Investigating the IPv6 Teredo Tunnelling Capability and Performance of Internet Clients

Sebastian Zander,
Lachlan L. H. Andrew,
Grenville Armitage
CAIA, Swinburne University of Technology
Melbourne, Australia
{szander,landrew,garmitage}@swin.edu.au

Geoff Huston,
George Michaelson
Asia Pacific Network Information Centre (APNIC)
Brisbane, Australia
{gih,ggm}@apnic.net

ABSTRACT

The Teredo auto-tunnelling protocol allows IPv6 hosts behind IPv4 NATs to communicate with other IPv6 hosts. It is enabled by default on Windows Vista and Windows 7. But Windows clients are self-constrained: if their only IPv6 access is Teredo, they are unable to resolve host names to IPv6 addresses. We use web-based measurements to investigate the (latent) Teredo capability of Internet clients, and the delay introduced by Teredo. We compare this with native IPv6 and 6to4 tunnelling capability and delay. We find that only 6–7% of connections are from fully IPv6-capable clients, but an additional 15–16% of connections are from clients that would be IPv6-capable if Windows Teredo was not constrained. However, Teredo increases the median latency to fetch objects by 1–1.5 seconds compared to IPv4 or native IPv6, even with an optimally located Teredo relay. Furthermore, in many cases Teredo fails to establish a tunnel.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network Monitoring*; C.4 [**Performance of Systems**]: Measurement Techniques

Keywords

IPv6, Teredo

1. INTRODUCTION

The IETF standardised several protocols that tunnel IPv6 across IPv4-only networks. Tunnelling protocols, such as 6to4 [1], require that the tunnel endpoints have public IPv4 addresses. But due to the IPv4 address shortage Network Address Translators (NATs) are now widely deployed in front of home networks [2], and not all home gateways support IPv6 tunnelling protocols.[1] Huitema *et al*. developed the Teredo protocol [3, 4], which allows IPv6 hosts behind IPv4 NATs to communicate with native IPv6 hosts or other IPv4 NATed IPv6-capable hosts. Other advantages of Teredo are that it only requires one relay on the path between two hosts (6to4 often requires one relay in each direction due to routing), and it is more firewall-friendly (6to4 uses protocol 41, which is often filtered). Furthermore, Teredo provides a general method for NAT traversal that can be used by many applications.

[1]It is unclear how many home gateways support 6to4. However, in our dataset almost 45% of connections were from Windows Vista and 7 clients that have 6to4 enabled by default and preferred over Teredo, yet only 4–5% of connections used 6to4. This indicates that 6to4 is often deactivated, filtered or not supported.

Teredo is supported by Windows since Windows XP and Windows Server 2003 [5], and is enabled by default on Windows Vista and Windows 7. Implementations for Linux, BSD variants and MacOS X also exist (e.g. Miredo [6]). However, on Windows Vista and Windows 7 by default Teredo is self-constrained: without native IPv6 network interface both operating systems (OSs) do not resolve host names to IPv6 addresses (do not send DNS AAAA requests) [7]. This leaves Teredo enabled for client-side applications that use it directly, but otherwise avoids unexpected behaviour in home networks, which are not prepared to handle DNS AAAA requests or IPv6 traffic [8]. Previous studies based on passive measurements [9–11] could not detect latent Teredo capability and also could not measure the latency introduced by Teredo.

We use web-based measurements to measure the IPv6 capability of clients, including latent Windows Teredo capability based on Steffann's technique [12]. On certain participating web sites the clients' web browsers not only download the "normal" web pages and images, but also a test script that fetches a few very small invisible images from our test server. One image can only be retrieved with IPv4, a second only with IPv6 and a third with either IPv4 or IPv6. To test Windows latent Teredo capability a fourth image is behind an IPv6 literal URL. The test script also measures the fetch latencies for these images, so we can analyse the latency introduced by Teredo and compare it with the latency of IPv4 or native IPv6.

Previously this measurement method limited the test sample to clients visiting participating web sites. However, we developed a novel method for embedding the test script in Flash ad banners, and use Google's AdSense platform to distribute the test to "random" clients. Our overall dataset is a combination of both data collection methods, giving us a broader sample without being limited to clients running Flash. To reduce the sampling error we re-weight our statistics based on global traffic data.

The paper is organised as follows. Section 2 outlines related work. Section 3 provides an overview about the Teredo protocol. Section 4 describes our experimental measurement setup. Section 5 describes the processing of the data and presents the results. Section 6 concludes and outlines future work.

