Performance Analysis and Comparison of Different DNS64 Implementations for Linux, OpenBSD and FreeBSD

Gábor Lencse
Department of Telecommunications
Széchenyi István University
Győr, Hungary
lencse@sze.hu

Sándor Répás
HunNet-Média Ltd.
Budapest, Hungary
RSandor@AHOL.co.hu

***Abstract*—The transition mechanisms for the first phase of IPv6 deployment are surveyed and the most important DNS64 solutions are selected. The test environment and the testing method are described. As for the selected DNS64 implementations, the performance of both BIND9 and TOTD running under Linux, OpenBSD and FreeBSD are measured and compared. The stability of all the tested DNS64 solutions was analyzed under serious overload conditions to test if they may be used in production environments with strong response time requirements.**

IPv6 deployment, IPv6 transition solutions, performance analysis, DNS64, BIND9, TOTD, Linux, OpenBSD, FreeBSD

I. INTRODUCTION

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

Different free DNS64 implementations were considered and two of them (BIND9 and TOTD) were selected for testing. The aim of our research was to compare the performance of the selected implementations running on different free operating systems (Linux, OpenBSD and FreeBSD) and to analyze their behavior under heavy load conditions. Our results are expected to give valuable information to many network administrators when selecting the appropriate IPv6 transition solution for their networks.

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

The performance analysis and comparison of some selected NAT64 implementations under Linux and OpenBSD was also a part of our research, however, the amount of the results would exceed the space limitations of this paper thus they are planned to be published in another paper [20].

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

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

A. The Most Important Solutions

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

1. NAT-PT/NAPT-PT [5] started its life as a proposed standard in 2000 but due to several issues it was put to historic status in 2007 [6].
2. The use of an Application Level Gateway [7] is an operable alternative, however, it is rather expensive as ALGs have to be both developed and operated for all the different applications.
3. The most general and flexible solution is the use of a DNS64 [1] server and a NAT64 [2] gateway.

B. The Operation of DNS64 and NAT64

To enable an IPv6 only client to connect to an IPv4 only server, one can use a *DNS64 server* and a *NAT64 gateway*. The DNS64 server should be set as the DNS server of the IPv6 only client. When the IPv6 only client tries to connect to any server, it sends a recursive query to the DNS64 server to find the IPv6 address of the given server. The DNS64 server uses the normal DNS system to find out the IP address of the server.

- If the answer contains an IPv6 address 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 constructs and returns a special IPv6 address called *IPv4-Embedded IPv6 Address* [8] containing the IPv4 address of the server in the last 32 bits. In the first 96 bits, it may contain the *NAT64 Well-known Prefix* or a *network specific prefix* from the network of the client.

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

The IPv6 only client uses the received IPv6 address to communicate with the desired (IPv4 only) server. The traffic of the IPv6 only client and the IPv4 only server travels through the NAT64 gateway, which makes their communication possible by constructing and forwarding the appropriate version IP packets.

For a more detailed but still easy to follow introduction, see [9] and for the most accurate and detailed information, see the relating RFCs: [1] and [2].

C. A Short Survey of the Current Research Results

Several papers were published in the topic of the performance of DNS64 and NAT64 in 2012. The performance of the TAYGA NAT64 implementation (and implicitly of the TOTD DNS64 implementation) is compared to the performance of NAT44 in [10]. 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 [11]. The performance of the Ecdysis NAT64 implementation (and implicitly the performance of its DNS64 implementation) is compared to the performance of both the NAT-PT and an HTTP ALG in [12]. All of these papers deal with the performance of a given DNS64 implementation with a given NAT64 implementation. On the one hand this is natural, as both services are necessary for the operation, on the other hand this is a kind of "tie-in sale" that may hide the real performance of a given DNS64 or NAT64 implementation by itself. Even though both services are necessary for the complete operation, in a large network they are usually provided by separate, independent devices; DNS64 is provided by a name server and NAT64 is performed by a router. Thus the best implementation for the two services can be – and also should be – selected independently. The performance of the BIND DNS64 implementation and that of the TAYGA NAT64 implementation are analyzed separately and also their stability is tested in [13]. However, only one implementation was considered for each service, so even if they were proved to be stable and fast enough, more research is needed for the comparison of the performance (and also the stability) of multiple DNS64 and NAT64 implementations. Due to space limitations, this paper deals with DNS64 only, our research results concerning NAT64 implementations will be published in a separate paper [20].

