Stability Analysis and Performance Comparison of Three 6to4 Relay Implementations

Sándor Répás, Viktor Horváth, and Gábor Lencse

Abstract— During the IPv6 deployment there is a frequently occurring situation where two IPv6 enabled hosts need to communicate with each other over a network that supports only IPv4. Application of the 6to4 IPv6 transition method can solve this problem. The performance and stability of the different 6to4 relay implementations is a very important subject. We measured the performance and tested the stability of three open source 6to4 relay implementations under Debian Linux, OpenBSD and OpenWRT platforms. We present and discuss our results, analyze the stability of the 6to4 relay implementations and compare their performance metrics. Our measurements methods may be useful for other researchers, and our results may help the system architects to choose the appropriate solution.

Keywords—6to4 relay, IPv6 transition, network communication, performance evaluation, stability analysis

I. INTRODUCTION

FOR more than two decades it is a known fact, that the size of the IPv4 address space is insufficient [1]-[2]. The lack of the IP addresses withholds the spread of the Internet and causes social and economic damage.

To prevent the IP address exhaustion, a new version of the Internet Protocol, the IPv6 has been developed. IPv6 was standardized in 1998 and published in the RFC 2460 [3], but it has not been widespread adopted. According to the statistics, less than 5.5% of the total amount of the traffic reached the Google servers used IPv6 protocol in December 2014 [4]. Three of the five Regional Internet Registries (RIR) already run out of their IPv4 address spaces [5]. The five RIRs have only 5.2 /8 ranges in total, whereas the IANA does not have more address space to assign to the five RIRs since 3 February 2011 [6]. The RIRs work according to strict policies and for a service provider it is a harder task than ever to get IPv4 address spaces. The speed up of the transition to the new protocol is inevitable. Several IPv6 transition techniques have been developed, which can help the process in different phases of the adoption of the new protocol on the Internet.

There are different situations to solve during the coexistence of the two versions of the IP protocol in the different phases of the transition process:

In theory, the best solution is the Dual Stack (DS) transition method [7], but with the requirements that the two communicating hosts and the network between them have to support a common version of the IP protocol, and because of the IPv4 exhaustion, there is not enough IPv4 addresses to use this solution. Thus, even though it could have been the best solution, now it is too late for using DS as an IPv6 transition method.

In a situation where an IPv6 only client computer needs to communicate with an IPv4 only server the DNS64 [8] and NAT64 [9] combination is a good solution. The performance, the stability and the application compatibility of some open source implementations of DNS64/NAT64 are examined and proved in [10]-[12].

If two IPv6 enabled hosts need to communicate with each other over an IPv4 network, they can use different tunneling methods. The 6in4 (also called manual tunnel) [13] with tunnel brokers [14]-[15], 6rd [16], Teredo [17] ISATAP [18] and 6to4 [19] have different requirements, benefits and drawbacks.

The above list is not exhaustive and a good survey of the different transition techniques can be found in [20].

The remainder of this paper is organized as follows: first, some properties of the 6to4 transition technique are introduced, second, a short survey of the results of the most current publications is given, third, the selected 6to4 relay implementations are introduced, fourth, our test environment is described, fifth, the performance measurement method of the different implementations is detailed, sixth, the results are presented and discussed, seventh, the comparison of our results is presented, finally, our conclusions are given.

II. THE 6TO4 TRANSITION TECHNIQUE

The 6to4 transition technique uses automatic tunnels, encapsulates the IPv6 packets into IPv4 packets [19] (using protocol number 41, as the configured IPv6 over IPv4 tunnel [21]). The main advantage of the automatic tunneling is the unnecessity of the manual configuration of the endpoint address of the tunnel. Automatic IPv6-over-IPv4 tunneling determines the IPv4 tunnel endpoint address from the IPv4 address embedded in the destination address of the IPv6 packet being tunneled. 6to4 protocol uses the reserved 2002::/16 6to4 prefix [22] to determine if a 6to4 tunnel creation is necessary. A 6to4 address is an IPv6 address constructed using a 6to4 prefix. The first 16 bits of the 6to4 address contain the 2002 hexadecimal value, whereas the next 32 bits contain the IPv4 address of the 6to4 tunnel endpoint. The next 16 bits can be used to create subnets, and the final 64 bits of the 6to4 address contain the interface ID.

