Performance analysis and comparison of four DNS64 implementations under different free operating systems

G. Lencse[1] · S. Répás[1]

Abstract The depletion of the global IPv4 address pool made the deployment of IPv6, the new version of the Internet Protocol, inevitable. In this paper, the transition mechanisms for the first phase of IPv6 deployment are surveyed and the DNS64 plus NAT64 solution is found appropriate. The most important free and open source DNS64 implementations are selected: BIND, TOTD, Unbound and PowerDNS. The test environment and the testing method are described. The first three of the selected DNS64 implementations are tested under Linux, OpenBSD and FreeBSD whereas PowerDNS is tested only under Linux. Their performance characteristics (response time, number of answered requests per second, CPU and memory consumption) are measured and compared. The effect of the hardware architecture of the test computer is also examined by using single-core, dual-core and quad-core test computers. The stability of all the tested DNS64 solutions are analyzed under overload conditions to test if they may be used in production environments with strong response time requirements. Our measurement results show significant differences in the performance of the tested DNS64 implementations, e.g. Unbound served four times more requests per second than PowerDNS (when executed by a single-core CPU under Linux and load was generated by eight clients). However, no absolute order can be determined, because it is influenced by different factors such as the architecture of the hardware, especially the number of cores, because BIND and PowerDNS are multithreaded (therefore they can profit from the multiple cores) but TOTD and Unbound are not. Also the operating system of the DNS64 server has significant influence on the performance of the DNS64 implementations under certain conditions. All the details of our measurements are disclosed and all the results are presented in the paper. An easy-to-use implementation selection guide is also provided as a short summary of our high number of results.

Keywords DNS64 · Internet · IPv6 deployment · IPv6 transition solutions · Performance analysis

G. Lencse
lencse@sze.hu

S. Répás
repas.sandor@sze.hu

[1] Department of Telecommunications, Széchenyi István University, 1 Egyetem tér, H-9026 Győr, Hungary

Submitted: 19 August, 2015, Revised: 27 November 2015.

1 Introduction

In the following years the Internet service providers (ISPs) will not be able to provide public IPv4 addresses to their high number of new clients because of the depletion of the global IPv4 address pool. Either they use carrier grade NAT (CGN) or they provide IPv6 addresses. The latter one is better for the long run, but it results in the issue that the IPv6 only clients have to be enabled to communicate with IPv4 only servers which ones are in majority in the Internet today. The authors believe that from among the several solutions proposed for this problem, the combination of DNS64 [1] and NAT64 [2] is the best available method that enables an IPv6 only client to communicate with an IPv4 only server.

Four free DNS64 implementations were found and 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. We also hope that the performance measurement methods we have developed may be useful for other researchers too.

The performance analysis and comparison of some selected NAT64 implementations under Linux and OpenBSD was also a part of our research. Our preliminary results are available in [3] and [4]. We plan to test further NAT64 implementations too.

The remainder of this paper is organized as follows. In section II 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. In section III the selection of the DNS64 implementations is discussed. The test environment and the performance measurement method are described in sections IV and V, respectively. In section VI the presentation method of the results is given. The DNS64 performance results produced by the single, dual and quad core test computers are presented and discussed in sections VII, VIII and IX, respectively. An easy-to-use implementation selection guide and our future plans are disclosed in sections X and XI, respectively. Section XII concludes our work.

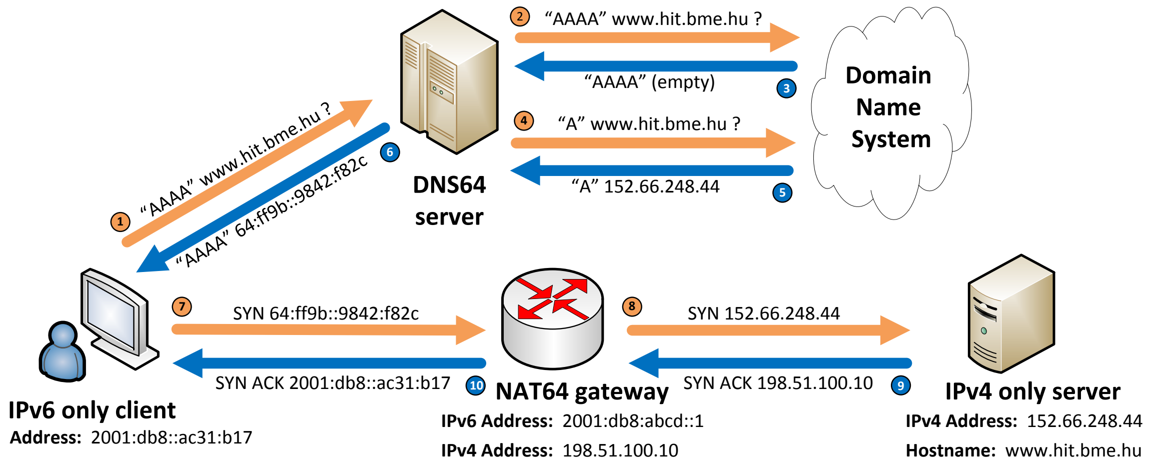

Fig. 1 The operation of the DNS64+NAT64 solution: an IPv6 only client communicates with and IPv4 only server [10]

2 IPv6 Transition Mechanisms for the First Phase of IPv6 Deployment

2.1 Most Important Solutions

The authors believe that the deployment of IPv6 will take place over a long period of time and the two versions will have to coexist for the foreseeable future 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 (because the new clients can get only IPv6 addresses due to the depletion of the public IPv4 address pool and a high number of servers will still use only IPv4 in the upcoming years). Several transition 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.

Our position can also be justified by [8]. They state: "The mainstream of translation techniques is network translation. Among the network translation mechanisms, IVI is a feasible stateless translation mechanism, and NAT64 is a feasible stateful translation mechanism." They do not mention any other feasible stateful methods and the applicability of the stateless one is apparently very limited because of the depletion of the public IPv4 address pool.

Reference [9] gives an up to date survey of the IPv4 address sharing methods, and concludes that: "The only actual address sharing mechanism that really pushes forward the transition to IPv6 is Stateful NAT64 (Class 4). All other (classes of) mechanisms are more tolerant to IPv4."

2.2 Operation of DNS64+NAT64

We demonstrate the operation of the DNS64 + NAT64 IPv6 transition solution using the example of an IPv6 only client and an IPv4 only web server taken from [10]. In this example, the DNS64 server uses the 64:ff9b::/96 *NAT64 Well-Known Prefix* [11] for generating *IPv4-embedded IPv6 address*es [11]. In a real-life solution, usually a *network specific prefix* from the network of the ISP of the client is used instead of 64:ff9b::/96.

There are two prerequisites for the proper operation:

1. A DNS64 server should be set as the DNS server of the IPv6 only client.
2. Packets towards the 64:ff9b::/96 network (or towards the selected network specific prefix) should be routed to a NAT64 gateway (routing must be configured that way).

Now let us follow the steps of the communication (taken verbatim from our conference paper [10]):

1. The client asks its DNS server (which one is actually a DNS64 server) about the IPv6 address of the **`www.hit.bme.hu`** web server.
2. The DNS64 server asks the DNS system about the IPv6 address of **`www.hit.bme.hu`**.
3. No IPv6 address is returned.
4. The DNS64 server then asks the DNS system for the IPv4 address of **`www.hit.bme.hu`**.
5. The 152.66.148.44 IPv4 address is returned.
6. The DNS64 server synthesizes an *IPv4-embedded IPv6 address* by placing the 32 bits of the received 152.66.148.44 IPv4 address after the 64:ff9b::/96 prefix and sends the result back to the client.
7. The IPv6 only client sends a TCP SYN segment using the received 64:ff9b::9842:f82c IPv6 address and it arrives to the IPv6 interface of the NAT64 gateway (since the route towards the 64ff9b::/96 network is set so in all the routers along the path).
8. The NAT64 gateway constructs an IPv4 packet using the last 32 bits (0x9842f82c) of the destination IPv6 address as the destination IPv4 address (this is exactly 152.66.248.44), its own public IPv4 address (198.51.100.10) as the source IPv4 address and some other fields from the IPv6 packet plus the payload of the IPv6 packet. It also registers the connection into its connection tracking table (and replaces the source

port number by a unique one if necessary). Finally it sends out the IPv4 packet to the IPv4 only server.

9. The server receives the TCP SYN segment and sends a SYN ACK reply back to the public IPv4 address of the NAT64 gateway.
10. The NAT64 gateway receives the IPv4 reply packet. It constructs an appropriate IPv6 packet using the necessary information from its state table. It sends the IPv6 packet back to the IPv6 only client.

For a more detailed but still easy to follow introduction, see [12] and for the most accurate and detailed information, see the relating RFCs: [1] and [2].

We note that Section 7 of [1] describes three different scenarios concerning where the DNS64 server is deployed. In this paper, we consider the one described in subsection 7.1 that is *"an IPv6 network to the IPv4 Internet" setup with DNS64 in DNS server mode*, where the DNS64 server is placed in client side and it is used in DNS server mode (not in stub resolver mode).

2.3 Survey of the Current Research Results

Several papers were published in the topic of the performance of DNS64 and NAT64 since 2012. The performance of the TAYGA NAT64 implementation (and implicitly of the TOTD DNS64 implementation) is compared to the performance of NAT44 in [13]. 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 [14]. 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 [15]. 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 therefore 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 [16]. However, only one implementation was considered for each service, so even if they were proved to be stable and fast enough, more research was needed for the comparison of the performance (and also the stability) of multiple DNS64 and NAT64 implementations.