III. The Selection of DNS64 Implementations

Only free software [14] (also called open source [15]) implementations were considered.

As BIND [16], the most widely used DNS implementation, contains native DNS64 support from version 9.8, it was a must that it will be chosen.

As BIND is a large and complex software containing all the different DNS functionalities (authoritative, recursive, DNSSEC support, etc.) our other choice was a lightweight one, namely TOTD [17].

Both BIND and TOTD were tested under all the three operating system, namely: Linux, OpenBSD and FreeBSD.

IV. The Test Environment for DNS64 Performance Measurements

The aim of our tests was to examine and compare the performance of the selected DNS64 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

A test network was set up in the *Infocommunications Laboratory* of the Department of Telecommunications, Széchenyi István University. The topology of the network is shown in Fig. 1. The central element of the test network is the DNS64 server.

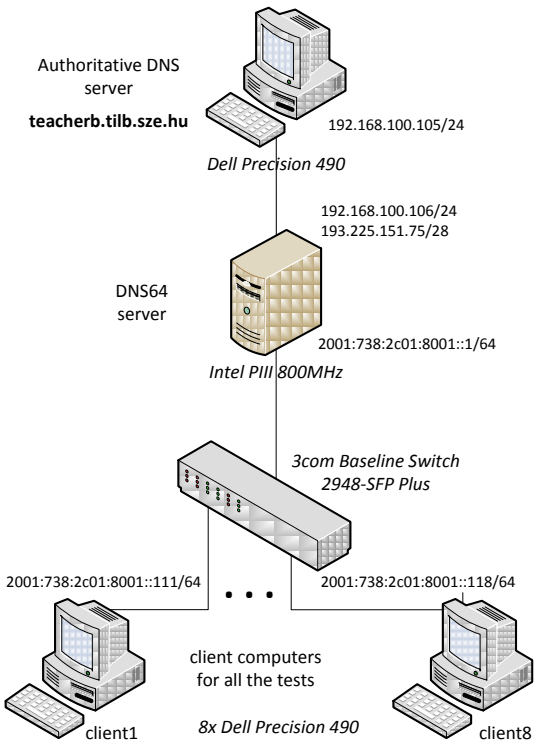

Figure 1. Topology of the DNS64 test network.

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 name space was used for this purpose. This namespace was mapped to the 10.0.0.0 – 10.10.255.255 IPv4 addresses by the name server at 192.168.100.105.

The DNS64 server mapped these IPv4 addresses to the IPv6 address range 2001:738:2c01:8001:ffff:ffff:0a00:0000 – 2001:738:2c01:8001:ffff:ffff:0a0a:ffff.

The DELL IPv6 only workstations at the bottom of the figure played the role of the clients for the DNS64 measurements.

B. The Configuration of the Computers

A test computer with special configuration was put together for the purpose of the DNS64 server in order that the clients will be able to produce high enough load for overloading it. The CPU and memory parameters were chosen to be as little as possible from our available hardware base in order to be able to create an overload situation with a finite number of clients, and only the network cards were chosen to be fast enough. The configuration of the test computer was:

- Intel D815EE2U motherboard
- 800MHz Intel Pentium III (Coppermine) processor
- 128MB, 133MHz 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 and the IPv4 DNS server) standard *DELL Precision Workstation 490* computers were used with the following configuration:

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

Debian Squeeze 6.0.3 GNU/Linux operating system was installed on all the computers (including the Pentium III test computer when it was used under Linux). The version of the OpenBSD and FreeBSD operating systems installed on the test computer were 5.1 and 9.0, respectively.

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`**. BIND was used for authoritative name server purposes in all the DNS64 experiments. 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 to 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 was the task of the DNS64 server.

B. The operation mode of the DNS servers

If a DNS (or DNS64) server receives a recursive query, it can act in two ways: it may resolve the query itself by performing a series of iterative queries or it may ask another name server to resolve the query. A name server that resolves the recursive queries is called *recursor* and a name server that asks another name server to resolve them is called *forwarder*. While BIND can be either of them, TOTD can act only as a forwarder. (See more details later.)