2. RELATED WORK

Over the last decade a number of researchers studied the progress of IPv6 deployment based on measurements at web servers, passive traffic measurements, topology measurements, and routing or DNS root server data. The survey paper [13] provides an excellent summary and also identifies areas that need further study. Only a few of the previous studies investigated Teredo.

Huang *et al*. [14], Hoeher *et al*. [15], and Aazam *et al*. [16] studied the performance of Teredo in testbeds. They compared the

latency and throughput of Teredo with other IPv6-over-IPv4 tunnelling technologies. Since their measured latencies are based on testbeds, they are not comparable with our results.

Malone [9] studied the fraction of native, 6to4, 6bone and Teredo IPv6 addresses observed at three servers (WWW, FTP, DNS) in 2007 and found that only 10% of IPv6 connections (0.04% of all connections) used Teredo. Karpilovsky *et al*. [10] quantified the IPv6 deployment based on NetFlow data. They found that in 2008 Teredo addresses accounted for only 4% of all IPv6 addresses. Defeche and Wyncke [11] studied the use of IPv6 in the BitTorrent file-sharing community. The fraction of IPv6-capable BitTorrent peers was only 1%, but 59% of them used Teredo.

The previous studies used passive measurements (NetFlow records or log files), and could not observe constrained Windows Teredo clients. Also, with NetFlow records it is unclear what fraction of the observed Teredo traffic is data exchange, or tunnel setup and maintenance. Our measurement technique allows detecting latent Teredo clients and differentiating between failed and successful tunnel setups. The data used in previous studies was collected at very few locations and may be biased heavily. We collected data from a larger number of sites and via Google ads, and we also use a technique to mitigate sampling error. Finally, unlike previous studies, we analyse Teredo's setup latency in the real world.[2]

3. TEREDO OVERVIEW

Tunnelling protocols, such as 6to4 [1], require that tunnel endpoints have public IPv4 addresses. But most home networks are behind IPv4 NATs and home gateways may not be 6to4-capable. Teredo is an auto-tunnelling protocol, enabling IPv6 hosts behind IPv4 networks to communicate with other IPV6 hosts, even if they are behind NATs and have no public IPv4 addresses [3, 4].

3.1 Teredo Protocol

The Teredo architecture consists of *clients*, *servers* and *relays*. Teredo clients run on IPv6-capable hosts behind NATed IPv4 networks. Teredo clients use Teredo servers for NAT detection, address configuration, and tunnel setup. Teredo servers are not part of the tunnels; no data traffic is routed across them and they can be stateless. Teredo relays act as IPv6 routers and forward data traffic between Teredo clients and native IPv6 hosts.

Initially every Teredo client performs a *qualification* procedure by exchanging messages with a Teredo server [3]. During the qualification the client determines whether it is behind a NAT, the type of NAT, and its public IPv4 address. If Teredo can be used through the detected NAT[3], the client assigns itself a Teredo IPv6 address that looks as follows [3]. Address bits 0–31 are the Teredo prefix (2001:0::/32), bits 32–63 are the IPv4 address of the Teredo server, bits 64–79 are flags, bits 80–95 contain the client's external port (obfuscated) and bits 96–127 contain the client's external IPv4 address (obfuscated). The qualification is repeated from time to time (by default every 30 seconds), since the NAT binding may change over time.

Figure 1 illustrates the Teredo protocol for our experimental setup, where NATed clients communicate with a web server connected to an IPv6 network. When an application sends packets to an IPv6 host, the Teredo client on the sending host queues the packets, performs the qualification (if not already qualified) and then performs a direct connectivity test. It sends an ICMPv6 echo request from its Teredo IPv6 address to the IPv6 destination encapsulated in a UDP/IPv4 packet sent to the Teredo server. The server decapsulates the ICMPv6 packet and sends it to the destination via IPv6. The destination responds with an ICMPv6 echo reply destined to the client's Teredo IPv6 address. The echo reply is routed to the nearest Teredo relay (relays advertise 2001:0::/32).

[2]We published a preliminary analysis of Teredo on the web [17].
[3]Clients behind symmetric NATs cannot use the original protocol, but extensions exist for some symmetric NAT scenarios [4].