A good survey of the most recent DNS64 and NAT64 research results is given in [17]. They also compared the CPU consumption of DNS64 to that of DNS as well as the CPU and memory requirements of NAT64 to that of NAT and they concluded that the DNS64+NAT64 system is an affordable solution for an Internet service provider. However, the stability of the different DNS64 and NAT64 implementations under heavy load conditions and the comparison of their performance were not addressed there.

The results of our DNS64 tests concerning BIND and TOTD were published in [18]. TOTD was found to be faster than BIND. However, TOTD was not stable due to an implementation bug that was described and eliminated in [19]. Besides the correction, a significant security improvement was performed on TOTD. Our patch was included by its developer into the 1.5.3 version of TOTD which is available from [20]. This version is tested in our current experiments. As for further improvements over our conference paper [18] on which our current paper is based, two further DNS64 implementations are tested (see the next section), different architecture test computers are used and the measurements are made more accurate by using automation now.

This paper deals with DNS64 only. Our preliminary research results concerning two NAT64 implementations (the stateless TAYGA + iptables under Linux and the stateful PF of OpenBSD) were published in [3] and [4]. Some further experiments are still to be performed with these and also other NAT64 implementations.

3 Selection of DNS64 Implementations

Only free software [21] (also called open source [22]) implementations were considered. We had multiple reasons for this decision:

- The licenses of certain vendors (e.g. [23] and [24]) do not allow the publication of benchmarking results.
- Free software can be used by anyone for any purposes thus our results can be helpful for anyone.
- Free software is free of charge for us, too.

(This reasoning was first published in [25].)

BIND [26] was a natural choice for our implementation, since it is the most widely used DNS implementation and it contains native DNS64 support from version 9.8.

BIND is a large and complex software containing all the different DNS functionalities (authoritative, recursive, DNSSEC support, etc.). Our second choice was a lightweight one, namely TOTD, which was implemented by Feike W. Dillema as a part of the 6net project [27]. Its original version used sequential transaction IDs (the generation of them contained a trivial programming bug, which was discovered and corrected in [19]) in the DNS messages therefore it was vulnerable to cache poisoning using the transaction ID attack. "This vulnerability was patched by a very computation efficient solution using random permutations and alternating ranges. The performance price of the increased security was found to be practically invisible." [19] This new version of TOTD was used in our experiments.

Unbound [28] was our third choice. As its name suggests, it was designed to be an alternative to BIND providing both better performance and security [29]. Reference [30] states (on page 553) that Unbound is significantly faster than BIND. Note that the DNS64 patches for BIND and Unbound were developed in the same Ecdysis project [31].

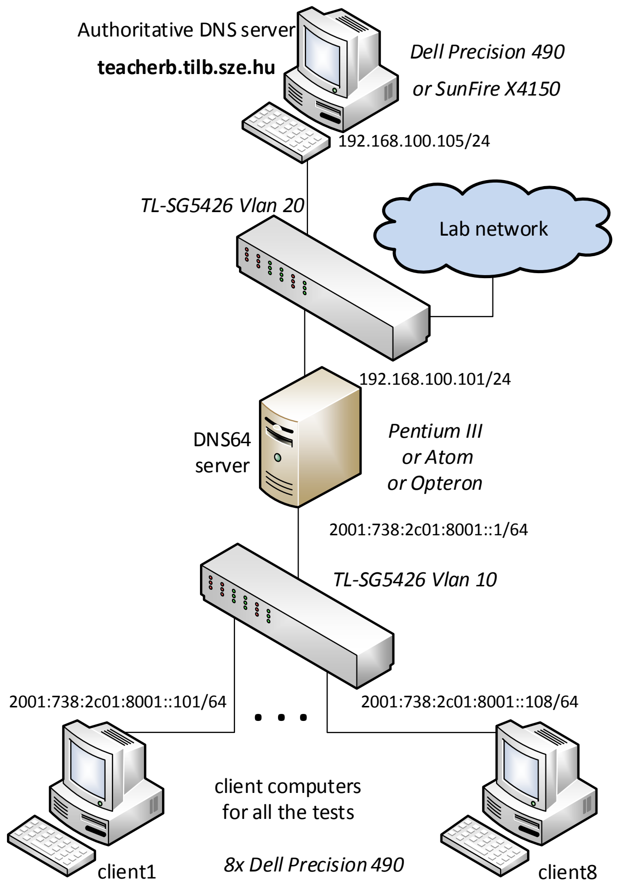

Fig. 2 Topology of the DNS64 test network for the measurements

Fourth, we also selected PowerDNS [32] for testing. It is also free software under GPL but its developers offer commercial support, too. It was named the third most popular DNS server on the Internet in 2008 [33].

We found no more free DNS64 implementations.

All four DNS64 implementations were intended to be tested under all the three free operating systems which are the most typical ones for this purpose, namely: Linux, OpenBSD and FreeBSD.

4 Test Environment for DNS64 Performance Measurements

4.1 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. 2. The central element of the test network is the DNS64 server.

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 `teacherb.tilb.sze.hu` 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.

4.2 Configuration of the Computers

Three test computers with special configuration were put together for the purpose of the DNS64 server. First, 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. Later on dual- and quad-core computers were selected for testing to find out how the examined implementations can benefit from the multi-core CPUs, which are dominant today.

Our first choice for the test computer was an old IBM eServer xSeries 200. Its old 9.1GB SCSI disk was replaced by a new SSD to be able to store the data during the measurements easily, and two identical gigabit Ethernet NICs were added. The configuration of the test computer was:

- 694X-686A motherboard
- 800MHz Intel Pentium III with 256 kB L2 cache and MMX technology CPU
- 128MB, 133MHz, 60ns ECC SDRAM
- BestConnection PCI SATA Raid controller + SATA SSD converter (to connect the SSD)
- KF1310MCJ14 32GB SSD
- Two Intel® PRO/1000 GT Desktop Adapter 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.

Our second choice for the test computer was an Intel Atom D525 based computer:

- Intel D525MW motherboard with integrated 1.8GHz dual core Intel Atom D525 CPU with 1MB L2 cache
- 2x 2GB, 800MHz, non-ECC DDR3 SDRAM
- KF1310MCJ14 32GB SSD
- One integrated and one Mini PCIe Realtek RTL8111DL Gigabit Ethernet NICs

Our third choice was a Sun Fire X4200 M2 server:

- Sun Microsystems Sun Fire X4200 Server motherboard with four integrated Intel 82546EB Gigabit Ethernet Controllers
- Two 2.2GHz Dual Core AMD Opteron 275 CPUs with 1MB L2 cache
- 4x 1GB 400MHz ECC DDR SDRAM
- SATA SSD converter (to connect the SSD)
- KF1310MCJ14 32GB SSD

For all the other purposes (the 8 client computers and for the authoritative DNS server for the measurements with the first two test computers with the exception of the CPUs) standard DELL Precision Workstation 490 computers were used with the following configuration:

- DELL 0GU083 motherboard with Intel 5000X chipset
- Two Intel Xeon 5140 2.3GHz dual core processors
- 4x 1GB 533MHz DDR2 SDRAM (accessed quad channel)
- Broadcom NetXtreme BCM5752 Gigabit Ethernet controller (PCI Express)

Note that the configuration of these computers was slightly changed since the measurements were done for [18].

A workstation of the same type but with somewhat faster CPUs was used as the authoritative DNS server for the measurements with the first two test computers. The CPUs were:

- Two Intel Xeon 5160 3.0GHz dual core processors

And a SunFire X4150 Sun Server was used as the authoritative DNS server for the measurements with the third test computer:

- Sun Microsystems S92 motherboard
- Two Quad Core Intel Xeon E5440 2.83GHz CPU
- 8GB DDR2 800MHz RAM
- Two near-line SAS 160GB HDDs
- Two Intel 82571EB Gigabit Ethernet NICs
- Two Intel 80003ES2LAN Gigabit Ethernet NICs (one of them was used for the measurements)

Debian Wheezy 7.6 GNU/Linux operating system was installed on all the computers, including the test computers when they were used under Linux, but excluding the authoritative DNS servers which had version 7.1. The version of the OpenBSD and FreeBSD operating systems installed on the test computers were 5.5 and 10.0, respectively. The 64-bit computers always had the 64-bit version of the given operating systems.

5 DNS64 Performance Measurement Method

5.1 IPv4 DNS Server Settings

The authoritative DNS server `teacherb.tilb.sze.hu` used the 192.168.100.105 IP address. BIND was used for authoritative name server purposes in all the DNS64 experiments. The version of BIND was 9.8.4 as this one can be found in the Debian Wheezy 7.1 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 (i.e. IPv4 addresses) and no “AAAA” records (i.e. IPv6 addresses), so the generation of the IPv6 addresses was the task of the DNS64 server.

5.2 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.

Both operation modes may be relevant in production systems. On the one hand, it is often desirable to work as a recursor in order to do the whole job without the help of another server. On the other hand security policy may require the DNS64 server to work as a forwarder and only the standard caching DNS server is given the right to query other name servers on the Internet. Thus we need to find the best solution in both operation modes.