C. DNS64 Server Settings

Several combinations of the DNS64 implementations and operating systems were tested, e.g. both BIND 9.8 and TOTD were tested under Linux, OpenBSD and FreeBSD.

1) Preparation of the Linux test system

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

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

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

```
interface eth2
{
        AdvSendAdvert on;
```

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

Now, the DNS64 server was functionally ready for the operation, however, during our preliminary tests, the *conntrack table* of the *netfilter* of the test computer providing the DNS64 service became full and the name resolution stopped functioning. As DNS64 does not require *netfilter*, the removal of the *netfilter* module from the kernel of the computer can be a feasible solution. If one needs *netfilter* for any reason then either the size of the *conntrack* table may be increased (it is necessary to increase the value of the *hashsize* parameter proportionally, too) or the value of the timeout can be decreased. As the first one has a resource (memory) requirement, the second one was chosen. The timeout for the UDP packets was decreased from 30 seconds to 1 second. The exact name of the changed kernel parameter was:

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

2) Preparation of the BSD test systems

Similarly to the Linux test system, the network interfaces of the BSD systems were set up as shown in Fig. 1. The content of the **/etc/rtadvd.conf** file was set as follows for both OpenBSD and FreeBSD:

```
default:\
:chlim#64:raflags#0:rltime#1800:rtime#0:retrans#0:\
:pinfoflags="la":vltime#2592000:pltime#604800:mtu#0:
sk1:\
:addr="2001:738:2c01:8001::":prefixlen#64:tc=default:
```

The limitation of the UDP timeout and the increase of the maximum number of connections were achieved by the following two lines in **/etc/pf.conf**:

```
set timeout interval 1
set limit states 40000
```

The following lines are optional, but they were used too:

```
pass in on sk0 inet proto udp from any port 53 to \
 193.225.151.75 no state
pass out on sk0 inet proto udp from 193.225.151.75 to \
 any port 53 no state
pass in on sk1 inet6 proto udp from any to \
 2001:738:2c01:8001::1 port 53 no state
pass out on sk1 inet6 proto udp from \
 2001:738:2c01:8001::1 port 53 to any no state
```

In this way, PF does not record the state of the DNS requests and answers. (It is not necessary, as the possible number of states was already increased to 40000 above.)

3) The set up of the BIND DNS64 server

The BIND 9.8 was compiled from source under Linux and OpenBSD. FreeBSD version 9.0 already contained the 9.8.1-P1 version of BIND.

The 2001:738:2c01:8001:ffff:ffff::/96 (network specific) prefix was set to BIND for the DNS64 function using the **dns64** option in the file **/etc/bind/named.conf.options**. Now, BIND was ready to operate as a recursor. To make the performance of BIND and TOTD comparable, BIND was also set as a forwarder. It was done by the following additional settings in the **named.conf** file:

```
forwarders { 192.168.100.105; };
forward only;
```

4) The set up of the TOTD DNS64 server

As TOTD is just a DNS forwarder and not a DNS recursor it was set to forward the queries to the BIND running on the **teacherb** computer. The content of the **/etc/totd.conf** file was set as follows:

```
forwarder 192.168.100.105 port 53
prefix 2001:738:2c01:8001:ffff::
retry 300
```