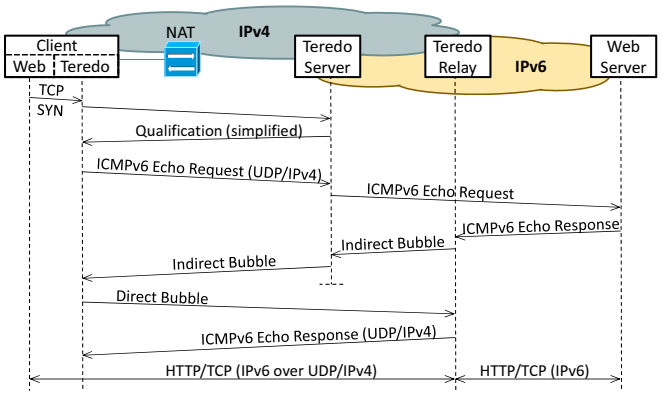

Figure 1: Teredo client tunnel setup to native IPv6 host

The relay queues the ICMPv6 echo reply and sends a so-called indirect bubble to the Teredo server. An indirect bubble is an IPv6 header destined to the client inside a UDP/IPv4 packet with an origin indication field containing the relay's IPv4 address and port. The server forwards the bubble to the client. The client then sends a direct bubble (IPv6 header inside UDP/IPv4) to the relay, which creates an entry in the client's NAT. After the relay receives the client's bubble, it sends the ICMPv6 echo reply encapsulated in UDP/IPv4 to the client. When the client receives the echo reply, the tunnel setup is complete. The client can exchange IPv6 packets with the web server over the UDP/IPv4 tunnel.

3.2 Teredo Implementations

In Windows Vista and Windows 7 Teredo is enabled by default, but it is constrained. Neither OS will query for DNS AAAA records if there are only network devices with link-local or Teredo IPv6 addresses [7] – any communications where host names are specified fail. For example, with a web browser Teredo works with literal IPv6 URLs, but not with the more commonly used host name URLs. Only changing a Windows registry parameter or configuring another interface with public IPv6 address will fully enable Teredo. Windows XP (SP 1 and higher) supports Teredo, but it is disabled by default. In managed networks with configured domain controllers Windows Teredo automatically deactivates.

Teredo implementations for Linux, BSD-variants and MacOS X also exist, most notably Miredo [6]. However, in general Teredo is not part of the default installation on these OSs and needs to be installed and enabled manually.

4. EXPERIMENTAL SETUP

We use active web-based measurements based on [12, 18] to measure whether hosts can use Teredo, 6to4, or native IPv6, prefer IPv6 over IPv4 in dual stack and measure the fetch latencies.

4.1 Web-based measurements

When users visit web sites their web browsers normally download several web pages, scripts, images etc. At certain participating web sites this includes a small test script that fetches a few invisible one-pixel images via HTTP from URLs pointing to our test web server and measures the fetch times. We refer to the script as

test and a single image download as *sub-test*. For URLs with host names the client's resolver has to perform DNS look-ups against our test DNS server prior to sending HTTP requests. Different sub-tests allow us to test IPv4 and IPv6 connectivity, dual stack behaviour, and (latent) Teredo behaviour:

1. The image can only be retrieved with IPv4 because the DNS server only returns an A record (IPv4-only);
2. The image can only be retrieved with IPv6 because the DNS server only returns an AAAA record (IPv6-only);
3. The image can be retrieved with IPv4 or IPv6 because the DNS server returns A and AAAA records (dual-stack);
4. The image is behind an IPv6 address literal URL (Windows hosts with only half-enabled Teredo will use it).

A sub-test is deemed successful if the image could be retrieved; otherwise it is deemed a failure. For each sub-test the test script measures the time it takes to fetch the image (with millisecond precision). For failed sub-tests the latency is undefined. After all sub-tests have been successfully completed or a timeout of ten seconds (whichever occurs first) the test script sends another HTTP request to the test web server that acts as a *test summary*.

The test summary reports the sub-test latencies and allows us to determine whether a browser has waited a sufficient amount of time for all sub-tests to complete. Since the test script stops when users move to another page, which can happen quickly, without the summary it is impossible to know whether a sub-test was not successful because a client lacked the capability or because the test script was interrupted.

The test script starts sub-tests in quick succession, but to which degree the images are fetched in parallel depends on a web browser's connection management. We assume that browsers try to fetch objects as quickly as possible without unnecessary delay. The URLs for each sub-test and summary resolve to different IPv4 or IPv6 addresses, which means browsers cannot multiplex different sub-tests over one TCP connection.

Figure 2 shows a logical diagram of our experimental setup. The DNS server handles the incoming DNS queries and the web server serves the test images. All servers are time-synchronised with NTP. Local Teredo and 6to4 relays are located close to the web server to reduce tunnelling delay as much as possible (Teredo server and client-side 6to4 relay are out of our control).[4] We collect HTTP log data, DNS log data, and capture the traffic with tcpdump.