Whereas BIND and PowerDNS can be either of them, TOTD can act only as a forwarder and Unbound (at least the version we tested) can provide DNS64 functionality only in the case if it is started as a recursor. Therefore we measured the performance of the tested DNS64 implementations in all their possible operation modes.

5.3 DNS64 Server Settings

The first three of the selected DNS64 implementations were tested under Linux, OpenBSD and FreeBSD whereas PowerDNS was tested only under Linux.

5.3.1 Preparation of the Linux test system

The network interfaces of the freshly installed Debian Wheezy Linux operating system on the test computer were set according to Fig. 2.

Netfilter (iptables) was not used during the measurements. To see the possible issues and their solutions using netfilter, see [18].

5.3.2 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. 2.

PF was not installed on the FreeBSD system.

On the OpenBSD system the state keeping was switched off by the following line in `/etc/pf.conf`:

```
pass no state
```

In this way, PF does not record the state of any requests and answers.

5.3.3 Set up of the BIND DNS64 server

The BIND 9.8.5-P2 was compiled from source under Linux and OpenBSD. FreeBSD version 10.0 already contained the 9.8.7-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 `/etc/bind/named.conf.options` file. Now, BIND was ready to operate as a recursor. BIND was also set as a forwarder by the following additional settings in the `named.conf` file:

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

5.3.4 Set up of the TOTD DNS64 server

TOTD 1.5.3. including our security enhancement patch was used. 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:ffff::
retry 300
```

5.3.5 Set up of the Unbound DNS64 servers

Unbound v1.4.20 with ecdysis patch was used. The server section of the `unbound.conf` configuration file contained:

```
private-domain: “sze.hu”
module-config: “dns64 validator iterator”
dns64-prefix: 2001:738:2c01:8001:ffff:ffff::/96
```

Unbound does not provide the DNS64 functionality when it is set up as a forwarder, thus it was tested only as a recursor.

5.3.6 Set up of the PowerDNS DNS64 server

PowerDNS Recursor v3.5.2 was used. It worked only under Linux. (As its performance was the lowest from among the four implementations during our preliminary tests executed by the Pentium III computer, we gave up compiling it under the BSD systems.)

The settings in the `recursor.conf` file were:

```
lua-dns-script=/etc/powerdns/dns64.lua
dns64.lua:
function nodata ( remoteip, domain, qtype, records )
    if qtype ~= pdns.AAAA then return -1, {} end  \
        --  only AAAA records
    setvariable()
    return "getFakeAAAARecords", domain, \
        "2001:738:2c01:8001:ffff:ffff::"
end
function endswith(s, send)
    return #s >= #send and s:find(send, #s-#send+1, true)
and true or false
end
```

5.3.7 Client Settings

Debian Wheezy 7.6 was installed on the DELL computers used for client purposes. 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
```

5.4 DNS64 Performance Measurements

5.4.1 Elementary measurement scripts

The CPU and memory consumptions of the DNS64 server were 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
for b in {0..255}
do
    /usr/bin/time -f "%E" -o dns64-$i.txt \
       -a ./dns-st-c.sh $i $b
done
```

The *synchronized start* of the client scripts was done by using broadcast, see the details later on.

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 -t AAAA 10-$1-$2-$c.zonat.tilb.sze.hu &
    host -t AAAA 10-$1-$2-$((c+1)).zonat.tilb.sze.hu &
    host -t AAAA 10-$1-$2-$((c+2)).zonat.tilb.sze.hu &
    host -t AAAA 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.)

Note that in [18] we did not use the `-t AAAA` option and thus then also the `MX` record was requested by the `host` command. But now we focused on the `AAAA` record only (as usually only this one is relevant, e.g. when browsing the web).

However, our client computers were not powerful enough to be able to overload the Sun test computer (having four cores) using the above `dns-st-c.sh` bash script. Thus we wrote a C program that sent 64 DNS queries and it was started in four instances (to utilize the 4 cores of the clients) in the measurements with the Sun computer by the following `dns-st-c-4.sh` bash script:

```
#!/bin/bash
dns-st-c $1 $2 0 64 5 &
dns-st-c $1 $2 64 64 5 &
dns-st-c $1 $2 128 64 5 &
dns-st-c $1 $2 192 64 5
```

The C program took five parameters. It requested `argv[4]` number of AAAA records of symbolic names in the `zonat.tilb.sze.hu` zone starting the first label (hostname) from `10-argv[1]-argv[2]-argv[3]` and using consecutive integers in the place of `argv[3]`. After sending a query, it always waited for the answer and continued after receiving it or the time out value given in `argv[5]` (measured in second). Its source code is not included because of its size, but it can be downloaded from: `http://ipv6.tilb.sze.hu/STS-DNS64/` (capitalization matters).

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. As for measuring the CPU utilization, not the direct CPU usage of the DNS64 server process was measured, because that method would leave out the CPU usage of some work from the accounting that was done not directly by the server process rather by the kernel but served the interest of the DNS64 service, e.g. processing packet headers. In the same way, the memory consumption of the DNS64 server process was calculated as the largest decrease of the free memory during the measurements. We admit that our method may also include the CPU and memory usage of other tasks too, but we considered it a less critical problem. Thus we measured and upper bound for both CPU and memory utilization.

Under Linux, the following command line was used:

