**The ISP Column**
*A monthly column on things Internet*

March 2011
Geoff Huston

# Testing Teredo

It's probably a massive understatement, but the problem of the transition to IPv6 has always been seen as somewhat challenging!

From an industry perspective, for a long time this was a form of a chicken and egg problem, in so far as ISPs saw very little motivation to engage seriously with IPv6 deployment until there was a visible demand from the end systems used by their client base, and the vendors of end systems saw little motivation to equip their systems with IPv6 until there was some IPv6 service deployed by the ISPs. This was the story for many years, and for as long as the spectre of IPv4 exhaustion was something that was comfortably well in the future, then nothing much happened with IPv6.

One early effort intended to break this impasse was the use of tunnelling mechanisms in end systems. The idea was to allow an end system to be IPv6 capable by accessing the IPv6 network via a tunnel through the local IPv4 network, setting up a rendezvous with the IPv6 network at some remote tunnel broker location.

However, this form of explicitly configured tunnelling can be cumbersome as it requires the end user to explicitly set up arrangements with an IPv6 tunnel broker. Given that the basic functionality of a tunnel broker is to tunnel IPv6 packets in IPv4, then all of the tunnel brokers look much the same at some level, and the client systems really don't have a strict requirement to connect to one particular tunnel broker as distinct from any other. Indeed, clients have an interest in connecting with "the closest" tunnel broker, without necessarily knowing which one that happens to be at any time. This inevitably lead to experimentation with "auto-tunnel" solutions, where the tunnel broker function was replicated in many locations in any anycast fashion, and the end user function automatically established a connection with the closest instance of the tunnel broker. This approach was intended to remove all manual configuration on the part of the client.

The first of these forms of auto-tunnelling IPv6 over IPv4 was defined over 10 years ago as *6to4* (see the February 2011 ISP column for a detailed explanation of how 6to4 works). 6to4 does not use TCP or UDP as its transport, but instead uses the IP tunnelling protocol, protocol 41, to carry IPv6 packets in IPv4.

However, there are many issues with 6to4, including the consideration that it does not directly work through NATS (although NATs can act as 6to4 gateways), and the widespread use of end network filters that block protocol 41 packets. The relatively sparse deployment of 6to4 relays has been the cause of widely asymmetric packet paths that, in turn, cause excessive RTT penalties when using 6to4. Measurements of 6to4 connection rates point to a connection failure rate of between 15% to 20% of clients' connection attempts.

Given that we are at a point where we need to engage with transition to IPv6 at a scale of the entire Internet then we we need to insist on IPv6 performance and reliability outcomes that are on a par with the best of IPv4 networks. These poor operational performance and excessive failure rates of 6to4 are a liability. It should be unsurprising that there are now calls to phase out the use of 6to4.

The latest of these efforts to phase out the use of 6to4 in the network is an IETF working document "Request to move Connection of IPv6 Domains via IPv4 Clouds (6to4) to Historic status" (draft-ietf-v6ops-6to4-to-historic-00.txt) by Ole Troan, which makes three quite explicit recommendations:

1. IPv6 nodes should treat 6to4 as a service of "last resort."

2. Implementations capable of acting as 6to4 routers should not enable 6to4 without explicit user configuration. In particular, enabling IPv6 forwarding on a device, should not automatically enable 6to4.

3. If implemented in future products, 6to4 should be disabled by default.

It's also the case that 6to4 was not designed to operate across NATs. Some vendors produced CPE devices that combined 6to4 router functionality with a NAT function, where the 6to4 router would advertise a 2002:xxxx::/48 IPv6 prefix into the local network, based on the 6to4 transform of the "external" IPv4 address, but 6to4 itself is not a protocol that was intended to operate across NATs. The use of protocol 41 in 6to4 makes conventional NAT port and address transforms inapplicable, as while 6to4 uses protocol 41 at the IP level, the common form of NATs, namely port translating NATs, rely on the use of TCP and UDP and the associated port addresses to undertake the NAT function.

Teredo was designed to provide similar functionality to 6to4, but operate successfully behind a NAT.

Teredo is described in a Microsoft technical note: http://technet.microsoft.com/en-us/library/bb457011.aspx.