A 6to4 router is an IPv6 router supporting a 6to4 pseudo-interface. It is normally the border router between an IPv6 site.
and a wide-area IPv4 network, whereas the 6to4 pseudo-interface is the point of the encapsulation of the IPv6 packets in the IPv4 packets (with other words: the tunnel end-point) [19]. If a 6to4 host have to communicate with a non 6to4 host (for example: native IPv6, Teredo) it needs to use a 6to4 relay router.

Several operating systems can work as a 6to4 router or 6to4 relay router, but for the correct operation, the 6to4 routers and relay routers need public IPv4 addresses.

A 6to4 relay router can be private or public. Public 6to4 relays use the 192.88.99.1 anycast address [23] from the 192.88.99.0/24 6to4 Relay anycast address range [24]. An estimation of the 6to4 relay routers published in 2006 [25]. According to the publication, 8 autonomous systems (AS-es) advertised the 192.88.99.0/24, whereas 6 AS-es advertised the 2002::/16 networks. At the end of the year 2014 these values were 14 and 11, according to the RIPEstat database [26].

It is a good practice, if the Internet Service Provider (ISP) provides a 6to4 relay for its customers in addition to other transition solutions. In this case the relay does not have to be public, and it can use the well-known anycast address, or a network specific address.

Though some security weaknesses are known of the 6to4 transition technique [27], it helps the implementation of the IPv6 protocol without the cooperation of the ISP.

More details of the operation of the 6to4 technique can be found in the publication [28], and in the related RFCs ([19], [24] and [27]).

III. TESTED IMPLEMENTATIONS

The following widely used open source [29] (also called free software [30]) operating systems and their 6to4 implementations were chosen for the tests: Debian Linux sit, FreeBSD sit interface, OpenWRT 6to4 plus kmod-sit packages. The open source software can be freely used by anyone, and their licenses allow the performance benchmarks. These two arguments were the most important ones in our selection of the implementations for testing.

The following software versions were used:

- Debian 7.1.0_x86 – sit (obsolete)
- OpenWRT (Attitude Adjustment) 12.09_x86 – sit
- FreeBSD 9.1_x86 – sit.

IV. TEST ENVIRONMENT

A. Topology of the Network

An isolated test network was built for the performance and the stability measurements. The topology of the network can be seen in Fig. 1. Due to the isolation, any IPv4 and IPv6 addresses could be used on the network. The computer on the top of the figure played the role of the “internet” and responded all of the queries. The queries were generated by the 10 client computers which can be seen on the bottom of the figure. These computers played the role of the large number of the clients. The clients sent their queries by 6to4 through the 6to4 relay router to the “internet” computer. These queries were generated different levels of load on the 6to4 relay computer during the measurement process. The load was tuned by the number of the active clients. The laptop and the connecting switch on the right side of the figure were used for the control of the experiments.

B. Hardware Configurations

1000Base-TX connections were used on all of the network segments.

A specially low performance computer was built for the 6to4 relay computer so that the client computers could produce high enough load for overloading it. The main goal of the measurements was the comparison of the different implementations and not any hardware related investigation.

The 6to4 relay computer had an Intel D815EE2U motherboard, an Intel Pentium III (800MHz) processor, 128MB (100MHz) SDRAM and two TP-LINK TG-3269 REV 3.0 Gigabit PCI Ethernet NICS.

All of the ten clients and the responder computer were Dell Precision 490 workstations with same configuration: DELL 0GU083 motherboard with Intel 5000X chip-set, two Intel Xeon 5140 2.33GHz dual core processors (in the responder: Intel Xeon 5160 3GHz), 4x1GB 533MHz DDR2 SDRAM (accessed quad channel) and Broadcom NetXtreme BCM5752 Gigabit Ethernet controller (PCI Express).

C. Software configurations

Debian Linux 6.0.7 with 2.6.32-5-amd64 kernel and OpenBSD 5.3 64 bit version were installed on the clients, and the responder, respectively.