```
nice -n 10 dstat -T -t -c -m -p -i -I 44,45 -n -N \
    eth1,eth2,total -d --unix  --output load.csv
```

Under the BSD operating systems, the command line was:

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

5.4.2 Automatic execution

The execution of the measurements was automated for both achieving higher accuracy and sparing human work time. The netcat [34] utility was selected for this purpose. Netcat can start the measurement program, when a packet is received on a given port. The packet can be TCP, UDP and UDP broadcast, too. The broadcast method was used for the synchronized start of all of the clients. The experiments with 1, 2, 4 and 8 parallel clients were executed 8, 4, 2 and 1 times, respectively. All the times the load was provided by different set of clients, to make the results more precise. The time of all of the computers were synchronized by NTP (Network Time Protocol) [35] for the accurate time measurement.

6 Method of the Presentation of our Results

Our measurements produced a huge number of results. The result space can be explored along different axes:

- The type of the DNS64 implementations (BIND, TOTD, Unbound, PowerDNS)
- The type of operating systems (Linux, OpenBSD, FreeBSD)
- The mode of operation (recursor, forwarder)
- The number of clients (1, 2, 4, 8)
- The CPU architecture of the test computers (Pentium III, Atom, Opteron).

However, not all the combinations are valid:

- TOTD can act only as a forwarder
- Unbound can do DNS64 only in the case if it is used as a recursor
- PowerDNS was tested only under Linux.

Our analysis will be done in the following order. First, we analyze the DNS64 implementations by themselves. That is, we take an implementation and analyze its stability (examine its behavior under overload) and then examine how it behaves under different operating systems. Second, we compare the implementations to each other. For this reason, we perform the one by one analysis of the DNS64 implementations in two separate groups: the forwarders (BIND, TOTD, PowerDNS) and the recursors (BIND, Unbound, PowerDNS). Third, we examine the possible effect of the CPU architecture of the test computer. For this reason, we perform the above analysis using the results of the Pentium III CPU first, and deal with their performance results produced by the Atom and Opteron CPUs later on.

We use tables (and not graphs) as the measured quantities are of different types which could not be plotted together. Therefore, we find tables more space effective than graphs. All the tables follow the same format thus we give a detailed explanation for the first one only; the others are to be interpreted in the same way.

The three tables of each groups (forwarders or recursors) using the same CPU architecture are put on the same page for the synoptic view and easy comparison of the results.

7. Results of the Measurements on Pentium III

7.1 Performance Results of BIND, Forwarder

The performance results of the DNS64 server realized by BIND used as a forwarder and executed by the Pentium III test computer were summarized in Table 1. The first row of the table specifies the operating system of the test computer. The second row of the table shows the number of clients. (The offered load of the DNS64 server was proportional to this parameter.) The third, fourth and fifth rows show the average, the standard deviation and the maximum value of the

Table 1 DNS64 Performance: BIND, Forwarder, Pentium III

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	1.25	1.38	2.69	5.31	1.19	1.28	2.44	4.85	1.41	1.68	3.39	6.68
4	**256 `host`**	std. deviation	0.01	0.02	0.04	0.07	0.01	0.02	0.03	0.09	0.02	0.02	0.04	0.05
5	**commands (s)**	maximum	1.32	1.47	2.78	5.41	1.28	1.43	2.52	4.98	1.49	1.81	3.50	6.78
6	**CPU utiliza-**	average	51.03	93.26	99.17	99.60	48.34	89.93	99.42	99.59	58.18	98.65	99.41	99.67
7	**tion (%)**	std. deviation	6.94	12.07	8.82	6.17	3.32	4.10	7.39	6.34	3.63	2.45	7.57	5.53
8	**Memory consumption (MB)**		37.238	57.254	57.289	57.262	30.695	51.855	52.605	51.047	40.496	55.004	56.508	58.688
9	**Number of requests served in a second (request/s)**		204	371	380	386	215	401	420	422	182	306	302	307

Table 2 DNS64 Performance: TOTD, Forwarder, Pentium III

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	0.85	0.89	1.06	2.03	0.87	0.95	1.37	2.54	0.86	0.89	1.10	1.98
4	**256 `host`**	std. deviation	0.01	0.02	0.02	0.06	0.01	0.02	0.08	0.42	0.01	0.02	0.04	0.10
5	**commands (s)**	maximum	0.89	0.94	1.14	2.09	0.95	1.11	1.53	6.35	0.90	1.07	1.24	2.09
6	**CPU utiliza-**	average	13.44	34.18	83.09	98.97	18.81	42.89	69.93	83.62	17.08	36.90	69.66	82.26
7	**tion (%)**	std. deviation	2.89	5.92	13.45	9.75	2.49	4.08	3.82	2.00	2.54	3.99	4.12	2.18
8	**Memory consumption (MB)**		1.516	1.016	1.734	1.563	2.469	3.438	2.723	1.258	2.500	2.434	2.746	0.949
9	**Number of requests served in a second (request/s)**		301	578	967	1010	293	540	749	806	298	576	934	1034

Table 3 DNS64 Performance: PowerDNS, Forwarder, Pentium III

1	**Operating System**		**Linux**			
2	**Number of clients**		**1**	**2**	**4**	**8**
3	**Exec. time of**	average	1.42	1.91	3.31	6.65
4	**256 `host`**	std. deviation	0.02	0.05	0.06	0.12
5	**commands (s)**	maximum	1.48	2.05	3.48	6.92
6	**CPU utiliza-**	average	57.32	85.66	99.33	99.67
7	**tion (%)**	std. deviation	7.18	9.49	7.97	5.59
8	**Memory consumption (MB)**		22.277	59.117	54.359	59.203
9	**Number of requests served in a second (request/s)**		180	268	309	308

execution time of the execution of 256 `host` commands (this is called one experiment), respectively.

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

Row number eight shows the estimated memory consumption of DNS64. (This parameter can be measured with high uncertainty, because other processes than DNS64 may also influence the size of free/used memory of the test computer.)

The *N* number of DNS64 requests per second, served by the test computer, was calculated according to (1) using the number of clients (in row 2) and the average execution time values (in row 3) and it is displayed in the last row of the table.

$$N = \frac{256 * Number\_of\_clients}{Average\_exec\_time\_of\_256\_host\_cmds} \quad (1)$$

Now we discuss the results separately for Linux, OpenBSD and FreeBSD.

7.1.1 Linux

As for the results, BIND shows stability in all measured values. Execution time results show very little (relative) deviation and the maximum values are always close the average at any number of clients. The increase of the load does not cause 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 a little bit less than *linearly* with the load (that is, with the number of clients): the average execution time is 2.69 and 5.31 seconds for 4 and 8 clients respectively whereas the maximum values are 2.78 and 5.41 (which is less than the double of 2.78). The number of requests served per second shows a small increase from 380 to 386 in this serious overload situation. Therefore, we can state that the behavior of the DNS64 system realized by BIND as a forwarder running under Linux complies with the so called graceful degradation [36] principle; if there are not enough resources for serving the requests then the response time of the system increases only linearly with the load.

Also the memory consumption of BIND is visibly moderate (less than 58MB) even for very high loads.

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

7.1.2 OpenBSD

By observing the results, we can state that BIND under OpenBSD shows similar stability than under Linux and gives somewhat better performance (at eight clients, it served 422 requests per second instead of 386).

7.1.3 FreeBSD

Even though some fluctuations can be observed in the request served in a second (306, 302, 307 for 2, 4, 8 clients, respectively), we can still state that BIND under FreeBSD also complies with the graceful degradation principle.

7.1.4 BIND as a forwarder under different operating systems

As for the stability of BIND, it may be used in production

Table 4 DNS64 Performance: BIND, Recursor, Pentium III

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	1.32	1.50	2.98	5.98	1.22	1.31	2.54	5.19	1.44	1.73	3.43	6.94
4	**256 `host`**	std. deviation	0.02	0.03	0.05	0.07	0.01	0.02	0.04	0.07	0.02	0.03	0.04	0.06
5	**commands (s)**	maximum	1.38	1.60	3.09	6.07	1.34	1.44	2.60	5.30	1.55	1.82	3.55	7.06
6	**CPU utiliza-**	average	54.04	95.27	99.33	99.67	49.27	91.54	99.32	99.62	58.97	97.98	99.41	99.69
7	**tion (%)**	std. deviation	6.79	11.36	8.03	5.72	3.45	4.72	8.12	6.14	3.46	5.48	7.58	5.44
8	**Memory consumption (MB)**		35.570	53.059	52.164	52.922	30.438	50.488	51.047	48.828	40.195	53.738	55.426	65.801
9	**Number of requests served in a second (request/s)**		193	342	343	343	211	389	403	394	178	296	299	295

Table 5 DNS64 Performance: Unbound, Recursor, Pentium III

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	0.91	0.93	1.02	1.69	0.93	0.96	1.09	1.99	0.95	0.99	1.18	2.14
4	**256 `host`**	std. deviation	0.01	0.02	0.02	0.03	0.01	0.02	0.02	0.02	0.01	0.02	0.02	0.02
5	**commands (s)**	maximum	0.94	0.99	1.10	1.74	0.98	1.05	1.22	2.03	1.02	1.16	1.27	2.21
6	**CPU utiliza-**	average	24.19	46.79	82.73	98.37	27.94	53.33	91.53	99.81	30.81	59.07	95.34	99.64
7	**tion (%)**	std. deviation	4.19	7.37	13.27	12.03	3.02	3.78	2.46	3.97	5.81	7.58	1.88	5.65
8	**Memory consumption (MB)**		14.586	15.039	14.512	14.879	21.270	20.402	22.859	20.594	20.039	20.367	18.648	18.668
9	**Number of requests served in a second (request/s)**		282	551	999	1211	276	533	944	1032	269	518	866	959

Table 6 DNS64 Performance: PowerDNS, FreeBSD, Recursor, Pentium III

1	**Number of clients**		**1**	**2**	**4**	**8**
2	**Exec. time of**	average	1.44	1.96	3.43	6.90
3	**256 `host`**	std. deviation	0.02	0.05	0.06	0.13
4	**commands (s)**	maximum	1.50	2.11	3.63	7.17
5	**CPU utiliza-**	average	58.01	85.96	99.34	99.68
6	**tion (%)**	std. deviation	7.10	9.43	7.98	5.43
7	**Memory consumption (MB)**		22.574	37.105	54.777	59.457
8	**Number of requests served in a second (request/s)**		177	262	298	297