Teredo, like 6to4, is an auto-tunnelling protocol coupled with an addressing structure. Like 6to4, Teredo uses its own address prefix, and all Teredo addresses share a common IPv6 /32 address prefix, namely 2001:0::/32. The next 32 bits are used to hold the IPv4 address of the Teredo server. The 64 bit IPv6 interface identifier field is used to support NAT traversal and it is encoded with the triplet of a field describing the NAT type, the relay's view of the UDP port number used to reach the client (the external UDP port number used by the NAT binding for the client) and the relay's view of the IPv4 address used to reach the client (the external IPv4 address used by the NAT binding for the client).

Teredo uses what has become a relatively conventional approach to NAT traversal, using a simplified version of the STUN active probing approach to determine the type of NAT, and uses concepts of "clients", "servers" and "relays".

- A *Teredo client* is a dual stack host that is located in the IPv4 world, assumed to be located behind a NAT.

- A *Teredo server* is an address and reachability broker that is located in the public IPv4 Internet.

- A *Teredo relay* is a Teredo tunnel endpoint that connects Teredo clients to the IPv6 network.

The tunnelling protocol used by Teredo is not the simple IPv6-in-IPv4 protocol 41 used by 6to4. NATs are sensitive to the transport protocol and generally pass only TCP and UDP transport protocols. In Teredo's case the tunnelling is UDP, so all IPv6 Teredo packets are composed of an IPv4 packet header, a UDP transport header, followed by the IPv6 packet as the UDP payload. Teredo uses a combination of ICMPv6 message exchanges to set up a connection and tunnelled packets encapsulated using an outer IPv4 header and a UDP header, and contain the IPv6 packet as a UDP payload.

It should be noted that this reliance on ICMP6 to complete an initial protocol exchange and confirm that the appropriate NAT bindings have been set up is not a conventional feature of IPv4 or even IPv6, and IPv6 firewalls that routinely discard ICMP messages will disrupt communications with Teredo clients.

The exact nature of the packet exchange in setting up a Teredo connection depends on the nature of the NAT device that sits in front of the Teredo client.

If the client is behind a *full cone* NAT (which allows any external address to use a NAT binding, once the binding is established) then a single ICMPv6 echo request / reply exchange with the remote IPv6 host allows the client to gain knowledge of the remote IPv6 device's chosen Teredo relay and use it.

If the client is behind a *restricted* NAT (which only allows external devices to use a NAT binding after the client has sent a packet to external device) the remote IPv6 device's ICMPv6 echo reply is passed to the remote Teredo relay, who then passes a Teredo bubble packet back to the client's Teredo server and from there to the client. The client now knows the address of the Teredo relay, and will send it a Teredo bubble packet, thereby allowing the Teredo relay to be able to release the ICMPv6 packet back to the client, allowing the client to use the Teredo relay.

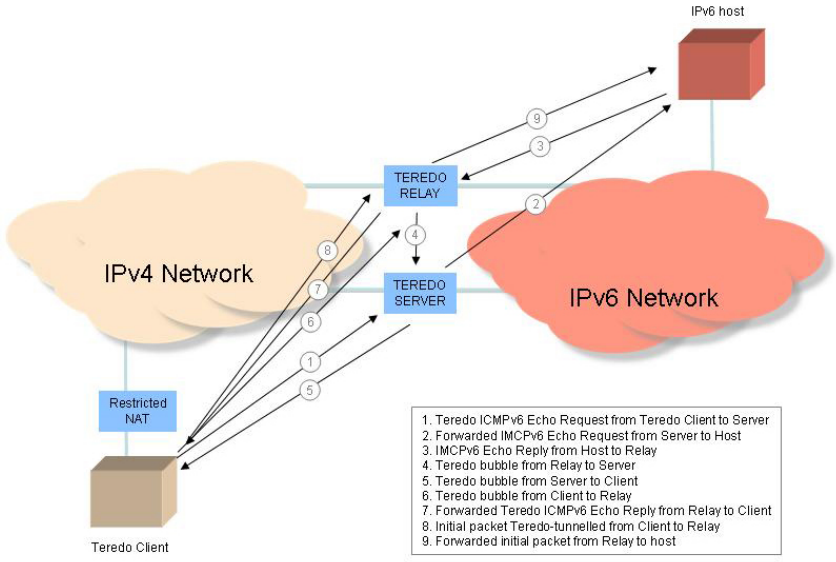


*Figure 1 – Teredo setup for a restricted NAT*

Teredo represents a different set of design trade-offs as compared to 6to4. In its desire to be useful in an environment that includes NATs in the IPv4 network path, Teredo has been positioned a per-host connectivity approach, as compared to 6to4's approach which can support both individual hosts and end networks using 6to4.