On the responder, NAT66 was used to simulate servers with different IPv6 addresses. The packets of the clients were redirected the computer itself, so that it can reply to them.

All of the client computers used sit interfaces and they used the network settings shown in Fig 1.

V. MEASUREMENT METHOD

The load was generated by pings6 commands with the following Bash shell script:

```
#!/bin/bash
i=`cat /etc/hostname | grep -o '^[0-9]'`
for b in {0..255}
do
  rm -rf $b
  mkdir $b
  for c in {0..255}
do
    echo -n $c
  done
done
```

Fig. 1. Topology of the test network
The results are shown in Table I. The first row shows the number of clients that executed the test script at the same time. The generated load on the 6to4 relay was proportional with the number of the clients. The second row contains the packet loss ratio. Rows 3, 4 and 5 show the average, the standard deviation and the maximum value of the response time, respectively. The average and the standard deviation of the CPU utilization of the 6to4 relay computer are shown in the rows 6 and 7. Row 8 contains the memory consumption of the 6to4 process on the relay computer. (This parameter can be measured with high uncertainty, because its value is very low and other processes than the 6to4 relay implementation may also influence the size of the used memory of the computer.) The last row shows the number of forwarded packets per seconds.

The graphical representation of the forwarded packets per second and the CPU utilization are shown in Fig. 2.

### Table I. Debian Linux -- sit 6to4 Relay Performance Results

<table>
<thead>
<tr>
<th>Number of clients</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet loss (%)</td>
<td>0.02</td>
<td>0.006</td>
<td>0.008</td>
<td>0.013</td>
<td>0.020</td>
<td>0.035</td>
<td>0.035</td>
<td>0.037</td>
<td>0.048</td>
<td>0.061</td>
</tr>
<tr>
<td>Response time (ms)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.287</td>
<td>0.353</td>
<td>0.445</td>
<td>0.566</td>
<td>0.710</td>
<td>0.868</td>
<td>1.043</td>
<td>1.209</td>
<td>1.411</td>
<td>1.626</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>0.174</td>
<td>0.248</td>
<td>0.353</td>
<td>0.423</td>
<td>0.509</td>
<td>0.588</td>
<td>0.685</td>
<td>0.722</td>
<td>0.832</td>
<td>0.864</td>
</tr>
<tr>
<td>Maximum</td>
<td>27.900</td>
<td>28.400</td>
<td>28.500</td>
<td>28.900</td>
<td>29.400</td>
<td>30.700</td>
<td>31.100</td>
<td>34.100</td>
<td>32.800</td>
<td>39.600</td>
</tr>
<tr>
<td>CPU Utilization (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>1.756</td>
<td>4.821</td>
<td>12.933</td>
<td>31.243</td>
<td>52.964</td>
<td>69.049</td>
<td>81.319</td>
<td>88.941</td>
<td>93.206</td>
<td>96.132</td>
</tr>
<tr>
<td>Memory consumption (KB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std. dev.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traffic volume (packets/sec)</td>
<td>18051</td>
<td>33953</td>
<td>46856</td>
<td>56534</td>
<td>62853</td>
<td>69947</td>
<td>69663</td>
<td>72304</td>
<td>73129</td>
<td>73050</td>
</tr>
</tbody>
</table>