Several data fields are used to convey information from a tested client to our test servers and prevent caching of DNS records or test images at the client or intermediate caches (see Figure 2). The following data is prepended to the host name *as well as* appended as URL parameters: Test time, test ID, test version, and sub-test name. Test time is the time a test started taken from the client's clock (Unix time plus milliseconds). Test ID is a "unique" random 32-bit integer number determined by the test script. Test version is the test's version number and sub-test name is a unique identifier for the sub-test, e.g. "IPv4-only".

4.2 Client sample

Clients interact with our measurement system in two different ways. A number of participating web sites link our JavaScript test script (*JS-test*) and visiting hosts are tested. The client sample is biased towards the participating web sites, but since the test is implemented in JavaScript the vast majority of visiting clients can be tested (we assume there are not many clients with disabled JavaScript, as it has become essential for many web pages).

[4]A Teredo relay close to the web server does not add additional tunnelling delay during the data exchange. In reality the delay may be increased, if the relay is off the lowest-latency path [17].

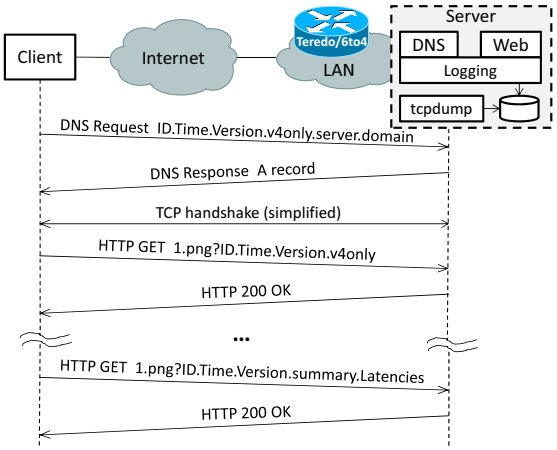

Figure 2: Experimental setup

We also implemented a Flash ActionScript test and embedded it in a Flash ad banner (*FA-test*). The ad is served to hosts via Google's AdSense. The test is carried out as soon as the ad is *displayed* by the web browser, the user does not have to click on it. We selected the ad's keywords and languages to achieve broad placement across different countries and sites. The FA-test reaches a more diverse client population, but cannot be carried out by clients without Flash (e.g. iPhones/iPads during our measurement).

The JS-test uses cookies to ensure clients are only tested once per day per web site. The FA-test cannot use cookies, so clients may be tested multiple times. In both cases there can be proxies or NATs, which look like repeating clients. However, our analysis shows that under 8% of IPs perform the test more than once and under 0.1% of IPs perform the test more than 10 times per day.

4.3 User impact

The JS-test script is 12 kB large and executed after a web page has loaded completely. Hence, it does not delay the loading of the page. The loading of the FA-test is the same as for other Flash ads. Our ad is 16 kB large, well below Google's size limit of 50 kB. The test images are only 157 bytes large. The total size of the test data transferred is well under 20 kB, very small compared to the average web page size. We argue that overall our test does not have a significant impact on a user's browsing experience, which we confirmed with test users. Also, our test does not trick users into revealing any sensitive information. Instead, we utilise information that web clients normally send to *every* visited web server.

5. TEREDO ANALYSIS

5.1 Methodology

Our analysis is over the time period between 16th of May 2011 and 19th of February 2012. We pre-processed the raw data as follows. We discarded 1% of tests as "invalid", such as tests without latency values for any of the sub-tests. We extracted "completed" tests where the web server received the test summary. Only for these we have the latencies and can be sure that the clients tried long enough. On average there were 180 000–200 000 valid completed tests per day (of which only 30 000–35 000 were FA-tests due to our limited ad budget and the rest were JS-tests). We anal-

ysed the data in blocks of 24 hours, but we will present the statistics averaged over weekly periods to reduce the error of the estimates.

We interpret the measurement results as *statistics of connections* (connections in the sense of connectivity) and not of clients (IPs) to avoid "observation bias" caused by multiple clients behind web proxies or NATs, or clients that change their IP addresses more often (home users, frequently offline clients). Furthermore, we avoid potential bias because we are more likely to observe clients that are more likely to use the Internet and potentially have more up-to-date systems and are more up to date with IPv6.