How well does Teredo perform?

# Measuring Teredo

In order to understand the performance of Teredo, an experiment has been set up that attaches a JavaScript object to a web page. The JavaScript, when executed by the client, directs the client to perform a number of additional fetch operations. In all cases the object being fetched is the same simple 1x1 image, and the fetches are allowed to occur in parallel. What we'll concentrate on here is the IPv6 aspect of these tests, and Teredo in particular.

## Counting Teredo

How much Teredo is being used? When we look at the clients who prefer to use IPv6 in a Dual Stack situation, Teredo is quite uncommon.

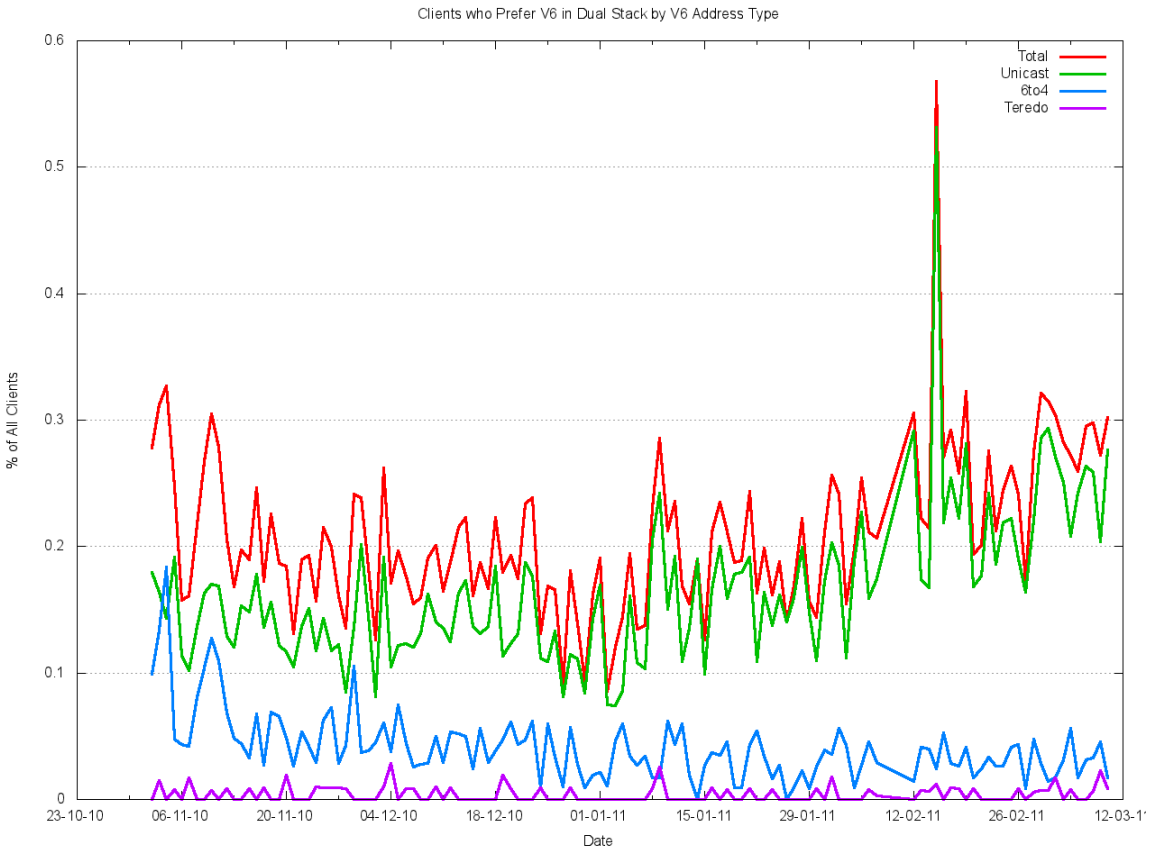


*Figure 2 – IPv6 Address Type of Clients who prefer IPv6 in Dual Stack*

Around one in ten thousand, or some 0.01% of clients, will use Teredo to access a dual stack object with IPv6.