### Table II. OpenWrt (Attitude Adjustment) 12_09_ x86 – sit 6to4 Relay Performance Results

<table>
<thead>
<tr>
<th>Number of clients</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet loss (%)</td>
<td>0.004</td>
<td>0.006</td>
<td>0.007</td>
<td>0.013</td>
<td>0.018</td>
<td>0.026</td>
<td>0.036</td>
<td>0.064</td>
<td>0.079</td>
<td>0.089</td>
</tr>
<tr>
<td>Response time (ms)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.314</td>
<td>0.402</td>
<td>0.568</td>
<td>0.733</td>
<td>0.909</td>
<td>1.118</td>
<td>1.358</td>
<td>1.616</td>
<td>1.873</td>
<td>2.160</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>0.161</td>
<td>0.239</td>
<td>0.330</td>
<td>0.420</td>
<td>0.508</td>
<td>0.583</td>
<td>0.652</td>
<td>0.705</td>
<td>0.773</td>
<td>0.829</td>
</tr>
<tr>
<td>Maximum</td>
<td>25.000</td>
<td>25.300</td>
<td>25.500</td>
<td>25.500</td>
<td>26.500</td>
<td>27.100</td>
<td>27.000</td>
<td>27.100</td>
<td>27.300</td>
<td>28.100</td>
</tr>
<tr>
<td>CPU Utilization (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>10.067</td>
<td>45.015</td>
<td>70.713</td>
<td>87.188</td>
<td>94.979</td>
<td>97.540</td>
<td>98.467</td>
<td>98.916</td>
<td>99.086</td>
<td>99.288</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>3.188</td>
<td>5.593</td>
<td>5.828</td>
<td>9.376</td>
<td>7.954</td>
<td>7.462</td>
<td>4.991</td>
<td>4.567</td>
<td>4.824</td>
<td>4.410</td>
</tr>
<tr>
<td>Traffic volume (packets/sec)</td>
<td>17595</td>
<td>32488</td>
<td>41906</td>
<td>49270</td>
<td>54196</td>
<td>56920</td>
<td>58272</td>
<td>59828</td>
<td>59332</td>
<td>58763</td>
</tr>
</tbody>
</table>

### Table III. FreeBSD 9.1_ x86 -- stf_6to4 Relay Performance Results

<table>
<thead>
<tr>
<th>Number of clients</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet loss (%)</td>
<td>0.013</td>
<td>0.008</td>
<td>0.010</td>
<td>0.012</td>
<td>0.013</td>
<td>0.015</td>
<td>0.017</td>
<td>0.018</td>
<td>0.019</td>
<td>0.019</td>
</tr>
<tr>
<td>Response time (ms)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.315</td>
<td>0.456</td>
<td>0.681</td>
<td>0.941</td>
<td>1.268</td>
<td>1.637</td>
<td>2.011</td>
<td>2.385</td>
<td>2.740</td>
<td>3.126</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>0.111</td>
<td>0.171</td>
<td>0.314</td>
<td>0.404</td>
<td>0.450</td>
<td>0.457</td>
<td>0.463</td>
<td>0.466</td>
<td>0.480</td>
<td>0.490</td>
</tr>
<tr>
<td>CPU Utilization (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std. dev.</td>
<td>6.899</td>
<td>5.140</td>
<td>6.465</td>
<td>7.398</td>
<td>7.593</td>
<td>3.447</td>
<td>5.336</td>
<td>6.445</td>
<td>5.971</td>
<td>0.475</td>
</tr>
<tr>
<td>Memory consumption (KB)</td>
<td>0.008</td>
<td>0.012</td>
<td>0.012</td>
<td>0.273</td>
<td>0.395</td>
<td>0.398</td>
<td>0.445</td>
<td>0.406</td>
<td>0.500</td>
<td>0.492</td>
</tr>
<tr>
<td>Traffic volume (packets/sec)</td>
<td>17594</td>
<td>30656</td>
<td>37613</td>
<td>41982</td>
<td>43681</td>
<td>43875</td>
<td>43819</td>
<td>43970</td>
<td>43737</td>
<td>43737</td>
</tr>
</tbody>
</table>
Evaluation of the results:

Despite the fact that packet loss occurred in all cases, the proportion of it was always very low and it increased with more clients. (The maximum value of it was 0.061\% with ten clients, which means about 6 packets from 10,000 packets were lost.)

The average, the standard deviation and the maximum value of the response times were increasing with higher load on the 6to4 relay computer, but the average value did not exceed 1.63 milliseconds with ten clients.

The CPU utilization were increasing continuously, but not linearly.

The deviation of the CPU utilization were higher with 4, 5, 6 and 7 clients, which indicates some fluctuation in the utilization.

The memory consumption was almost constant and very low, and the maximum value of it was 11.14kB with ten clients.

The traffic volume increased until the system reached its limit with 9 clients. With 10 clients, the number of transferred packets were slightly decreased from 73129 to 73050.