Our sample of tested clients is biased towards the web sites activating the JS-tests and FA-tests. For example, Indonesian JS-test connections are well over-represented due to a large participating Indonesian web site. Furthermore, the participating web sites change over time. To mitigate this bias we weight each test based on the tested client's country. The weight for a single test W_t is:

$$W_t = P_c \frac{T}{T_c},$$

where P_c is the weight of country c ($\sum P_i = 1$), T is the total number of tests, and T_c is the number of tests of country c . Our weights P_c are based on country traffic statistics estimated by Cisco [19] for 16 countries that generate 79% of the Internet's traffic and Wikipedia country traffic statistics [20] for the remaining countries. We use MaxMind's GeoLite country database [21] to map IPv4 client addresses to countries (claimed accuracy of 99.5%). Our approach is described in more detail in [18]. All the results presented in Section 5 are based on the re-weighted data.

There still may be a bias towards clients visiting the particular set of web sites, but our set of sites is relatively diverse and large. There are 55–75 different domains (covering universities, ISPs, gaming sites, blog sites) that refer at least 100 tested clients per day. Furthermore, we compared conclusions from JS-tests and FA-tests and found the conclusions match well qualitatively [18].

5.2 Results

We analyse the percentage of Teredo connections, the percentage of successful and failed Teredo connections per OS, the latency introduced by Teredo and the reasons for Teredo failures.

5.2.1 *Latent Teredo capability*

Figure 3 shows the weekly percentages of Teredo, 6to4 and native IPv6 connections, as identified by the IPv6 address prefix (native connections possibly include point-to-point IPv6-over-IPv4 tunnels). Our test period included World IPv6 day (June 8th, 2011), during which major Internet network and service providers enabled IPv6 as a test. For native IPv6 the week of the World IPv6 day shows a little bump and percentages are slightly higher until mid July 2011. Conversely, for Teredo there is a dent around IPv6 day. The percentage of Teredo and native IPv6 are relatively constant, but the percentage of 6to4 shows a slight decrease.

Of the 15–16% Teredo connections, 99.8% were from Windows Vista and 7 hosts with latent IPv6 capability. These hosts could be forced to use Teredo with a literal URL, but would not perform DNS AAAA requests and hence were not truly IPv6-capable. Only 0.1–0.2% of Teredo connections were from IPv6-capable hosts (manually tuned Windows Teredo or Teredo on other OS). Clients using 6to4 accounted for 4–5% of connections and 1–2% of connections were from clients with native IPv6. Only clients with native IPv6 preferred IPv6 in dual-stack (90–100% preferred IPv6). The proportions of 6to4 and Teredo connections that preferred IPv6 were under 1% and well under 0.1% respectively. Now we focus on Teredo, for other results see [18].

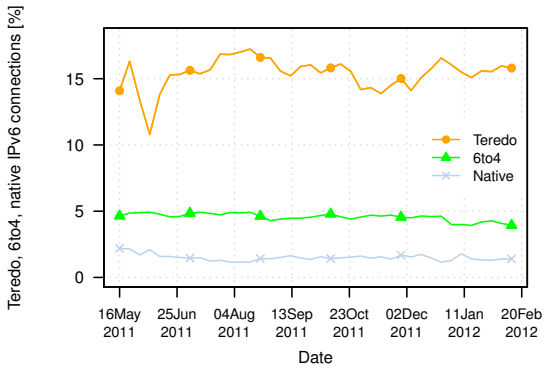

Figure 3: Weekly percentages of Teredo, 6to4 and native IPv6 connections

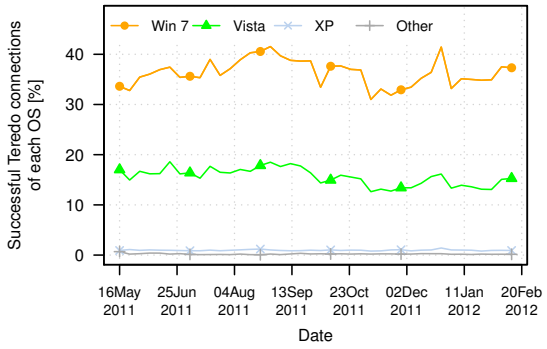

Figure 4: Weekly percentages of successful Teredo connections for each OS

5.2.2 *Successful/failed Teredo*

Figure 4 shows the percentages of successful Teredo connections for each OS. For Windows 7 roughly 30–40% of connections were successful whereas for Windows Vista only 10–20% of connections were successful. For Windows XP and other OS (mostly Linux and MacOS X) the percentages were 1% and 0.2% respectively.