One possible explanation of this low level of use of Teredo is that there are very few end systems that are configured with an active IPv6 Teredo interface. But this explanation does not seem to be consistent with the observation that the two recent software releases of the Windows OS, Vista and 7, both have IPv6 configured to be "on" by default, and when the end system is on an IPv4-only ISP service and behind a NAT, then this IPv6 "on by default" capability is provided by Teredo. Perhaps the explanation is that Teredo takes a lower preference than IPv4, and when presented with a dual stack object, then Windows will prefer to use IPv4.

So what happens with the IPv6-only test, where there is no IPv4 option for the client system to use?

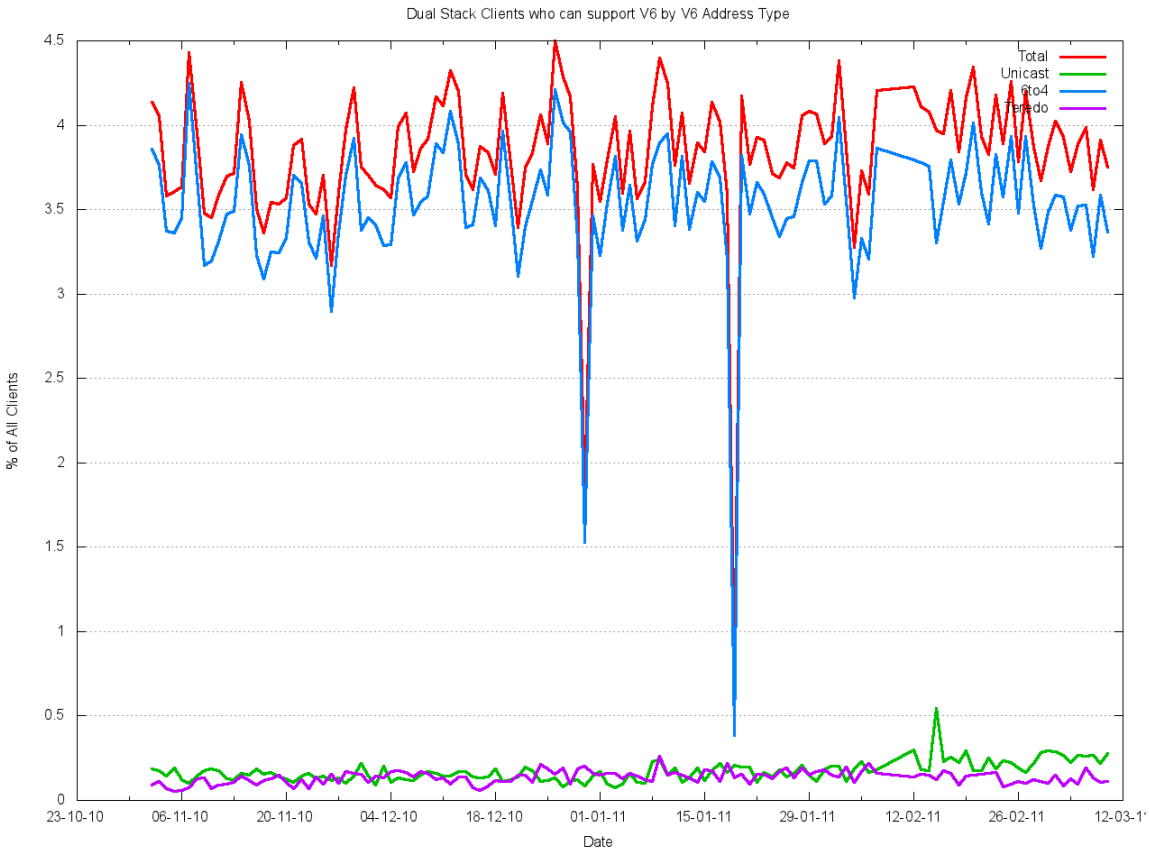


*Figure 3 – IPv6 Address Type of Clients who prefer IPv4 in Dual Stack*

This is again surprisingly low for Teredo. The relative number of clients using Teredo is only slightly higher for the IPv6-only object, with some two in every thousand, or 0.2% of all clients using Teredo to retrieve the IPv6 only object, while most clients who can support IPv6 but prefer to use IPv4 in a dual stack scenario use 6to4.

So this still does not account for the number of Windows Vista and Windows 7 clients that have been deployed with Teredo turned "on".