systems under all the three operating systems. It showed its best performance under OpenBSD by serving 422 requests per second for eight clients, but its performance under Linux was close to it (386 requests/s). It produced its poorest performance results under FreeBSD (307 requests/s). But this performance sacrifice may be acceptable in some cases e.g. for security, as FreeBSD is the only operating system from the three ones that makes it possible to execute the server programs in a jail environment [37].

7.2 Performance Results of TOTD, Forwarder

The performance results of the DNS64 server realized by TOTD used as a forwarder were summarized in Table 2. The eye catching 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, thus the lack of caching caused no performance penalty. However, in a real life system, the average performance of TOTD might be worse than BIND which uses caching. But the very low memory consumption of TOTD can be an advantage in a small embedded system.

Note that memory consumption values of TOTD are to be considered as order of magnitude estimations only (and thus may be paraphrased as "a few megabytes") because of the before mentioned high uncertainty of the measurements.

7.2.1 Linux

TOTD is stable under Linux and it provides excellent performance (1010 requests/s at 8 clients).

7.2.2 OpenBSD

As for the execution time of one experiment at eight clients, the standard deviation (0.42s) is about 16.5% of the average (2.54s) and the maximum value is 6.35s. Whereas they are not unacceptable, we consider them as warnings about the stability. The fact that the CPU utilization could not approximate 100% while the number of requests served in a second was 749 at four clients and it could grow to 806 only at eight clients is considered another warning sign. Therefore, we do not recommend OpenBSD for TOTD.

7.2.3 FreeBSD

Even though the CPU utilization values of TOTD under FreeBSD are very similar to that of TOTD under OpenBSD, the stability of TOTD under FreeBSD is unquestionable on the basis of the execution time of one experiment: the standard deviation (0.1s) is very low compared to the average (1.98s), and the maximum value (2.09s) is very close to the average. The performance of TOTD is outstanding: it can serve 1034 requests per second.

7.2.4 TOTD as a forwarder under different operating systems

As for the stability and performance of TOTD, it may be used in production systems under both Linux and FreeBSD. It showed its best performance under FreeBSD by serving 1034 requests per second for eight clients, but its performance under Linux was close to it (1010 requests/s). It produced its poorest performance results under OpenBSD (806 requests/s) and also its stability was not convincing.

Table 7 DNS64 Performance: BIND, Forwarder, Atom

1	Operating System		Linux				OpenBSD				FreeBSD			
2	Number of clients		1	2	4	8	1	2	4	8	1	2	4	8
3	Exec. time of	average	0.93	0.96	1.05	1.76	0.98	1.01	1.37	2.80	0.95	0.99	1.13	2.08
4	256 `host`	std. dev.	0.01	0.02	0.02	0.04	0.02	0.02	0.06	0.10	0.01	0.02	0.02	0.02
5	commands (s)	maximum	0.98	1.03	1.14	1.93	1.03	1.10	1.47	2.95	1.00	1.06	1.19	2.14
6	CPU utiliza-	average	22.73	43.86	77.63	91.40	17.46	33.83	51.18	50.75	26.34	51.55	88.16	95.87
7	tion (%)	std. dev.	1.33	1.50	1.60	3.74	1.75	2.13	2.36	4.36	3.17	3.99	2.33	1.23
8	Memory cons. (MB)		48.582	83.949	148.121	187.984	39.555	72.707	112.379	107.418	58.258	92.547	164.145	175.242
9	Number of requests served in a second (req./s)		274	533	974	1164	262	506	750	730	268	517	903	987

Table 8 DNS64 Performance: TOTD, Forwarder, Atom

1	Operating System		Linux				OpenBSD				FreeBSD			
2	Number of clients		1	2	4	8	1	2	4	8	1	2	4	8
3	Exec. time of	average	0.84	0.84	0.96	1.81	0.84	0.84	0.89	1.52	0.84	0.84	0.88	1.37
4	256 `host`	std. dev.	0.01	0.01	0.02	0.09	0.01	0.02	0.02	0.29	0.01	0.01	0.02	0.08
5	commands (s)	maximum	0.89	0.88	1.04	1.92	0.88	0.89	0.96	5.58	0.89	0.89	0.95	1.53
6	CPU utiliza-	average	4.36	10.98	37.15	51.59	6.22	13.40	30.58	42.08	5.17	12.25	27.59	40.85
7	tion (%)	std. dev.	1.04	1.29	1.81	0.29	1.69	2.21	2.66	4.17	1.29	1.86	2.36	2.26
8	Memory cons. (MB)		1.188	1.082	1.270	1.875	2.207	2.328	2.484	3.734	3.773	3.453	2.984	3.883
9	Number of requests served in a second (req./s)		306	606	1062	1128	305	606	1154	1348	306	609	1165	1492

Table 9..DNS64 Performance: PowerDNS, Linux, Forwarder, Atom

1	Number of clients		1	2	4	8
2	Exec. time of	average	0.93	0.95	1.12	1.53
3	256 `host`	std. dev.	0.01	0.02	0.03	0.04
4	commands (s)	maximum	0.97	1.01	1.23	1.63
5	CPU utiliza-	average	18.43	36.88	62.92	99.95
6	tion (%)	std. dev.	1.19	1.21	1.71	0.15
7	Memory cons. (MB)		32.938	56.430	88.277	150.133
8	Number of requests served in a second (req./s)		276	537	914	1339

7.3 Performance Results of PowerDNS, Forwarder, Linux

The performance results of the DNS64 server realized by PowerDNS used as a forwarder were summarized in Table 3. Considering both the standard deviation and the maximum value of the execution time of one experiment, PowerDNS proved to be stable and its memory consumption is also bounded (less than 60MB), thus it can be used in production systems, though its performance is moderate (308 requests/s at 8 clients).

7.4 Comparison of the Forwarders

The best candidate for a DNS64 implementation used as a forwarder (on the Pentium III platform) is TOTD under FreeBSD (1034 requests/s) and TOTD under Linux is quite close to it (1010 request/s). The performance of BIND (422 request/s under OpenBSD, 386 request/s under Linux, 307 request/s under FreeBSD) and that of PowerDNS (308 request/s) are far from it, though they are also stable and may be used if they are preferred for some reasons. But if performance is important, TOTD is definitely the only choice. FreeBSD or Linux is a matter of taste – unless FreeBSD must be chosen for security reasons. (But we did not test the performance of TOTD when running in jail.)

7.5 Performance Results of BIND, Recursor

The performance results of the DNS64 server realized by BIND used as a recursor were summarized in Table 4.

7.5.1 Linux

Similarly to the case when it was a forwarder, BIND shows stability in every measured values. Its recursor performance (343 requests/s) is only 11% less than its forwarder performance (386 requests/s) at eight clients.

7.5.2 OpenBSD

Even though its performance shows somewhat degradation as the number of requests served in a second was 403 at four clients and it was only 394 at eight clients, this is only a 2.2% decrease and the maximum execution time of one experiment (5.3s) is very close to the average (5.19s) having also a very small standard deviation (0.07s). Therefore we consider it stable. Its recursor performance (394 requests/s) is only 6.6% less than its forwarder performance (422 requests/s) at eight clients.

7.5.3 FreeBSD

Similarly to OpenBSD, a little (but smaller) decrease of the performance can observed at eight clients, but it does not garble the stability of BIND. The increase in the memory consumption at eight clients (65.8MB) is also acceptable. Its recursor performance (295 requests/s) is only 3.9% less than its forwarder performance (307 requests/s) at eight clients.

7.5.4 BIND as a resursor under different operating systems

The same can be said about BIND as a recursor as we stated about it as a forwarder.

Table 10 DNS64 Performance: BIND, Recursor, Atom

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	0.95	0.98	1.10	1.94	1.00	1.03	1.43	2.92	0.96	1.00	1.16	2.14
4	**256 `host`**	std. dev.	0.01	0.02	0.02	0.05	0.02	0.03	0.07	0.09	0.01	0.02	0.02	0.03
5	**commands (s)**	maximum	0.99	1.09	1.18	2.12	1.05	1.13	1.57	3.05	1.02	1.07	1.23	2.21
6	**CPU utiliza-**	average	24.30	47.27	81.94	92.09	18.23	35.21	51.88	50.45	27.14	53.01	89.52	95.95
7	**tion (%)**	std. dev.	1.20	1.54	1.96	3.70	1.74	1.94	2.38	4.54	3.24	4.75	2.29	1.27
8	**Memory cons. (MB)**		49.211	84.668	151.891	175.527	41.500	72.781	108.363	103.207	58.078	91.816	159.398	175.715
9	**Number of requests ser-ved in a second (req./s)**		269	521	929	1054	257	495	714	701	267	512	882	959

Table 11 DNS64 Performance: Unbound, Recursor, Atom

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	0.85	0.86	0.88	0.96	0.86	0.87	0.90	1.20	0.86	0.87	0.90	1.08
4	**256 `host`**	std. dev.	0.01	0.01	0.02	0.02	0.01	0.01	0.02	0.07	0.01	0.01	0.02	0.02
5	**commands (s)**	maximum	0.91	0.90	0.93	1.01	0.92	0.92	0.95	1.33	0.90	0.93	0.95	1.14
6	**CPU utiliza-**	average	8.01	15.94	30.43	50.21	9.91	19.20	36.78	53.92	8.77	18.23	34.83	54.23
7	**tion (%)**	std. dev.	1.11	1.18	1.15	0.66	1.88	2.31	2.37	4.10	1.58	1.84	2.23	1.56
8	**Memory cons. (MB)**		16.082	16.016	15.961	15.992	25.570	23.266	22.594	25.750	20.270	18.875	18.711	18.711
9	**Number of requests ser-ved in a second (req./s)**		299	594	1161	2122	297	587	1139	1700	297	586	1135	1894

Table 12 DNS64 Performance: PowerDNS, Linux, Recursor, Atom

1	**Number of clients**		**1**	**2**	**4**	**8**
2	**Exec. time of**	average	0.94	0.96	1.15	1.60
3	**256 `host`**	std. dev.	0.01	0.02	0.03	0.04
4	**commands (s)**	maximum	0.97	1.04	1.28	1.69
5	**CPU utiliza-**	average	19.06	37.81	64.63	99.96
6	**tion (%)**	std. dev.	1.17	1.30	1.92	0.14
7	**Memory cons. (MB)**		33.074	56.422	89.012	150.254
8	**Number of requests ser-ved in a second (req./s)**		273	531	887	1279

7.6 Performance Results of Unbound, Recursor

The performance results of the DNS64 server realized by Unbound used as a recursor were summarized in Table 5. As there are no stability issues with Unbound, we discuss its results for the different operating systems together. Unbound is stable under all three operating systems, and it requires relatively small amount of memory under all of them. It gave the best performance under Linux by serving 1211 requests per second at eight clients. Its performance is somewhat lower under OpenBSD (1032 requests/s) and FreeBSD (959 requests/s), but they are still excellent. Its memory consumption is the least under Linux (about 15MB) and is a bit higher under the BSD systems (about 20-23 MB).

7.7 Performance Results of PowerDNS, Recursor, Linux

