interconnected by two Gigabit Ethernet switches using VLANs. (The VLAN IDs are written to the network in circles.)

B. *The Configuration of the Computers*

A test computer with special configuration was put together for the purposes of the DNS64 server and the NAT64 gateway in order that the clients will 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
- 800 MHz Intel Pentium III (Coppermine) processor
- 128 MB, 133 MHz SDRAM
- Two 3Com 3c940 Gigabit Ethernet NICs

Note that the speed of the Gigabit Ethernet could not be fully utilized due to the limitations of the PCI bus of the motherboard, but the speed was still enough to overload the CPU.

For all the other purposes (the 8 client computers, the IPv4 DNS server 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 2 GHz dual core processors
- 4x1 GB 533 MHz DDR2 SDRAM (Quad 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, too).

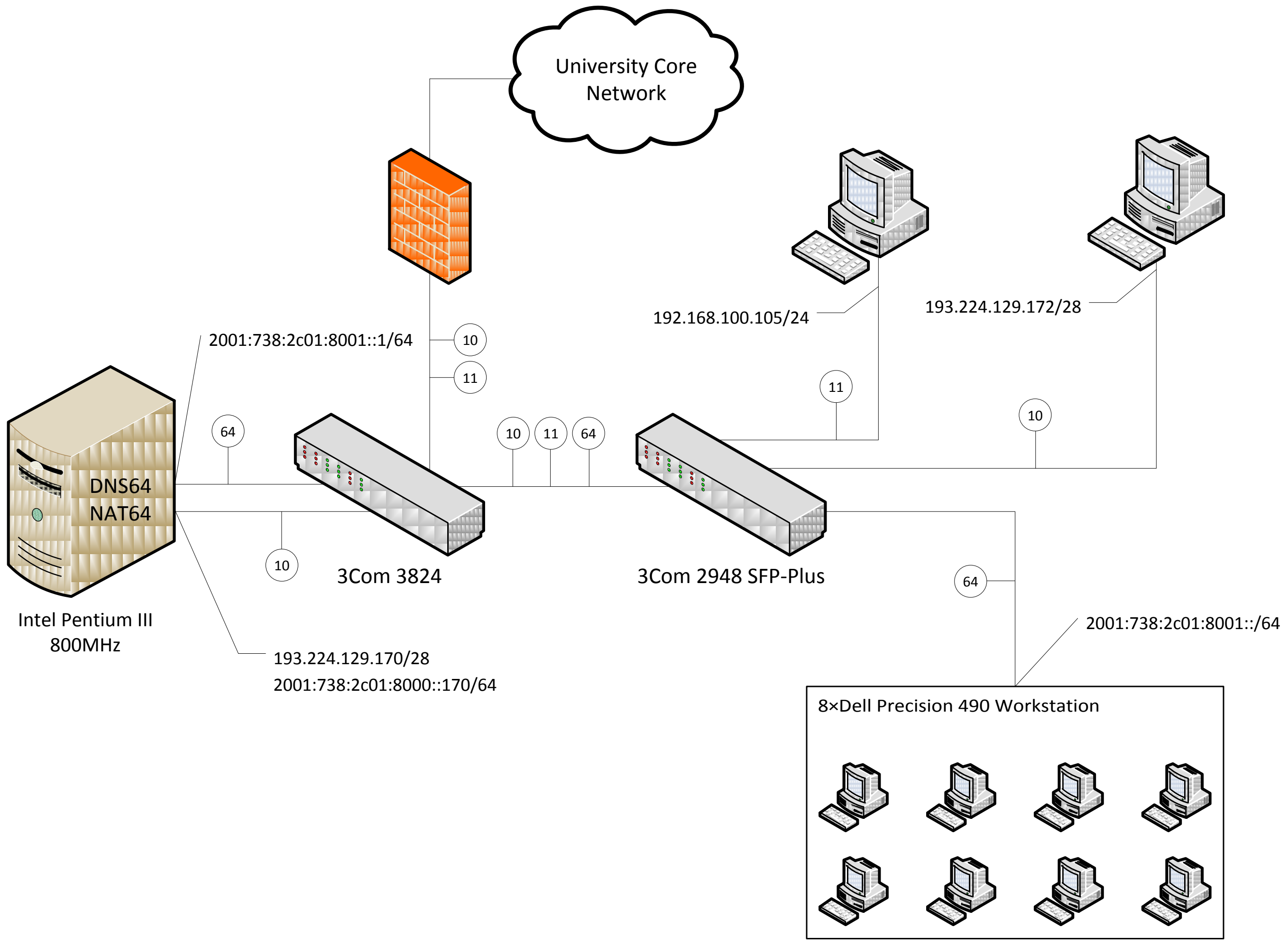

Fig. 3. Physical Topology of the DNS64 *and NAT64 Test Network*

V. DNS64 Performance Measurement Method

A. IPv4 DNS Server Settings

The DNS server was a standard DELL Linux workstation using the 192.168.100.105 IP address and the symbolic name **teacherb.tilb.sze.hu**. The version of BIND was 9.7.3 as this one can be found in the Debian Squeeze distribution and there was no need for special functions (unlike in the case of the DNS64 server).

The 10.0.0.0/16-10.10.0.0/16 IP address range was registered into the **zonat.tilb.sze.hu** zone with the appropriate symbolic names. The zone file was generated by the following script:

```
#!/bin/bash
cat > db.zonat.tilb.sze.hu << EOF
\$ORIGIN zonat.tilb.sze.hu.
\$TTL    1
@ IN SOA teacherb.tilb.sze.hu. kt.tilb.sze.hu. (
                    2012012201 ; Serial
                         28800 ; Refresh
                          7200 ; Retry
                        604800 ; Expire
                             2 ) ; Min TTL

@   86400   IN    NS   teacherb.tilb.sze.hu.

EOF

for a in {0..10}
do
    for b in {0..255}
    do
        echo '$'GENERATE 0-255 10-$a-$b-$ IN A \
             10.$a.$b.$ >> db.zonat.tilb.sze.hu
    done
done

echo "" >> db.zonat.tilb.sze.hu
```

The first general line of the zone file (describing the symbolic name resolution) was the following one:

```
$GENERATE 0-255 10-0-0-$ IN A 10.0.0.$