Looking a little further into this points to the way Windows uses Teredo. As described in a Microsoft technical note (http://technet.microsoft.com/en-us/library/bb727035.aspx), a Windows Vista or Windows 7 client will not query the DNS for an IPv6 address (query a DNS name for a AAAA record) if the only local IPv6 interfaces are link-local and Teredo interfaces. In other words, while Teredo may be enabled on a large number of end systems that are connected to the Internet and located behind NATs, such systems will not invoke Teredo to access an IPv6-only URL in the normal course of events because they will not query the DNS for an IPv6 address to use.

There is an interesting side note here about the interaction of the DNS with dual stack operation on Windows systems.

The same Microsoft technical note points out that "If the host has at least one IPv6 address assigned that is not a link-local or Teredo address, the DNS Client service sends a DNS query for A records and then a separate DNS query to the same DNS server for AAAA records. If an A record query times out or has an error (other than name not found), the corresponding AAAA record query is not sent."

In other words, even when the client has a unicast IPv6 interface, its use of IPv6 transport requires that it receives a response to a DNS query for the corresponding A record.

This introduces the possibility that errors in the DNS relating to the (mis)handling of A record queries in a given domain may cause an IPv6-only resource in that domain to be unreachable for Windows clients.

The note also implies that the DNS queries are staggered such that the AAAA query is not even sent out until the client has received a response to it's A query, or timed out, imposing a time penalty on the performance of dual-stack enabled Windows clients.

When we look at the types of connections to an IPv6 only URL, it should come as no surprise to see that very few connections are initiated from Teredo clients.

We can expose the amount of "hidden" Teredo capability by using a special form of URL that, in effect, bypasses the DNS? A way of doing this is to use a literal IPv6 address in the URL, as distinct from a DNS name. The experiment setup uses a fourth form of text, namely the IPv6 literal: http://[2401:2000:6660::f003]/1x1.png

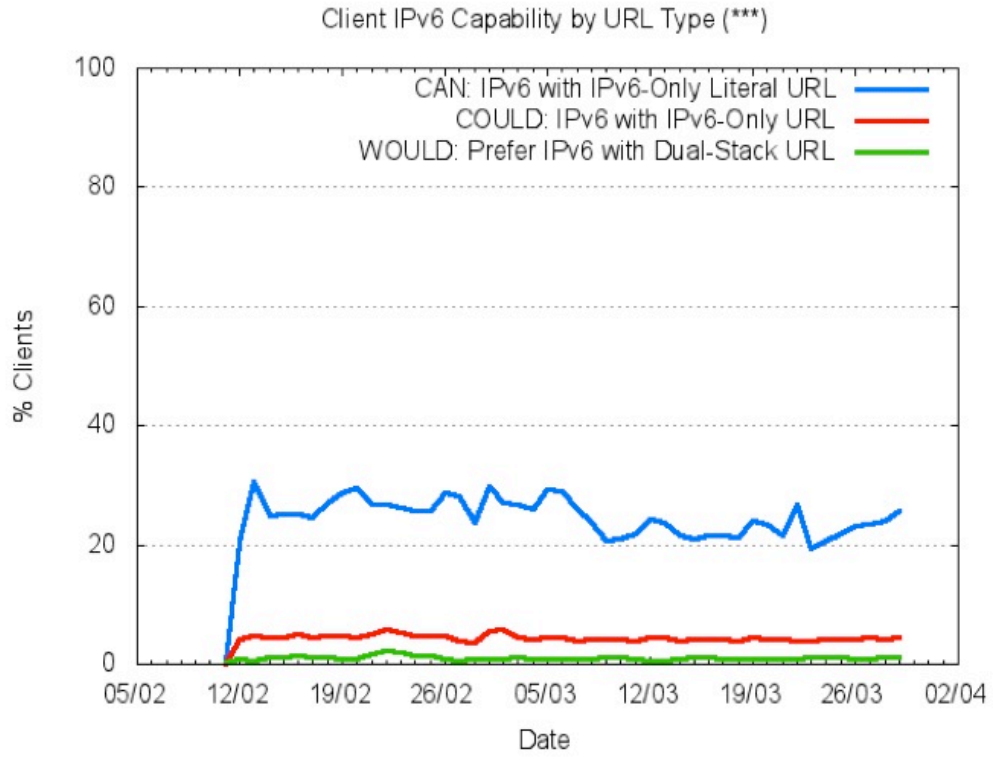


*Figure 4 – IPv6 Capability by URL Type*

What Figure 4 shows is that some 20% to 30% of the client population is capable of retrieving an object using this IPv6-literal form of URL. Taking into account the 0.2% of clients that use unicast IPv6 and some 4% of clients that use 6to4 auto-tunnelling, this shows that some 16% to 26% of clients have latent capability to complete a connection using IPv6 via Teredo auto-tunnelling.