D. Client Settings

Debian Squeeze was installed on 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
```

E. 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, from which the first three were started asynchronously ("in the background") that is, the four commands were running in (quasi) 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.)

In the series of measurements, 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.

Under Linux, the following command line was used:

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

Under the BSD operating systems, the command line was:

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

VI. DNS64 Performance Results

The results are presented in similar tables for all the tested DNS64 implementation and operating system combinations. A detailed explanation is given for the first table only – the others are to be interpreted in the same way.

A. DNS64 Performance Results of BIND

First, BIND implementing DNS64 was used as a recursor, as it is a natural solution to complete the whole task. However, as TOTD can act only as a forwarder, later BIND was also tested as a forwarder to be able to compare their performance.

1) Linux, BIND is a recursor

The performance results of the DNS64 server realized by BIND used as a recursor running under Linux were summarized in Table I. 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, third and fourth rows shows the average, the standard deviation and the maximum value of the execution time of the execution of 256 **host** commands (this is called one experiment). The results show little deviation and the maximum values are always close the average.

Rows number five and six show the average value and the standard deviation of the CPU utilization, respectively.

TABLE I. DNS64 Performance: BIND, Linux, Recursor

1	**Number of clients**	**1**	**2**	**4**	**8**
2	**Exec. time of 256 host commands (s)**	1.242	1.862	3.748	7.541
3	**standard deviation**	0.018	0.050	0.076	0.282
4	**maximum value**	1.550	2.260	3.970	12.690
5	**CPU utilization (%)**	67.66	96.63	100.00	100.00
6	**standard deviation**	1.2	2.3	0.0	0.0
7	**DNS64 memory consumption (MB)**	36	49	52	50
8	**Number of requests served (request/s)**	206	275	273	272

Row number seven shows the estimated memory consumption of DNS64. (This parameter can be measured with high uncertainty, as its value is not too high and other processes than DNS64 may also influence the size of free/used memory of the Linux box.)

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

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 moderate 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. The reason for this was that the test program did not send a new request until the lastly started one of the four host commands (running in parallel) received an answer.

The results presented above are very important, because they show that the behavior of the DNS64 system realized by BIND as a recursor running under Linux complies with the so called *graceful degradation* [18] principle; if there are not enough resources for serving the requests then the response time of the system increases only *linearly* with the load.

Another very important observation is that even for 8 clients, the standard deviation of the execution time (0.28s) is less than 4% of the average (7.54s) and also the maximum value of the execution time (12.69s) is less then than double of the average. This means that the system shows stability even in a very serious overload situation.

These two observations make BIND running under Linux a good candidate for DNS64 server solution in a production network with strong response time requirements.

2) OpenBSD, BIND is a recursor

The performance results of the DNS64 server realized by BIND used as a recursor running under OpenBSD were summarized in Table II. Even though the average values for the execution time or for the number of requests served in a second are not far from that of the Linux system, there is a serious difference: for eight clients the standard deviation (7.3s) of the execution time of 256 **host** commands is about as high as its average value (7.6s) and the maximum value of the execution time (52.7s) is about 7 times higher than the average. Let us see the distribution of the execution time more precisely. Fig. 2 shows a histogram of the execution time of 256 **host** commands (that is one experiment) for two