```

A line of this kind is equivalent with 256 traditional “IN A” lines; the **$GENERATE** directive was used for shorthand purposes.

As it can be seen from the script above and as it has been mentioned earlier, these symbolic names have only “A” records and no “AAAA” records, so the generation of the IPv6 addresses is the task of the DNS64 server.

B. DNS64 Server Settings

The network interfaces of the freshly installed Debian Squeeze Linux operating system on the Pentium III computer were set according to the logical topology shown in Fig. 2.

For the purposes of the DNS64 server, the BIND 9.8 was compiled from source (as the Debian Squeeze did not contain this version yet).

The 2001:738:2c01:8001:ffff:ffff::/96 prefix was set to BIND for the DNS64 function using the **dns64** option in the file **/etc/bind/named.conf.options**.

In order to facilitate the IPv6 SLAAC (*Stateless Address Autoconfiguration*) of the clients, **radvd** (*Router Advertisement Daemon*) was installed on the NAT64 gateway.

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

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

C. Client Settings

Debian Squeeze was installed for the DELL computers used for client purposes, too. On these computers, the DNS64 server was set as name server in the following way:

```
echo "nameserver 2001:738:2c01:8001::1" > \
      /etc/resolv.conf
```

D. DNS64 Performance Measurements

The CPU and memory consumption of the DNS64 server was measured in the function of the number of requests served. The measure of the load was set by starting test scripts on different number of client computers (1, 2, 4 and 8). In order to avoid the overlapping of the namespaces of the client requests (to eliminate the effect of the DNS caching), the requests from the number **i** client used target addresses from the 10.$i.0.0/16 network. In this way, every client could request 2^{16} different address resolutions. For the appropriate measurement of the execution time, 256 experiments were done and in every single experiment 256 address resolutions were performed using the standard **host** Linux command. The execution time of the experiments was measured by the GNU **time** command. (Note that this command is different from the **time** command of the bash shell.)