The performance results of the DNS64 server realized by PowerDNS used as a recursor were summarized in Table 6. There is no stability issue and the performance of PowerDNS as a recursor (297 requests/s) was only 3.6% less than its forwarder performance (308 requests/s) at eight clients.

7.8 Comparison of the Recursors

The best candidate for a DNS64 implementation used as a recursor (on the Pentium III platform) is Unbound under Linux (1211 requests/s) and Unbound showed good performance under the BSD systems, too (1032 requests/s under OpenBSD and 959 requests/s under FreeBSD). The performance of BIND and PowerDNS are far from it (they can serve only about 300-400 requests per second).

8 Results of the Measurements on Atom CPU

We present all the measurement results of the Intel Atom test computer but in the discussion, we focus on the differences between the two platforms only, as there are no stability issues found (except TOTD under OpenBSD). The memory consumption of the Atom system is usually higher than that of the Pentium III system (as this computer contains 4GB RAM instead 128MB), and it is often growing with the number of the clients, but when the number of the clients is doubled, the growth of the memory consumption is always far less from being doubled thus it is not an issue.

Before the presentation of the results, we need to clear the meaning of the CPU utilization values in the tables because there are two equally good but different practices exist. One of them takes the performance of a single core to 100% and thus the performance of two cores is denoted as 200%. The other one takes the performance of all the cores to 100% and thus the performance of one core of a dual core system is denoted as 50%. Our measurement programs used the second one and thus we do so in the following tables.

8.1 Performance Results of BIND, Forwarder

The performance results of the DNS64 server realized by BIND used as a forwarder were summarized in Table 7.

8.1.1 Linux

The CPU utilization values show that BIND under Linux benefits from the dual core architecture. However, it cannot fully utilize the computing power of both cores: the CPU utilization is 77.63% and the average execution time of one

Table 13 DNS64 Performance: BIND, Forwarder, Opteron

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	0.067	0.098	0.213	0.409	0.094	0.188	0.382	0.783	0.082	0.121	0.239	0.479
4	**256 DNS**	std. dev.	0.005	0.007	0.017	0.018	0.006	0.005	0.007	0.015	0.010	0.005	0.008	0.012
5	**queries (s)**	maximum	0.140	0.170	0.290	0.530	0.180	0.260	0.440	0.860	0.490	0.160	0.290	0.540
6	**CPU utiliza-**	average	57.91	72.20	63.00	69.92	26.15	27.22	27.42	27.28	66.88	87.82	88.68	89.52
7	**tion (%)**	std. dev.	0.95	1.38	3.02	2.45	0.85	0.87	0.98	0.93	2.10	1.76	1.76	1.74
8	**Memory cons. (MB)**		49.992	80.691	147.262	277.242	38.324	67.652	123.117	233.848	62.609	94.406	169.414	303.613
9	**Number of requests ser-ved in a second (req./s)**		3838	5208	4816	5003	2721	2724	2682	2615	3130	4215	4290	4272

Table 14 DNS64 Performance: TOTD, Forwarder, Opteron

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	0.081	0.169	0.357	0.775	0.063	0.129	0.271	0.572	0.058	0.117	0.240	0.509
4	**256 DNS**	std. dev.	0.029	0.052	0.095	0.102	0.012	0.021	0.035	0.052	0.009	0.021	0.067	0.058
5	**queries (s)**	maximum	0.190	0.310	1.070	1.230	0.360	0.200	0.370	0.690	0.080	0.160	1.040	0.740
6	**CPU utiliza-**	average	23.95	25.15	25.14	25.24	25.33	27.58	27.38	27.25	23.95	26.42	26.69	26.71
7	**tion (%)**	std. dev.	0.33	0.41	0.33	0.37	0.79	0.78	1.01	0.92	0.98	0.69	0.54	0.61
8	**Memory cons. (MB)**		1.582	1.734	2.035	1.461	2.609	2.598	2.633	2.246	5.793	5.902	3.980	4.996
9	**Number of requests ser-ved in a second (req./s)**		3180	3025	2871	2642	4035	3978	3779	3582	4428	4366	4275	4025

Table 15 DNS64 Performance: PowerDNS, Linux, Forwarder, Opteron

1	**Number of clients**		**1**	**2**	**4**	**8**
2	**Exec. time of**	average	0.070	0.099	0.179	0.366
3	**256 DNS**	std. dev.	0.003	0.012	0.011	0.019
4	**queries (s)**	maximum	0.130	0.190	0.240	0.440
5	**CPU utiliza-**	average	42.47	60.20	96.90	97.59
6	**tion (%)**	std. dev.	0.59	2.69	0.53	0.37
7	**Memory cons. (MB)**		35.277	64.613	96.195	160.199
8	**Number of requests ser-ved in a second (req./s)**		3670	5146	5723	5600

experiment is 1.05 at four clients and when the number of clients is doubled the CPU utilization grows only to 91.4% and the execution time grows to 1.76s. BIND cannot fully parallelize all its tasks to do. (There could be other reasons, e.g. the performance could be limited by the I/O capacities, but later we can see that PowerDNS can do more requests in a second thus it is sure that the limitation is not caused by the lack of other resources, but rather by the behavior of BIND.)

8.1.2 OpenBSD

The CPU utilization values show that BIND under OpenBSD cannot much benefit from the dual core architecture. Though the CPU utilization is somewhat higher than 50% at four and eight clients, it is caused by other processes that are scheduled for the other core. The single-threaded nature of OpenBSD is liable for this situation[2].

8.1.3 FreeBSD

The CPU utilization values show that BIND under Linux benefits from the dual core architecture, but it cannot fully utilize the computing power of both cores.

[2] Though the OpenBSD kernel supports multithreading since version 5.2, the OpenBSD port of BIND still does not support it.

8.1.4 BIND as a forwarder under different operating systems

The Linux system produced the best results by answering 1164 requests per second and FreeBSD was the second one with 987 requests/s. Because of the single-threaded nature of OpenBSD it produced the poorest results (730 requests/s). If one uses more than two cores then the disadvantage of OpenBSD is growing further.

8.2 Performance Results of TOTD, Forwarder

The performance results of the DNS64 server realized by TOTD used as a forwarder were summarized in Table 8. Unfortunately, TOTD cannot benefit from the dual core architecture under any of the three operating systems because it was written single threaded.

This fact could be a comparative advantage for the OpenBSD platform, but we have to issue a warning about the stability of TOTD under OpenBSD, see the large maximum (5.58s) and standard deviation (0.29s) values of the execution time compared to the average (1.52s) at eight clients. Therefore we recommend only the FreeBSD operating system with 1492 requests per second, even though the performance of OpenBSD is not much lower (1348 requests/s). If Linux is preferred for some reason, it is also stable and its performance is also good (1128 requests/s).

8.3 Performance Results of PowerDNS, Forwarder

The performance results of the DNS64 server realized by PowerDNS used as a forwarder were summarized in Table 9. The 99.95% CPU utilization value at eight clients shows that PowerDNS can fully utilize the computing power of the two cores. It can answer 1339 requests per second.

8.4 Comparison of the Forwarders

Considering only the bare numbers, the best candidate for a

Table 16 DNS64 Performance: BIND, Recursor, Opteron

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	0.071	0.103	0.219	0.427	0.100	0.201	0.411	0.839	0.086	0.123	0.243	0.491
4	**256 DNS**	std. dev.	0.015	0.006	0.014	0.015	0.003	0.003	0.008	0.015	0.006	0.005	0.007	0.014
5	**queries (s)**	maximum	0.600	0.190	0.290	0.530	0.180	0.250	0.470	0.930	0.140	0.160	0.290	0.560
6	**CPU utiliza-**	average	58.79	75.46	70.10	76.03	26.24	27.30	27.20	27.15	67.26	88.74	89.23	90.10
7	**tion (%)**	std. dev.	2.09	1.49	2.44	2.40	0.82	0.85	0.80	0.97	2.40	1.76	1.85	1.88
8	**Memory cons. (MB)**		51.227	81.387	150.078	282.293	38.344	67.543	122.230	234.668	62.836	94.461	167.320	302.922
9	**Number of requests served in a second (req./s)**		3623	4961	4682	4796	2555	2553	2491	2440	2994	4151	4208	4170

Table 17 DNS64 Performance: Unbound, Recursor, Opteron

1	**Operating System**		**Linux**				**OpenBSD**				**FreeBSD**			
2	**Number of clients**		**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
3	**Exec. time of**	average	0.041	0.070	0.137	0.271	0.054	0.083	0.168	0.347	0.050	0.070	0.135	0.281
4	**256 DNS**	std. dev.	0.004	0.002	0.005	0.027	0.006	0.005	0.007	0.019	0.003	0.003	0.005	0.016
5	**queries (s)**	maximum	0.100	0.140	0.160	0.320	0.170	0.160	0.250	0.400	0.140	0.110	0.150	0.330
6	**CPU utiliza-**	average	21.09	24.92	25.15	25.50	21.46	28.32	28.20	28.40	20.59	27.08	27.61	27.43
7	**tion (%)**	std. dev.	0.65	0.40	0.35	0.75	1.63	0.85	1.05	0.97	1.28	0.78	0.65	0.62
8	**Memory cons. (MB)**		16.355	15.629	15.969	15.523	22.484	23.727	23.652	22.707	21.871	20.074	21.883	20.152
9	**Number of requests served in a second (req./s)**		6259	7314	7493	7545	4725	6195	6087	5900	5101	7349	7610	7291

Table 18 DNS64 Performance: PowerDNS, Linux, Recursor, Opteron

1	**Number of clients**		**1**	**2**	**4**	**8**
2	**Exec. time of**	average	0.071	0.101	0.184	0.378
3	**256 DNS**	std. dev.	0.011	0.015	0.012	0.019
4	**queries (s)**	maximum	0.360	0.310	0.360	0.460
5	**CPU utiliza-**	average	42.86	61.09	96.89	97.70
6	**tion (%)**	std. dev.	0.57	2.62	0.54	0.39
7	**Memory cons. (MB)**		35.438	64.016	96.453	159.035
8	**Number of requests served in a second (req./s)**		3616	5065	5560	5425

DNS64 implementation on the dual core Atom platform, used as a forwarder is the TOTD under FreeBSD (1492 requests/s). As we disqualified TOTD under OpenBSD (1348 requests/s) for its stability warning, the second and third ones are PowerDNS under Linux (1339 requests/s) and BIND under Linux (1164 requests/s). However, as PowerDNS and BIND are multi-threaded they will very likely overtake the single-threaded TOTD when more than two cores will be used.

8.5 Performance Results of BIND, Recursor

The performance results of the DNS64 server realized by BIND used as a recursor were summarized in Table 10. The results are very similar to those when BIND was used as a forwarder: Linux is the best one (1054 request/s), FreeBSD is the second one (959 requests/s) and OpenBSD produced the poorest performance (701 requests/s) due to being single threaded.