## Performing Teredo

Teredo performance can be quite poor. The test setup was deliberately set up without a local Teredo relay, so that it would be like any other dual stack server, and use an anycast Teredo relay.

The time to perform the fetch of the object was taken, and the time to perform the fetch in IPv4 was taken as the base level for each client. While the performance of 6to4 was consistently some 1.2 seconds longer on average than IPv4, the Teredo performance was seen to be highly variable, with average times up to 3 seconds longer than IPv4.


*Figure 5 – IPv6 Average Retrieval Times relative to IPv4*

It is possible to break down this additional delay into two components: the time to set up the tunnel, and the round trip time.

In 6to4 the data path is asymmetric. The client uses a "local" 6to4 relay to get the IPv4-encapsulated packet into IPv6 network, while the server uses its "local" 2002::/16 6to4 relay to pass the IPv6 packet back into its IPv4-encapsulated format.

The introduction of a NAT in the path makes this a little more challenging, and Teredo attempts to create a symmetric path by having the client lock into the server's choice of Teredo relay. For "full cone" NATs where, once a UDP NAT binding is established through the NAT, any external device can direct inward packets to the NAT binding, the initial setup is simple as a single ICMPv6 exchange. For "restricted" NATs, where inbound packets from an external IPv4 address may only be accepted once an outbound packet has been passed through the NAT directed to that same IPv4 address, then the initial ICMPv6 exchange involves

an additional intermediate packet handover between the client, the client's Teredo Server and the server's Teredo Relay before the ICMPv6 exchange completes.

The result of this packet exchange is that Teredo requires a minimum of one round trip time to set up the Teredo tunnel prior to the first application-level packet being sent from the client to the server.

Because browsers operate using parallel sessions, then by undertaking a packet capture of the client's interactions at the web server, and performing a comparison of the opening packets of the web object fetches in each of the three cases, it is possible to gain an understanding of the amount of time Teredo clients spend while waiting for the Teredo tunnel to be set up. The following figure shows the distribution in time between the opening SYN packet of the IPv4 TCP SYN packet and the opening SYN packet of the Teredo connection.