Figure 5 shows the percentages of observed failed Teredo connections for each OS. These are connections where clients started the tunnel setup (ICMPv6 echo request arrived at server) but could not complete it. This could be related to issues with creating the hole in the NAT, but we think a major cause is excessive delay introduced by Teredo. For Windows 7 and Windows Vista the percentages are 20–30%, for Windows XP and other OS the percentages are 0.6% and under 0.1% respectively. For Windows 7 and Windows Vista for roughly 33% and 60% of connections respectively there was no sign of Teredo. Either Teredo was disabled, the qualification failed or took too long, or there were other causes, such as corporate firewalls blocking UDP or misbehaving NATs.

Figure 6 shows the average weekly cycle of the percentage of all successful Teredo connections. For each test we computed the client-local test time based on GeoIP time zone information and averaged the hourly percentages. The percentage of successful Teredo was higher on weekdays during non-working hours and on weekends. We assume during these times there was a higher percentage of home users, and Teredo more likely worked for home users (no domain controllers or corporate firewalls). Qualitatively the observed failed Teredo connections have a similar cycle.

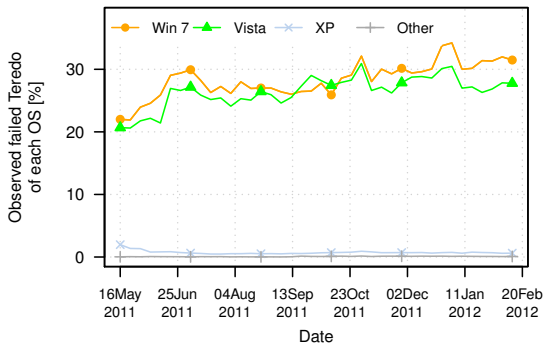

Figure 5: Weekly percentages of observed failed connections (tunnel setup was started but not completed) for each OS

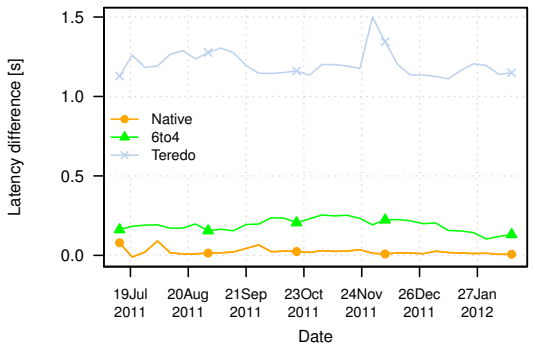

Figure 7: Weekly median fetch latency difference between Teredo, 6to4, native IPv6 and IPv4

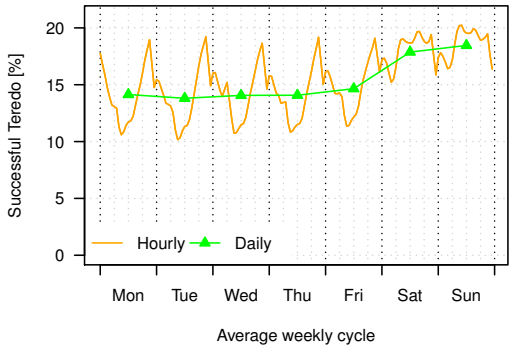

Figure 6: Average hourly/daily successful Teredo percentage

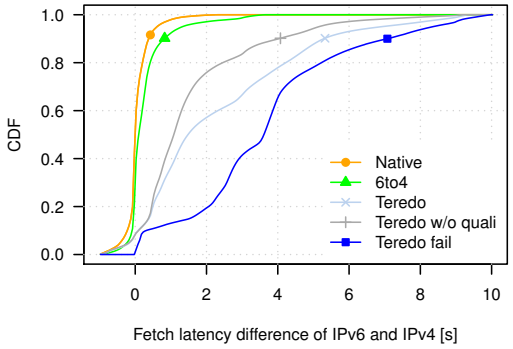

Figure 8: Fetch latency difference between native IPv6, 6to4, Teredo (with and without qualification) and IPv4. Latency difference between observed failed Teredo and IPv4.