The clients used the following script to execute the 256 experiments:

```
#!/bin/bash
i=`cat /etc/hostname|grep -o .$`
rm dns64-$i.txt
do
    for b in {0..255}
    do
        /usr/bin/time -f "%E" -o dns64-$i.txt \
           -a ./dns-st-c.sh $i $b
    done
done
```

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**).

The **dns-st-c.sh** script (taking two parameters) was responsible for executing a single experiment with the resolution of 256 symbolic names:

```
#!/bin/bash
for c in {0..252..4} # that is 64 iterations.
do
    host 10-$1-$2-$c.zonat.tilb.sze.hu &
    host 10-$1-$2-$((c+1)).zonat.tilb.sze.hu &
    host 10-$1-$2-$((c+2)).zonat.tilb.sze.hu &
    host 10-$1-$2-$((c+3)).zonat.tilb.sze.hu
done
```

In every iteration of the **for** cycle, four **host** commands were started, out of which the first three were started asynchronously (“in the background”) that is, the four commands were running in parallel; and the core of the cycle was executed 64 times, so altogether 256 host commands were executed. (The client computers had two dual core CPUs that is why four commands were executed in parallel to generate higher load.)

First, a test was performed with one client only. After a while, the *conntrack table* of the *netfilter* of the test computer running DNS64 became full and the name resolution stopped functioning. As DNS64 does not require *netfilter*, the *netfilter* module was removed from the kernel of the computer. After this, the test was completed with no errors.

As a production system may require the presence of **iptables** for security reasons[1], a different solution may be needed. One may increase the size of the *conntrack* table (it is necessary to increase the value of the *hashsize* parameter proportionally, too), or decrease the value of the timeout. As the first one has a resource (memory) requirement, the second one was chosen. The timeout for the UDP packets was decreased from 30s to 1s. The exact name of the changed kernel parameter is:

```
/proc/sys/net/netfilter/nf_conntrack_udp_timeout
```

With this setting, the test was completed with no errors even with 8 clients (that produce much higher load).

Next, the number of clients was increased from one to eight (the used values were: 1, 2, 4 and 8) and the time of the DNS resolution was measured. The *CPU and memory utilization* were also measured on the test computer running DNS64. The following command line was used:

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

VI. DNS64 PERFORMANCE RESULTS

The most important results were summarized in Table 1. The first row of the table shows the number of clients. (The load of the DNS64 server was increasing in the function of this parameter.) The second row shows the execution time of one experiment (that is the execution of 256 **host** commands). Even though the results showed little deviation, the standard deviation is included in the third row.

[1]Firewalls often use *stateful packet inspection* today (that requires a *conntrack table*); however this solution is highly susceptible to DDoS attacks. The proliferation of DDoS attacks reported in [12] may require the use of the *stateless packet inspection* instead of the *stateful* one.

Rows number four and five show the CPU utilization and the standard deviation of the CPU utilization, respectively.

Row number six shows the estimated memory consumption of DNS64. (This parameter can be measured with high uncertainty, as its value is quite low and other processes than DNS64 may also influence the size of free/used memory of the Linux box.) Fortunately, it can be seen, that its value was always really low.

The size of the *conntrack table* was also logged during the experiments using the **nf_conntrack_count** kernel parameter. Its maximum is displayed in row 7.

The number of DNS64 requests per second, served by the test computer, was calculated using the number of clients (in row 1) and the execution time values (in row 2) and it is displayed in the last row of the table.