TABLE II. DNS64 PERFORMANCE: BIND, OPENBSD, RECURSOR

	Number of clients	1	2	4	8
2	Exec. time of 256 `host` commands (s)	1.259	2.013	4.194	7.626
3	standard deviation	0.009	0.051	3.564	7.305
4	maximum value	1.300	2.180	23.340	52.690
5	CPU utilization (%)	72.94	96.90	97.02	98.12
6	standard deviation	1.8	2.2	5.5	5.6
7	DNS64 memory consumption (MB)	35	45	45	45
8	Number of requests served (request/s)	203	254	244	269

clients (each client executed 256 experiments thus 512 execution time values were produced). The execution time values are located in a relatively narrow range. Fig. 3 shows the histogram of the execution time values for 8 clients (there are 8 times 256 = 2048 values). The results are scattered in a wide range.

Because of the huge deviation of the results, we do not recommend the use of BIND under OpenBSD for DNS64 services in a production environment.

3) FreeBSD, BIND is a recursor

The performance results of the DNS64 server realized by BIND used as a recursor running under FreeBSD were summarized in Table III. From the three analyzed platforms, BIND produced the poorest performance results under FreeBSD considering the average execution time or the number of requests served in a second. It is very likely caused by the program execution in a jail environment [19] under FreeBSD for security reasons. However, BIND under FreeBSD showed even greater stability than under Linux. As for the execution time of one experiment in the case of eight clients, the value of the standard deviation (0.09s) is less than 1% of the average (11.92s) and the maximum value (12.14s) is very close to the average. The system also complied with the graceful degradation principles. Thus if the FreeBSD platform is preferred for security or some other reasons and one can accept the performance sacrifice, BIND under FreeBSD can be a choice for DNS64 service in a production system.

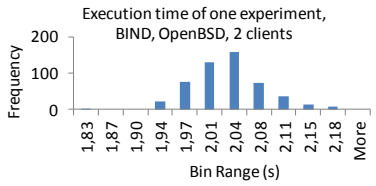

Figure 2. Distribution of the execution time of one experiment for BIND, OpenBSD, 2 clients

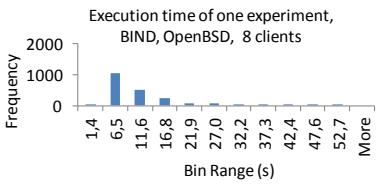

Figure 3. Distribution of the execution time of one experiment for BIND, OpenBSD, 8 clients.

TABLE III. DNS64 PERFORMANCE: BIND, FREEBSD, RECURSOR

	Number of clients	1	2	4	8
2	Exec. time of 256 `host` commands (s)	1.815	2.979	5.900	11.923
3	standard deviation	0.018	0.034	0.075	0.090
4	maximum value	2.000	3.070	6.380	12.140
5	CPU utilization (%)	78.58	97.51	100.00	100.00
6	standard deviation	1.8	1.6	0.0	0.0
7	DNS64 memory consumption (MB)	30	36	42	53
8	Number of requests served (request/s)	141	172	174	172

The performance of BIND as a recursor under the three platforms is visualized for comparison in Fig. 4. The huge maximum values of the execution time under OpenBSD totally discredit the platform for BIND under heavy load.

4) Linux, BIND is a forwarder

The performance results of the DNS64 server realized by BIND used as a forwarder running under Linux were summarized in Table IV. They appear here mainly for a fair performance comparison with TOTD. There is not much difference but a slight (about 20%) increase of the peak performance can be observed comparing the values to Table I. It is due to handing over the task of the recursion to the BIND running on **`teacherb.tilb.sze.hu`**.

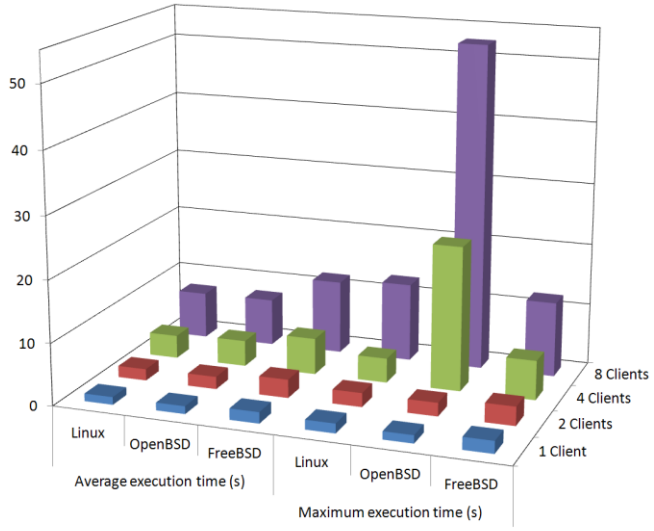

Figure 4. The Execution time of one experiment (256 **`host`** commands) using BIND as a recursor under Linux, OpenBSD and FreeBSD

TABLE IV. DNS64 PERFORMANCE: BIND, LINUX, FORWARDER

	Number of clients	1	2	4	8
2	Exec. time of 256 `host` commands (s)	1.127	1.542	3.183	6.318
3	standard deviation	0.049	0.037	0.081	0.104
4	maximum value	1.650	1.680	3.310	6.470
5	CPU utilization (%)	61.77	95.56	100.00	100.00
6	standard deviation	4.1	2.3	0.0	0.0
7	DNS64 memory consumption (MB)	40	58	57	57
8	Number of requests served (request/s)	227	332	322	324

TABLE V. DNS64 PERFORMANCE: BIND, OPENBSD, FORWARDER

	Number of clients	1	2	4	8
2	Exec. time of 256 `host` commands (s)	1.211	1.749	3.633	7.291
3	standard deviation	0.008	0.051	2.900	7.800
4	maximum value	1.230	1.910	17.510	37.890
5	CPU utilization (%)	67.46	97.21	98.11	98.48
6	standard deviation	1.9	2.4	5.4	6.0
7	DNS64 memory consumption (MB)	37	52	49	47
8	Number of requests served (request/s)	211	293	282	281

5) OpenBSD, BIND is a forwarder