5.2.3 *Fetch latency*

Figure 7 shows the median fetch latency difference between the different IPv6 techniques (Teredo, 6to4, native IPv6) and IPv4 connections measured by the clients.[5] Since the measured latencies include the time for DNS resolution for all sub-tests except literal sub-tests (almost all Teredo), we added the estimated time for DNS resolution to the measured Teredo latencies. As estimate we use the time difference between the TCP SYN of the HTTP connection reaching the web server and the DNS query of the client's DNS server reaching the DNS server for IPv4 connections. Since the DNS hierarchy adds latency our estimate is conservative. Compared to IPv4 the median fetch latency of native IPv6 is less than 30 ms higher most of the time, and the median fetch latency of 6to4 is 150–250 ms higher. However, the median fetch latency of Teredo is 1–1.5 seconds higher. For Teredo it seems the latency difference is trending down slightly over time.

Figure 8 shows CDFs of the fetch latency differences over the whole time. For native IPv6 the tail of the distribution is likely caused by IPv6 point-to-point tunnels, whereas for 6to4 the tail is likely caused by sub-optimally located relays. For Teredo the tail is much longer than for native IPv6 or 6to4 and extends to multiple seconds. Figure 8 also shows the estimated latency difference for observed failed Teredo connections. Despite not even making it to a TCP SYN, in many cases the failed attempts took even longer. The much higher delay for Teredo, especially failed Teredo, is due to a number of factors we now discuss.

After starting Windows the Teredo client is dormant and it must perform the qualification procedure before the first tunnel setup, which takes at least two round trip times between Teredo client and server. Afterwards the Teredo client will stay active and periodically re-qualify. Since many hosts probably do not use Teredo, we assume often our tests wake up dormant clients. Figure 8 also shows a conservative estimate of the Teredo fetch latency difference if clients would have been already qualified when the test started. (The Windows 7 Teredo client starts generating ICMPv6 echo requests when the first TCP SYN from the web client is queued, but only sends the echo requests *after* qualification. If qualification takes long this causes *back-to-back* echo requests we use as indicator that clients had to qualify.)

Even with an optimally located Teredo relay the initial tunnel setup still has to go through the Teredo server. The default Windows Teredo servers are located in the US and UK, so for clients further away from the default servers the tunnel setup adds significant delay. Teredo tunnels only stay up for 30 seconds without traffic (typical UDP NAT timeout), so it is likely that clients had to establish tunnels before they could fetch the test image, but in reality tunnel setup would also occur frequently.

5.2.4 *Failures depending on Teredo servers*

The worst problem only became apparent after a number of experiments with a Windows 7 Teredo client. In many cases we observed the loss of one or more packets during the qualification and tunnel setup. The packet loss causes huge delays, for example one lost ICMPv6 echo request/response adds 2 seconds of delay. This explains the longer delays of observed failed connections in Figure 8. Many of these probably effectively failed due to long delays. By

[5]Later start date because we only kept DNS logs since July 2011.

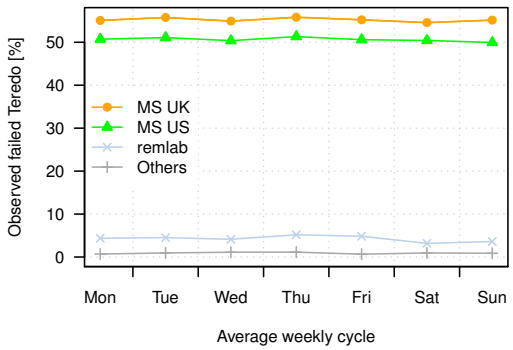

Figure 9: Average observed failed Teredo connections as percentage of all observed Teredo connections for different Teredo servers

default a Windows Teredo client only tries to establish a tunnel for 4 seconds (three ICMPv6 echo requests), and only re-retries establishing a tunnel after a 3rd retransmitted TCP SYN from the web client arrives at 9 seconds; but this is close to the 10 second test timeout (and exceeds human patience).

Figure 9 compares the average weekly cycle of observed failed Teredo connections as percentage of all observed Teredo connections for the default Windows MS Teredo servers (used by over 99% of connections), the most popular alternative server (teredo.remlab.net) and an average of all other servers. The average failure rates are much higher for the default servers, possibly due to high load.

6. CONCLUSIONS AND FUTURE WORK

We investigated the Teredo capability of Internet clients including a large proportion of self-constrained Windows Vista and Windows 7 clients using web-based measurements. While slightly over 6% of connections were from fully IPv6-capable clients using native IPv6 or 6to4, more than twice as many connections (15–16%) were from Windows clients that could have used IPv6, if Windows Teredo was fully enabled by default. However, Teredo greatly increased the median delay to fetch objects by 1–1.5 seconds compared to IPv4 or native IPv6. While we expected an increased delay, the increase is unexpectedly high, especially since we had an optimally located Teredo relay. Furthermore, for over 80% of Windows Vista and 66% of Windows 7 connections Teredo failed to establish a tunnel. Dormant Windows Teredo clients and the poor performance of the default Windows Teredo servers seemed to be the major causes. Other causes may have been disabled Teredo clients, firewalls blocking UDP or misbehaving NATs.