TABLE 1. DNS64 PERFORMANCE RESULTS

1	Number of clients	1	2	4	8
2	Exec. time of 256 **host** commands [s]	1,195	1,746	3,643	7,287
3	(standard deviation)	0,074	0,040	0,031	0,050
4	CPU utilization [%]	66,9	97,0	100,0	100,0
5	(standard deviation)	3,8	2,0	0,0	0,0
6	DNS64 memory consumption [MB]	0,5	1,4	2,0	2,4
7	Maximum size of the conntrack table	2347	4989	6474	7493
8	Number of requests served [request/s]	214	293	281	281

On the basis of the results above, we can state that:

- The increase of the load does not cause serious performance degradation and the system does not at all tend to collapse due to overload. Even when the CPU utilization is about 100% the response time increases approximately *linearly* with the load (that is, with the number of clients)
- We cannot give an exact estimation for the memory consumption of DNS64, but it is visibly very low even for extremely high loads.
- It can be seen from the last row of the table that the maximum of the number of requests served was achieved using two clients. The further increase in the number of clients caused only increase in the response time, but the number of requests per second could not increase. It was so, because the test program did not send a new request until all the four **host** commands (running in parallel) received an answer.
- The maximum size of the *conntrack table* increased with the load; it means that one must be aware of the size of the *conntrack table* in the case of a production system.

The results presented above are very important, because they show that the behaviour of the DNS64 system complies with the so called *graceful degradation* [9] principle; if there are not enough resources for serving the requests then the response time of the system increases only *linearly* with the load.

To compare the performance of the DNS64 to the performance of a *caching-only DNS server*, the same series of experiments were performed with the only difference that the DNS64 was switched off in the BIND, so the test computer was functioning as a caching-only DNS server. The results showed that DNS64 needs only a very little more computing power than a caching-only name server. E.g. the DNS64 test with 8 clients (each clients executed 256 **host** commands) lasted 7,287s (std. dev.: 0,050), and the same experiment with the caching-only DNS server lasted 7,009s (std. dev.: 0,036).

VII. NAT64 PERFORMANCE MEASUREMENT METHOD

A. *NAT64 Gateway Settings*

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 next hop address for the 10.0.0.0/8 IPv4 network was set to 193.224.129.172.

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 193.224.129.172 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 193.224.129.172
```

As the DELL computer had much more computing power than that of the Pentium III test computer, it was able to answer easily instead of the IPv4 only computers. E.g. in the case of the **ping6** test with 8 clients, the CPU utilization of the DELL computer was under 4%.

C. *Client Settings*

The DELL computers were used as clients. No DNS server was used, but the clients prepared the necessary IPv4-converted IPv6 addresses for themselves by concatenating the 2001:738:2c01:8001:ffff:ffff::/96 prefix and the appropriate IPv4 addresses from 10.0.0.0/8. At the client computers, the next hop towards the 2001:738:2c01:8001:ffff:ffff::/96 network was set to the IPv6 address of the NAT64 gateway:

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

D. *NAT64 Performance Measurements*

The following script was executed by 1, 2, 4 and 8 clients:

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

Using the **ping6 -c11** command, eleven *echo request* ICMPv6 messages were sent to all of the generated IPv6 addresses.

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
```

VIII. NAT64 PERFORMANCE RESULTS

The results can be found in Table 2. Row 1 shows the number of clients that executed the test script. Rows 2, 3 and 4 show the packet loss ratio, the response time, and the standard deviation of the response time, respectively. The following two rows show the CPU utilization of the test computer and its standard deviation. Row 7 shows the number of 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 line show the memory consumption measured at the test computer.

TABLE 2. NAT64 *PERFORMANCE RESULTS*

1	number of clients	1	2	4	8
2	packet loss [%]	0,025	0,020	0,008	0,025
3	average response time of **ping6** [ms]	0,36	0,37	0,52	1,11
4	(std. deviation)	0,20	0,06	0,11	0,20
5	CPU utilization [%]	26,1	49,6	79,2	93,9
6	(std. deviation)	4,1	5,0	5,6	3,0
7	measure of the traffic [packets/s]	2207	4322	6804	7491
8	NAT64 memory consumption [MB]	4,6	6,2	5,8	3,1

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).
- The response time showed no increase while the CPU utilization was far from 100% (0.36 and 0.37 for one and two clients, respectively). Later it started growing and for eight clients it was nearly twice as much as it was for four clients.
- The behaviour of the response time is in a good agreement with the changes that can be seen in the number of packets: when the number of clients was increased from one to two, the number of packets were also nearly doubled, the next doubling of the number of clients could cause only 50% increase in the number of packets, and for 8 clients the number of packets grew only a little from 6804 to 7491 as TAYGA could not serve more packets due to the lack of CPU capacity.
- The memory consumption does not show correlation with the load, but it is very low again.