The performance results of the DNS64 server realized by BIND used as a forwarder running under OpenBSD were summarized in Table V. Similarly to the Linux system, there is not much difference but a slight increase of the performance can be observed comparing the values to Table II. And also the deviation was increased further; for 8 clients it is even larger than the average.

6) FreeBSD, BIND is a forwarder

The performance results of the DNS64 server realized by BIND used as a forwarder running under FreeBSD were summarized in Table VI. The stability of BIND under FreeBSD remained, but the peak performance increased only about 7% compared to the values in Table III.

B. DNS64 Performance Results of TOTD

As it was mentioned earlier, TOTD can act only as a forwarder.

1) Linux, TOTD is a forwarder

The performance results of the DNS64 server realized by TOTD used as a forwarder running under Linux were summarized in Table VII. TOTD under Linux excels with both its very low memory consumption and its high average performance. The low memory consumption is very likely caused by the lack of caching. As our experiments were designed to eliminate the effect of caching by using different IP addresses in each query, the lack of caching caused no performance penalty. However, in a real life system, the average performance of TOTD may be worse than BIND that uses caching. But the very low memory consumption of TOTD can be an advantage in a small embedded system.

TOTD under Linux can perform well if the load can be limited. For one client, the maximum value of the execution time was acceptable (less than twice of the average) and TOTD served 324 requests per second, while BIND running under Linux as a forwarder could serve only 227 (see Table IV).

However, the results of TOTD for 2, 4 and 8 clients are unacceptable: the maximum values of the execution time of one experiment are always higher than a minute. This phenomenon was investigated and it was found that in high load situations, TOTD occasionally stopped responding for about a minute and continued the operation afterwards. TOTD produced similar behavior under all the three operating systems. It was also checked that the authoritative DNS server at **`teacherb`** was still responsive. So without the limitation of the load, TOTD is not safe to be used in a production system.

We believe that due to its excellent average performance, TOTD would deserve a thorough code review and bug fix.

2) OpenBSD, TOTD is a forwarder

The performance results of the DNS64 server realized by TOTD used as a forwarder running under OpenBSD were summarized in Table VIII. Comparing TOTD running under OpenBSD to TOTD running under Linux, it can be said that the average performance results are similar but its stability is somewhat better under OpenBSD than under Linux; it produced acceptable maximum execution time values for two clients under OpenBSD. It can also be observed that the maximum values of the execution time of one experiment are less than a minute under OpenBSD (but they are still too high, so is the standard deviation). For these reasons, if one chooses to use TOTD with load limitation, it is worth using it under OpenBSD rather than under Linux.

3) FreeBSD, TOTD is a forwarder

The performance results of the DNS64 server realized by TOTD used as a forwarder running under FreeBSD were summarized in Table IX.

TABLE VI. DNS64 PERFORMANCE: BIND, FREEBSD, FORWARDER

	Number of clients	1	2	4	8
2	Exec. time of 256 `host` commands (s)	1.711	2.831	5.475	11.126
3	standard deviation	0.014	0.037	0.082	0.099
4	maximum value	1.790	2.950	5.590	11.390
5	CPU utilization (%)	77.15	97.32	100.00	100.00
6	standard deviation	1.8	1.5	0.0	0.0
7	DNS64 memory consumption (MB)	31	37	39	37
8	Number of requests served (request/s)	150	181	187	184

TABLE VII. DNS64 PERFORMANCE: TOTD, LINUX, FORWARDER

	Number of clients	1	2	4	8
2	Exec. time of 256 `host` commands (s)	0.791	1.310	2.158	4.348
3	standard deviation	0.038	3.863	3.754	5.301
4	maximum value	1.370	64.950	63.730	68.540
5	CPU utilization (%)	38.00	58.05	80.16	84.79
6	standard deviation	2.4	27.1	27.5	29.2
7	DNS64 memory consumption (MB)	1.0	1.1	1.6	0.8
8	Number of requests served (request/s)	324	391	474	471

TABLE VIII. DNS64 PERFORMANCE: TOTD, OPENBSD, FORWARDER

	Number of clients	1	2	4	8
2	Exec. time of 256 `host` commands (s)	0.808	1.118	2.276	4.445
3	standard deviation	0.010	0.141	2.724	5.468
4	maximum value	0.830	3.060	28.560	41.080
5	CPU utilization (%)	48.99	79.98	76.51	78.34