Currently Teredo seems limited by a lack of infrastructure. Having only a few and possibly unreliable Teredo servers at a few locations adds significant delay to the qualification and tunnel setup. Reliable and geographically-spread servers are needed to reduce the latency. According to Teredo developers the performance of the default Teredo servers may have been affected by Windows 8 testing [8]. Windows 8 clients, without other IPv6 interfaces, will use Teredo for IPv6-only sites [8], so it seems likely that in the future the quality of the default Teredo servers will be improved.

We will continue our measurements in the future and report any new trends. We also plan to extend our measurement methodology so that we can identify the causes for Teredo failures (disabled Teredo, firewalls or NATs).

Acknowledgements

We thank the anonymous reviewers. This research was supported under Australian Research Council's Linkage Projects funding scheme (project LP110100240) in conjunction with APNIC Pty Ltd and by Australian Research Council grant FT0991594.

7. REFERENCES

[1] B. Carpenter, K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), Feb. 2001.

[2] G. Maier, F. Schneider, A. Feldmann. NAT usage in Residential Broadband Networks. In *Passive and Active Measurement Conference*, pages 32–41, 2011.

[3] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), February 2006.

[4] D. Thaler. Teredo Extensions. RFC 6081 (Proposed Standard), January 2011.

[5] Microsoft Technet. Teredo Overview. `http://technet.microsoft.com/en-us/library/bb457011.aspx`.

[6] R. Denis-Courmont. Miredo: Teredo for Linux and BSD. `www.remlab.net/miredo/`.

[7] Microsoft Technet. DNS Client Behavior in Windows Vista. `http://technet.microsoft.com/en-us/library/bb727035.aspx`.

[8] Private email conversation with Microsoft Teredo developers.

[9] D. Malone. Observations of IPv6 Addresses. In *Passive and Active Measurement Conference (PAM)*, pages 21–30, 2008.

[10] E. Karpilovsky, A. Gerber, D. Pei, J. Rexford, A. Shaikh. Quantifying the Extent of IPv6 Deployment. In *Passive and Active Measurement Conference (PAM)*, pages 13–22, 2009.

[11] M. Defeche, E. Vyncke. Measuring IPv6 Traffic in BitTorrent Networks. IETF draft-vyncke-ipv6-traffic-in-p2p-networks-01.txt, Marc h 2012.

[12] S. Steffann. IPv6 test. `http://ipv6test.max.nl/`.

[13] kc claffy. Tracking IPv6 Evolution: Data We Have and Data We Need. *ACM SIGCOMM Computer Communication Review (CCR)*, (3):43–48, Jul 2011.

[14] S. Huang, Q. Wu, Y. Lin. Tunneling IPv6 through NAT with Teredo Mechanism. In *Conference on Advanced Information Networking and Applications (AINA)*, March 2005.

[15] T. Hoeher, M. Petraschek, S. Tomic, M. Hirschbichler. Evaluating Performance Characteristics of SIP over IPv6. *Journal of Networks*, 2(4):40–50, August 2007.

[16] M. Aazam, S. A. H. Shah, I. Khan, A. Qayyum. Deployment and Performance Evaluation of Teredo and ISATAP over Real Test-bed Setup. In *International Conference on Management of Emergent Digital EcoSystems*, 2010.

[17] G. Huston. Testing Teredo, April 2011. `https://labs.ripe.net/Members/gih/testing-teredo`.

[18] S. Zander, L. L. H. Andrew, G. Armitage, G. Huston, G. Michaelson. Mitigating Sampling Error when Measuring Internet Client IPv6 Capabilities. In *Internet Measurement Conference*, Nov. 2012. (accepted, to appear).

[19] Cisco Systems. Visual Network Index. `http://www.cisco.com/en/US/netsol/ns827/networking_solutions_sub_solution.html`.

[20] Wikipedia. Wikimedia Traffic Analysis Report. `http://stats.wikimedia.org/wikimedia/squids/SquidReportPageViewsPerCountryOverview.htm`.

[21] MaxMind's GeoIP Country Database. `http://www.maxmind.com/app/geoip_country`.