To sum up the findings above, we can lay down that TAYGA performed well, its memory consumption was found to be very low and its response time started growing at high CPU utilization but still remained proportional only with the load, that is, TAYGA also complied with the *graceful degradation* principle.

IX. CONCLUSIONS

A test environment and the methods for the performance analysis of DNS64 and NAT64 solutions were described.

The resource requirements and the performance of the DNS64 support of BIND 9.8 and of the stateful NAT64 solution achieved by the combination of the stateless TAYGA plus iptables NAT44 were measured.

It was found that these implementations are stable even under heavy load conditions and their performance complies with the graceful degradation principle under serious overload.

We conclude that they are safe to be used in a production environment like the network of an internet service provider.

REFERENCES

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

[2] Geoff Huston, “IPv4 address exhaustion: A progress report”, in: NANOG 53, Philadelphia, PA, USA, October 9-12. 2011. http://www.nanog.org/meetings/nanog53/presentations/Wednesday/Huston.pdf

[3] 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)

[4] 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)

[5] “NAT64” http://en.wikipedia.org/wiki/NAT64

[6] 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)

[7] “Ecdysis: open-source implementation of a NAT64 gateway” http://ecdysis.viagenie.ca/news.html

[8] “TAYGA: Simple, no-fuss NAT64 for Linux” http://www.litech.org/tayga/

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

[10] J. Mohácsi, “IPv6 deployments strategies for enterprises and ISPs”, in “IPv6 in a nutshell”, Hungarian IPv6 Forum Conference, Budapest, Hungary, May 3, 2012. (in Hungarian) http://ipv6forum.hu/sites/ipv6forum.hu/files/03.Mohacsi_Janos_ipv6_diohej.pdf

[11] L. Muka and G. Muka, “Creating and using key network-performance indicators to support the design and change of enterprise info-communication infrastructure”, 2012 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2012), Genoa, Italy, July 8-11, 2012. Volume 44, Book 12, ISBN: 978-1-61839-982-3. pp. 737-742.

[12] Arbor Networks, “Worldwide Infrastructure Security Report”, Volume VII, 2011. http://www.arbornetworks.com/report

Gábor Lencse received his MSc in electrical engineering and computer systems at the Technical University of Budapest in 1994, and his PhD in 2001.

He has been working for the Department of Telecommunications, Széchenyi István University in Győr since 1997. He teaches Computer networks, Networking protocols and the Linux operating system. Now, he is an Associate Professor. He is responsible for the specialization of information and communication technology of the BSc level electrical engineering education. He is a founding member of the Multidisciplinary Doctoral School of Engineering Sciences, Széchenyi István University. The area of his research includes discrete-event simulation methodology and performance analysis of computer networks. Dr. Lencse has been working part time for the Department of Telecommunications, Budapest University of Technology and Economics (the former Technical University of Budapest) since 2005. There he teaches Computer architectures and Media communication networks. He was the leader of the “IPv6 deployment strategy for the Telenor Hungary” R&D project of the Department of Telecommunications, Széchenyi István University.

Gábor Takács received his BSc in electrical engineering with specialisation in information and communication technology at the Széchenyi István University in January 2012. His thesis was promoted to the “HTE Thesis Competition 2012”.

He has five year experience in Linux administration. He worked in the Electromagnetic Field Laboratory at the University. His job was to keep running the high performance computers (e.g. IBM BladeCenter) and the network administration. He played an important role in the “IPv6 deployment strategy for the Telenor Hungary” R&D project of the Department of Telecommunications, Széchenyi István University.