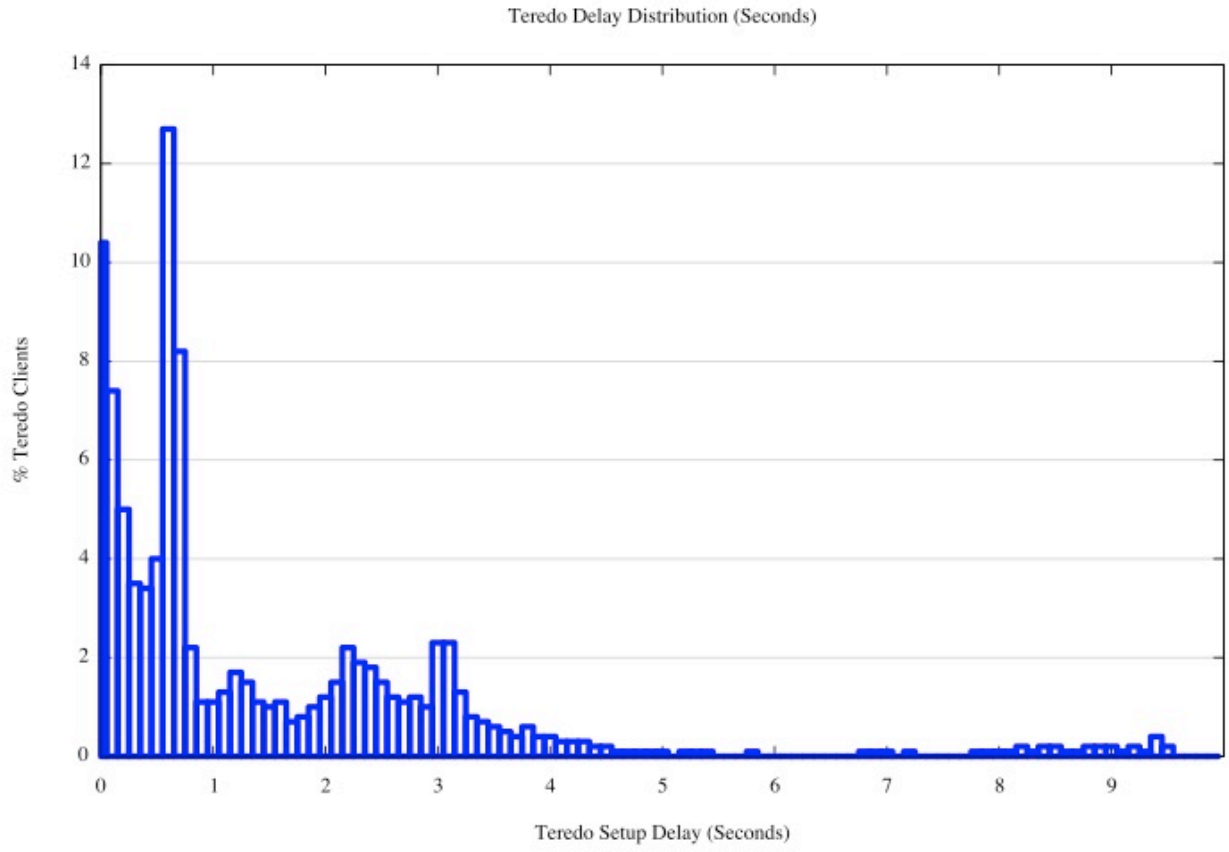


*Figure 6 – Teredo Tunnel Setup Time distribution*

As can be seen from this data set, there is a large peak at 0.7 seconds, and smaller peaks at 2.2 seconds and 3 seconds, and a trailing edge of in excess of 9 seconds to set up the Teredo tunnel.

The second observation concerns the difference in the round trip time between IPv4 and Teredo. The opening 3 way handshake of TCP happens without any form of application-induced delay. By timing the interval between the arrival of the client's opening SYN packet and the following ACK packet, the RTT to the client can be measured. In this experiment we've compared the RTT of the IPv4 connection against the RTT of the Teredo connection from the same client, as measured in the TCP 3-way handshake. The distribution of the difference in RTT is shown in the following figure.