8.6 Performance Results of Unbound, Recursor

The performance results of the DNS64 server realized by Unbound used as a recursor were summarized in Table 11. Unbound is single-threaded and yet very fast! It gave the best performance under Linux by serving 2122 requests per second at eight clients. Its performance is lower but still very good under FreeBSD (1894 requests/s) and OpenBSD (1700 requests/s). It is interesting that Unbound under the OpenBSD operating system (1032 requests/s) produced somewhat better results than under FreeBSD (959 requests/s) when it was executed by the Pentium III computer and the order is opposite when it is executed by the Atom computer. As Unbound is single threaded, this phenomenon is an important warning sign that the architecture itself (and not only the number of cores) may have a significant influence on the performance of the different DNS64 implementations.

The memory consumption of Unbound is really low. It is the least under Linux (about 16MB) and it is a bit higher under FreeBSD (about 18-20 MB) and under OpenBSD (23-26MB).

8.7 Performance Results of PowerDNS, Recursor, Linux

The performance results of the DNS64 server realized by PowerDNS used as a recursor were summarized in Table 12. PowerDNS can fully utilize the computing power of the two cores. It can answer 1279 requests in a second.

8.8 Comparison of the Recursors

Even though Unbound is single threaded its performance is far the best. It gave the best performance under Linux by serving 2122 requests per second at eight clients. Its performance is lower but still very good under FreeBSD (1894 requests/s) and OpenBSD (1700 requests/s). PowerDNS can answer only 1279 requests in a second by fully utilizing both cores. And the performance of BIND is somewhat lower.

However, PowerDNS and BIND have the potential to outperform Unbound when executed by systems with higher number of CPU cores.

9 Results of the Measurements with Opteron CPUs

We present all the measurement results of the Sun test computer with two dual core Opteron CPUs, but we focus on how the presence of the four cores change the ranking of the DNS64 implementations under the different operating systems.

There are two minor changes in the tables from this point:

- Lines 2-4 show the execution time of 256 *DNS queries* (and not `host` *commands*) because the C program was used with the third test computer.
- There are 3 digits used after the decimal point in the time values because the values are less than a second due to the high computing power of third test computer.

9.1 Performance Results of BIND, Forwarder

The performance results of the DNS64 server realized by BIND used as a forwarder were summarized in Table 13.

The CPU utilization values show that BIND under Linux benefits from the four cores. However, it is far from fully utilizing the computing power of all the four cores: the CPU utilization is always below 75% (which is equivalent with the computing power of 3 cores). BIND under OpenBSD can benefit a little from the multiple cores e.g. it produced 27.42% CPU utilization at four clients instead of the 25% (which is equivalent with a single core) but the gain is insignificant. BIND produced its highest CPU utilization value under FreeBSD; it is 89.52% at eight clients, which is much better than under Linux but it is still not 100%. However, as for the number of requests served in a second, Linux performed the best by processing about 5000 requests in a second.

9.2 Performance Results of TOTD, Forwarder

The performance results of the DNS64 server realized by TOTD used as a forwarder were summarized in Table 14. Whereas the CPU utilization was always less than 25.3% under Linux, the system could benefit 2-3% of the multiple cores under the BSD operating systems. TOTD showed it best performance under FreeBSD by processing more than 4000 requests in a second. (Its performance showed continuous but small degradation under all the three operating systems from one client to eight clients.)

9.3 Performance Results of PowerDNS, Forwarder

The performance results of the DNS64 server realized by PowerDNS used as a forwarder were summarized in Table 15. The CPU utilization values show that PowerDNS under Linux can nearly fully utilize the four cores (97.59% at eight clients). PowerDNS processed 5600 requests in a second at eight clients.

9.4 Comparison of the Forwarders

PowerDNS was the best performing one among the forwarders due to being able to nearly fully utilize the computing power of all the four cores. Note that this result was achieved under serious overload and under normal high load (at one or two clients) BIND showed similar results than PowerDNS. TOTD was the third one, but if we consider the FreeBSD operating system, its performance is practically the same as that of BIND at 2, 4 and 8 clients (about 4000 requests in a second) and at one client, TOTD even seriously outperformed BIND by answering 4428 requests per second whereas BIND could do only 3130.

Our expectations were fulfilled that BIND and PowerDNS outperformed TOTD under both multi-threaded operating systems (Linux and FreeBSD).

Table 19 DNS64 Implementation Selection Guide: The best Performing DNS64 Implementations with the Number of Forwarded Packets per Second

Opera-tion Mode	Operating System	Intel Pentium III (single-core)	Intel Atom (dual-core)	AMD Opteron (quad-core)
forwarder	Linux	**TOTD** 1010	**PowerDNS** 1339	**PowerDNS** 5600
	BSD	**TOTD** 1034/Free	**TOTD** 1492/Free	**BIND** 4272/Free
recursor	Linux	**Unbound** 1211	**Unbound** 2122	**Unbound** 7545
	BSD	**Unbound** 1032/Open, 959/Free	**Unbound** 1894/Free	**Unbound** 7291/Free

9.5 Performance Results of BIND, Recursor

The performance results of the DNS64 server realized by BIND used as a resursor were summarized in Table 16. The trend of the values is similar to the one when BIND was used as a forwarder. The performance results are somewhat less, as expected.

9.6 Performance Results of Unbound, Recursor

The performance results of the DNS64 server realized by Unbound used as a resursor were summarized in Table 17. Whereas the CPU utilization never exceeds 25.5% under Linux, the system can benefit 3-4% of the multiple cores under the BSD systems. Unbound showed its best performance under Linux by continuously increasing the number of processed requests and reaching 7545 requests in a second at eight clients, but its performance under FreeBSD was very close to it. Even though FreeBSD has somewhat overtaken Linux at four clients (7610 vs. 7493 requests per second), we put FreeBSD to the second place because of the other results and also because of the performance degradation it showed at eight clients. As for the performance order of the two BSDs, the earlier observed tendency continued: the performance of the OpenBSD system (about 6000 requests per seconds from two to eight clients) is now visibly lower than that of FreeBSD.

9.7 Performance Results of PowerDNS, Recursor

The performance results of the DNS64 server realized by PowerDNS used as a resursor and executed by the Sun test computer were summarized in Table 18. The CPU utilization values show that PowerDNS under Linux can nearly fully utilize the four cores (97.7% at eight clients). PowerDNS processed 5425 requests per second at eight clients.

9.8 Comparison of the Recursors

Unbound performed the best amoung the recursors with significant vantage. At eight clients, Unbound served 7545 request in a second whereas PowerDNS could do only 5425 and BIND did 4796. On the basis of the two core results, PowerDNS was expected to catch up with Unbound, but it did not happen: Unbound significantly outperformed PowerDNS despite of the fact that Unbound is single threaded and PowerDNS could nearly fully utilize all the four cores.

10 Implementation Selection Guide

We provide an easy to use guide for those who would like to choose a DNS64 implementation suitable for their purposes quickly. The performance optimized choices are shown in Table 19. We give the best choices for both operation modes (forwarder and recursor), for all the tested CPUs and also for Linux and BSD systems. We also specify the number of forwarded packets per second for 8 clients, thus they can be a key for the decision if one has no special preference between Linux and BSD. Within the BSD platform, we denote OpenBSD and FreeBSD by supplementing the performance value of the DNS64 implementations with words "Free" and "Open", respectively. (Note the performance of certain implementations was higher for lower number of clients, but we always give the performance values measured with 8 clients.)

11 Plans for Future Research

Our current results can serve as a starting point when selecting the directions of future investigations. There should be several modern CPUs tested including servers with 8 or 16 cores. It is a very interesting question how the multi-threaded DNS64 implementations scale on different CPU architectures having several cores. Some of these would require more powerful and/or higher number of client computers for load generation than those we used. For a well tunable and cost effective solution, we consider building a 64 or 128 element cluster of single board computers (SBCs) similar to the Raspberry Pi cluster described in [38] but using more powerful single board computers. (Some candidates are under testing, see some of our SBC comparison results in [39].)

We have also rewritten the complete test program in C/C++ for achieving higher efficiency in load generation [40].

MTD64, the tiny Multi-Threaded DNS64 server we proposed in [10] was not ready for testing at the time when our measurements were performed, but we also plan to include it into our furthers investigations.

The impact of caching is another very interesting topic, and we plan to deal with it in a later paper.

12 Conclusion