A. OpenWRT (Attitude Adjustment) 12.09_x86 – sit

The results have been listed in Table II, whereas the graphical representation of the forwarded packets per second and the CPU utilization are shown in Fig. 3.

Evaluation of the results:

The packet loss ratio was always very low and it strictly increased with the number of clients. The maximum value of it was 0.089\% with ten clients.

The average and the standard deviation value of the response times were increasing with higher load on the 6to4 relay computer, but the average value did not exceed 2.16 milliseconds with ten clients.

The CPU utilization with two clients was 4.5 times greater than the value of one client. Then the slope was reduced, until the CPU approached its maximum capacity with 6 clients.

The standard deviation of the CPU utilization were under 10\% in each case, which indicates consistent utilization of the CPU.

The memory consumption was almost constant and very low, and the maximum value of it was 10.75kB with seven clients.

The traffic volume increased until the system reached its limit with 9 clients. With 10 clients, the number of transferred packets were decreased by 0.97\% from 59332 to 58763.

B. FreeBSD 9.1_x86 – stf

The results have been listed in Table III, whereas the graphical representation of the forwarded packets per second and the CPU utilization are shown in Fig. 4.

Evaluation of the results:

The packet loss ratio was always very low and starting from two clients it increased with the number of clients, whereas the value of it was the same with one and five clients. The maximum value of it was 0.019\% with ten clients.

The average and the standard deviation value of the response times were increasing with higher load on the 6to4 relay computer, but the average value did not exceed 3.13
millions with ten clients. The maximum value of the response times showed some fluctuation.

One client could generate 51.53% load on the CPU. The CPU utilization was increasing continuously, but not linearly, until the CPU reached its almost maximum capacity (99.44%) with 6 clients.

The standard deviation of the CPU utilization was under 10% in each case, whereas it was very small (0.46%) with ten clients. This phenomenon indicates consistent utilization of the CPU.

The memory consumption was extremely low and it was growing almost continuously.

The traffic volume increased until the system reached its limit with 6 clients. From this point the throughput of the system started very slightly fluctuating. The maximum value of the number of transferred packets per second was 43970 with 9 clients.

VII. COMPARISON OF THE RESULTS

All of the tested implementations proved to be reliable and the packet loss ratios of the different implementations were always very low. The packet loss ratio of the Linux and OpenWrt implementations seriously increased with the number of clients, whereas the packet loss of FreeBSD stf did not show much increase in the function of the load.

Under high load conditions, Linux sit tunnel forwarded the most packets per second and OpenWrt sit was the second one. The FreeBSD system was the last competitor in the performance comparison. At 10 clients, Linux outperformed FreeBSD by 1.67 times.

All of the implementations use negligibly small amount of memory, which is usually proportional to the generated load.

With one client, all of the implementations forwarded similar number of packets, but with significantly different CPU utilization, which property can explain the high degree of difference in the performance with more clients. Linux sit 6to4 relay implementation used 1.76% of CPU with one client, whereas FreeBSD stf used 51.53%, which means about 29 times difference.

VIII. CONCLUSION

The 6to4 protocol is a useful transition technique in a situation, where two IPv6 enabled hosts have to communicate over an IPv4 only network. All of the tested open source 6to4 relay implementations are viable solutions in production networks, but Linux sit showed the best performance characteristics, whereas OpenWrt sit was the second best one. In an environment, where a BSD system is preferred, FreeBSD stf is a usable solution as well.

The authors believe that their work has contributed to the early adoption of the IPv6 protocol and the published results and methodology are valuable for both researchers and network professionals.

ACKNOWLEDGEMENT

The measurement of the performances of the different 6to4 implementations was the BSc thesis (final project) work of the second author at the Dept. of Telecommunications, Széchenyi István University under the supervision of the third author.

REFERENCES

[10] G. Lencse and S. Répás, “Performance analysis and comparison of different DSN64 implementations for Linux, OpenBSD and FreeBSD,” in Proc. 27th IEEE Int. Conf. on Advanced Information Networking and Applications (AINA-2013), Barcelona, 2013, pp. 877-884. DOI: 10.1109/AINA.2013.80