6	standard deviation	2.2	6.6	13.4	11.4
7	DNS64 memory consumption (MB)	0.2	0.1	1.1	0.4
8	Number of requests served (request/s)	317	458	450	461

TABLE IX. DNS64 PERFORMANCE: TOTD, FREEBSD, FORWARDER

1	**Number of clients**	**1**	**2**	**4**	**8**
2	**Exec. time of 256 `host` commands (s)**	0.923	1.127	2.580	4.750
3	**standard deviation**	0.676	0.061	6.356	6.181
4	**maximum value**	5.800	1.300	65.650	69.700
5	**CPU utilization (%)**	45.01	78.66	69.54	83.23
6	**standard deviation**	6.4	5.2	41.2	34.2
7	**DNS64 memory consumption (MB)**	3.0	0.7	0.8	0.9
8	**Number of requests served (request/s)**	277	454	397	431

It can be observed that the average performance results of TOTD are quite similar under all the three platforms. For two clients, TOTD under FreeBSD even outperformed TOTD under Linux by serving 454 requests vs. 391 requests.

However, the validity of these numbers is rather low, because the average values are highly influenced by the phenomenon of the unresponsiveness of TOTD. The numbers of the served requests for one client are much more faithfully representing the performance of TOTD under the two platforms: it can serve 324 requests per second under Linux and only 227 requests per second under FreeBSD.

The authors have another recommendation about TOTD. As this tiny DNS64 solution used very little memory under all the three operating systems and it outperformed BIND and worked well while the load was limited, TOTD is a good candidate for a DNS64 solution on the client side. The load of one computer is surely limited and its low memory and CPU requirements make it the best choice for a client side DNS64 service.

VII. CONCLUSIONS

BIND9 running under Linux showed the best overall performance characteristics from among the tested DNS64 solutions. It was stable under serious overload conditions and its memory requirement was found moderate. BIND9 running under Linux is our number one recommendation for a DNS64 solution in a production system.

BIND9 running under FreeBSD was very stable but showed less good performance than under Linux, due to running in jail. It can also be a choice if FreeBSD platform is preferred for security reasons.

The memory usage of TOTD running under all the three operating systems was very low thus it can be a good DNS64 solution for small embedded systems but it needs an external tool for load limitation otherwise it may lose its stability.

Due to its average performance, TOTD would deserve a thorough code review and bug fix. It is also a good candidate for a client side DNS64 solution.

ACKNOWLEDGMENTS

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.

The publication of this paper was supported by the TÁMOP-4.2.2/B-10/1-2010-0010 project and by the Széchenyi István University (15-3202-08).

REFERENCES

[1] 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)

[2] 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)

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

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

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

[6] C. Aoun and E. Davies, "Reasons to move the Network Address Translator - Protocol Translator (NAT-PT) to historic status", IETF, July 2007. (RFC 4966)

[7] P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) terminology and considerations", IETF, August 1999. (RFC 2663)

[8] 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)

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

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

[11] 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

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

[13] G. Lencse and G Takács, "Performance analysis of DNS64 and NAT64 solutions", Infocommunications Journal, vol. 4. no 2.

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

[15] Open Source Initiative, "The open source definition", http://opensource.org/docs/osd

[16] Internet Systems Consortium, "Berkeley Internet Name Daemon (BIND)" , https://www.isc.org/software/bind

[17] Feike W. Dillema, "Trick Or Treat Daemon (TOTD)", http://www.dillema.net/software/totd.html

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

[19] FreeBSD Handbook, Chapter 16 Jails http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html

[20] G. Lencse and S. Répás, "Performance analysis and comparison of the TAYGA and of the PF NAT64 implementations", unpublished