We have found that the CPU architecture has a strong influence on the performance ranking of the analyzed DNS64 implementations. Now, we give a brief summary of the results, stating also the number of processed requests per second with eight clients for each mentioned implementations.

On the single-core Pentium III platform, TOTD under FreeBSD (1034 requests/s) or under Linux (1010 requests/s) was found highly the best among the forwarders – it outperformed about 2.5 times the second one, BIND under OpenBSD (422 requests/s) whereas PowerDNS could process only 308 requests/s. As for the recursors on Pentium III, Unbound under Linux (1211 requests/s) was the very best one, and it was also very good under OpenBSD (1032 request/s) and FreeBSD (959 requests/s), too. The performance of BIND under its best operating system, OpenBSD (394 requests/s) and the results of PowerDNS (297 requests/s) lag far behind Unbound.

The increasing of the number of cores could partially reverse the performance ranking. Using four cores, the best performing forwarder was PowerDNS (5600 request/s), the second one was BIND under Linux (5003 requests/s). The performance of TOTD under FreeBSD (4025 request/s) was still notable because it was achieved using only a single core. As for the recursors, the single threaded Unbound kept its first place with a high vantage, its best performing operating system was Linux (7545 requests/s), but FreeBSD (7291 requests/s) was close to it. PowerDNS (5425 requests/s) could nearly fully utilize the computing power of the four cores whereas BIND under Linux (4796 requests/s) could utilize the computing power of about three of them.

The "race" is still open for eight or more cores and/or with different CPU types.

Nearly all of the implementations were found stable. Only TOTD under OpenBSD has got stability warning on the Pentium III and Atom platforms.

We hope that our results can serve as useful guidelines for network administrators and architects when selecting the best suitable DNS64 implementations for their networks. We believe that our work may contribute to the global deployment of the IPv6 protocol.

References

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

[2] Bagnulo, M., Matthews, P., & Beijnum, I. (2011). Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers, IETF RFC 6146.

[3] Lencse, G., & Répás, S. (2013). Performance analysis and comparison of the TAYGA and of the PF NAT64 implementations, in *Proceedings of the 36th International Conference on Telecommunications and Signal Processing*, Rome, Italy, doi: 10.1109/TSP.2013.6613894

[4] Répás, S., Farnadi, P. & Lencse, G. (2014). Performance and stability analysis of free NAT64 implementations with different protocols, *Acta Technica Jaurinensis,* 7(4), doi: 10.14513/actatechjaur.v7.n4.340

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

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

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

[8] Wu, P., Cui, Y., Wu, J., Liu, J. & Metz C. (2013). Transition from IPv4 to IPv6: A state-of-the-art survey" *IEEE Communication Surveys & Tutorials*, 15 (3), doi: 10.1109/SURV.2012.110112.00200

[9] Skoberne, N., Maennel, O., Phillips, I., Bush, R., Zorz, J., Ciglaric, M. (2014). IPv4 address sharing mechanism classification and tradeoff analysis, *IEEE/ACM Transactions on Networking*, 22 (2) doi: 10.1109/TNET.2013.2256147

[10] Lencse, G. & Soós, A. G. (2015). Design of a tiny multi-threaded DNS64 server, in *Proceedings of the 38th International Conference on Telecommunications and Signal Processing*, Prague, Czech Republic, pp. 27–32. 10.1109/TSP.2015.7296218

[11] Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., & Li, X. (2010). IPv6 addressing of IPv4/IPv6 translators, IETF RFC 6052.

[12] Bagnulo, M., Garcia-Martinez, A., & Beijnum, I. V. (2012). The NAT64/DNS64 tool suite for IPv6 transition, *IEEE Communication Magazine*, 50(7), 177–183.

[13] Llanto, K. J. O., & Yu, W. E. S. (2012). Performance of NAT64 versus NAT44 in the context of IPv6 migration, in *Proceedings of the International MultiConference of Engineers 2012*, vol. I., pp. 638–645.

[14] Monte, C. P., Robles, M. I., Mercado, G., Taffernaberry, C., Orbiscay, M., Tobar, S., Moralejo, R., Pérez, S. (2012). Implementation and evaluation of protocols translating methods for IPv4 to IPv6 transition, *Journal of Computer Science & Technology*, 12(2) 64–70.

[15] Yu, S., & Carpenter, B. E. (2012). Measuring IPv4 – IPv6 translation techniques, Dept. of Computer Science, Univ. Auckland, Auckland, New Zeeland, Technical Report 2012-001, Jan., 2012. https://www.cs.auckland.ac.nz/~brian/IPv4-IPv6coexistenceTechnique-TR.pdf Accessed 19 August 2015.

[16] Lencse G., & Takács, G. (2012). Performance analysis of DNS64 and NAT64 solutions, *Infocommunications Journal*, 4(2) pp. 29–36.

[17] Hodzic, E., & Mrdovic, S. (2012). IPv4/IPv6 transition using DNS64/NAT64: Deployment issues, in *Proceedings of the 2012 IX International Symposium on Telecommunications*, Sarajevo, Bosnia and Herzegovina, doi: 10.1109/BIHTEL.2012.6412066

[18] Lencse G., & Répás, S. (2013). Performance analysis and comparison of different DNS64 implementations for Linux, OpenBSD and FreeBSD, in *Proceedings of the IEEE 27th International Conference on Advanced Information Networking and Applications*, Barcelona, Spain, doi: 10.1109/AINA.2013.80

[19] Lencse, G., & Répás, S. (2014). Improving the performance and security of the TOTD DNS64 implementation, *Journal of Computer Science & Technology*, 14(1), pp. 9–15.

[20] Dillema. F. W. (2014). TOTD 1.5.3 source code, https://github.com/fwdillema/totd Accessed 19 August 2015.

[21] Free Software Foundation, The free software definition, http://www.gnu.org/philosophy/free-sw.en.html Accessed 19 August 2015.

[22] Open Source Initiative, The open source definition, http://opensource.org/docs/osd Accessed 19 August 2015.

[23] Cisco, End user license agreement, http://www.cisco.com/en/US/docs/general/warranty/English/EU1KEN_.html Accessed 19 August 2015.

[24] Juniper Networks, End user license agreement, http://www.juniper.net/support/eula.html Accessed 19 August 2015.

[25] Lencse, G., & Répás, S. (2013). Performance analysis and comparison of 6to4 relay implementations, *International Journal of Advanced Computer Science and Applications*, 4(9) doi: 10.14569/IJACSA.2013.040903

[26] Internet Systems Consortium, *Berkeley Internet Name Daemon (BIND)*, https://www.isc.org/software/bind Accessed 19 August 2015.

[27] Dunmore, M. (Ed.) (2015). *An IPv6 Deployment Guide*, The 6NET Consortium, https://www.6net.org/book/deployment-guide.pdf Accessed 19 August 2015.

[28] NLnet Labs, *Unbound*, http://unbound.net Accessed 19 August 2015.

[29] Marsan, C. D. (2008). New open source DNS server released, http://www.infoworld.com/t/applications/new-open-source-dns-server-released-599 Accessed 19 August 2015.

[30] Nemeth, E., Snyder, G., Hein, T. R., & Whaley, B. (2011). *Unix and Linux System Administration Handbook*, 4th ed., Michigan, Pearson Education, Inc., http://books.google.hu/books?isbn=0132117363 Accessed 19 August 2015.

[31] Perreault, S., Dionne, J.-P., & Blanchet, M. (2010). Ecdysis: Open-source DNS64 and NAT64", *AsiaBSDCon*, Tokyo, Japan, https://2010.asiabsdcon.org/papers/abc2010-P4B-paper.pdf Accessed 19 August 2015.

[32] Powerdns.com BV, "PowerDNS", http://www.powerdns.com Accessed 19 August 2015.

[33] Klein, A. (2008). PowerDNS recursor DNS cache poisoning, Trusteer, http://www.trusteer.com/docs/powerdns_recursor_dns_cache_poisoning.pdf Accessed 19 August 2015.

[34] Kanclirz, J., Jr. (Ed.), (2008). *Netcat Power Tools*, Syngress Publishing, http://dl.acm.org/citation.cfm?id=2155689 Accessed 19 August 2015.

[35] Mills, D., Martin, J., Burbank, J., Kasch, W. (2010). Network time protocol version 4: Protocol and algorithms specification, IETF RFC 5905.

[36] NTIA ITS, "Definition of 'graceful degradation' " http://www.its.bldrdoc.gov/fs-1037/dir-017/_2479.htm Accessed 19 August 2015.

[37] The FreeBSD Documentation Project (1995-2015), *FreeBSD Handbook* (Chapter 16. Jails), http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html Accessed 19 August 2015.

[38] Cox, S. J., Cox, J. T., Boardman, R. P., Johnston, S. J., Scott, M., & O'Brien, N. S. (2014). Iridis-pi: a low-cost, compact demonstration cluster, *Cluster Computing*, 17 (2) doi: 10.1007/s10586-013-0282-7

[39] Lencse, G. & Répás, S. (2015). Method for benchmarking single board computers for building a mini supercomputer for simulation", in *Proceedings of the 38th International Conference on Telecommunications and Signal Processing*, Prague, Czech Republic, pp. 246–251. DOI: 10.1109/TSP.2015.7296261

[40] Lencse, G. (2015). Test program for the performance analysis of DNS64 servers", *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 4. no. 3. pp 60–65. DOI: 10.11601/ijates.v4i3.121

Gábor Lencse received MSc and PhD in computer science from the Budapest University of Technology and Economics, Budapest Hungary in 1994 and 2001, respectively.

He works for the Department of Telecommunications, Széchenyi István University, Győr, Hungary since 1997. Now, he is an Associate Professor. He is also a part time Senior Research Fellow at the Department of Networked Systems and Services, Budapest University of Technology and Economics since 2005. His research interests include the performance analysis of communication systems, parallel discrete event simulation methodology and IPv6 transition methods.

Sándor Répás received his BA in Business Administration and Management from the Corvinus University of Budapest, Budapest Hungary in 2009, BSc in Electrical Engineering from the Óbuda University in 2011 and MSc in Electrical Engineering from the Széchenyi István University in 2013.

He is a full time PhD student in information technology at the Széchenyi István University, Győr Hungary. The main field of his research is the IPv6 implementation technologies. His other favorite topics are computer networking and information security. He has several certificates from Microsoft, Cisco, ISACA and other vendors.