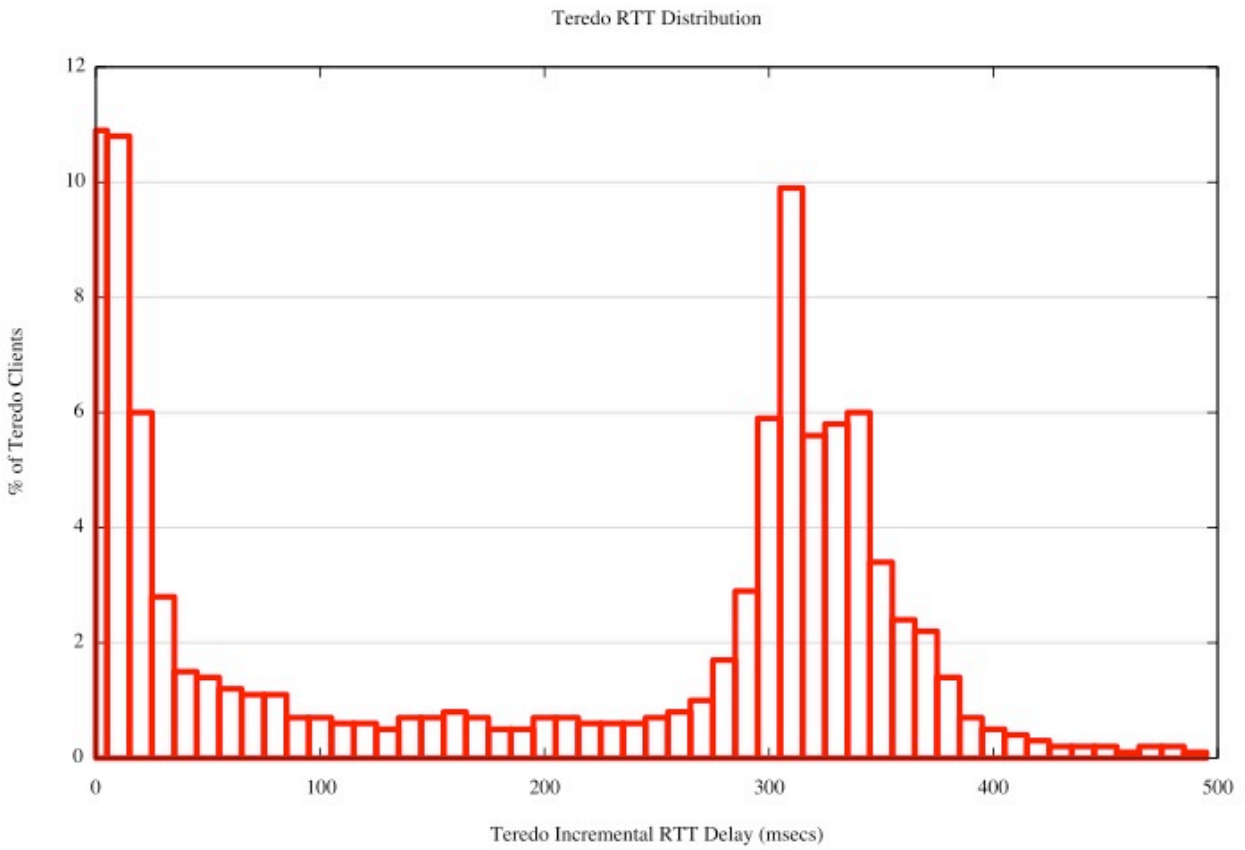


*Figure 7 – Teredo incremental RTT distribution*

For some 20% of the Teredo clients the RTT is directly comparable between Teredo and IPv4, but for 30% of clients, the Teredo connection has an RTT which is around 300ms longer than the associated IPv4 connection.

Clearly, the experiment is exposing some significant performance issues here that lead to the experimental observation that a simple object fetch using Teredo is, on average, some 1 to 3 seconds longer than the same transaction over IPv4.

## Failing Teredo

There is a related question about reliability of Teredo. How well does this Teredo connection protocol work in practice? What is the Teredo connection failure rate? How many Teredo connections do not get to the point of completing the initial TCP 3-way handshake and submitting the HTTP GET request?

In the case of Teredo the initial packet transaction seen at the server is an ICMPv6 Echo Request, to which the server will respond with an ICMP Echo Reply. The next packet seen at the server is the incoming TCP SYN packet, assuming that the Teredo setup is still proceeding, which, in turn, will generate a SYN+ACK response by the server. If the server subsequently then sees an incoming ACK, then the connection is complete.

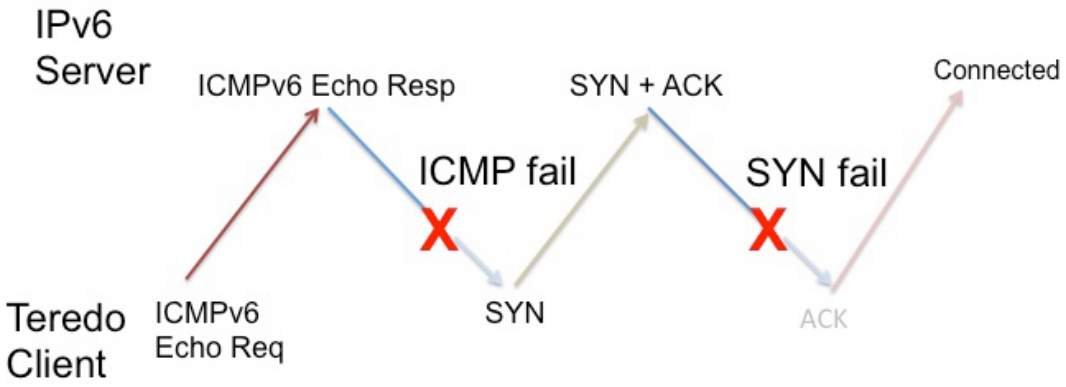


*Figure 8 – Teredo setup packet exchange*

The failure rate for Teredo is certainly high. Some 37% of Teredo connections where the initial ICMPv6 packet is seen at the server do not get to completion. Of these, the bulk of the connections, some 28%, fail at the initial ICMPv6 exchange, where no subsequent SYN packet in seen in response to the ICMPv6 echo response. The remaining 9% of failed connections fail at the TCP SYN handshake where there is no ACK packet seen to complete the initial SYN handshake.

This 37% figure for is a lower bound on the failure rate, as it does not include any consideration of the failure of the initial Teredo exchange with a Teredo server that attempts to match the client with a Teredo Server and establish the client's NAT behaviour type, nor does in include consideration of any failure of the client's ICMPv6 ICMP Echo Request to reach the server.

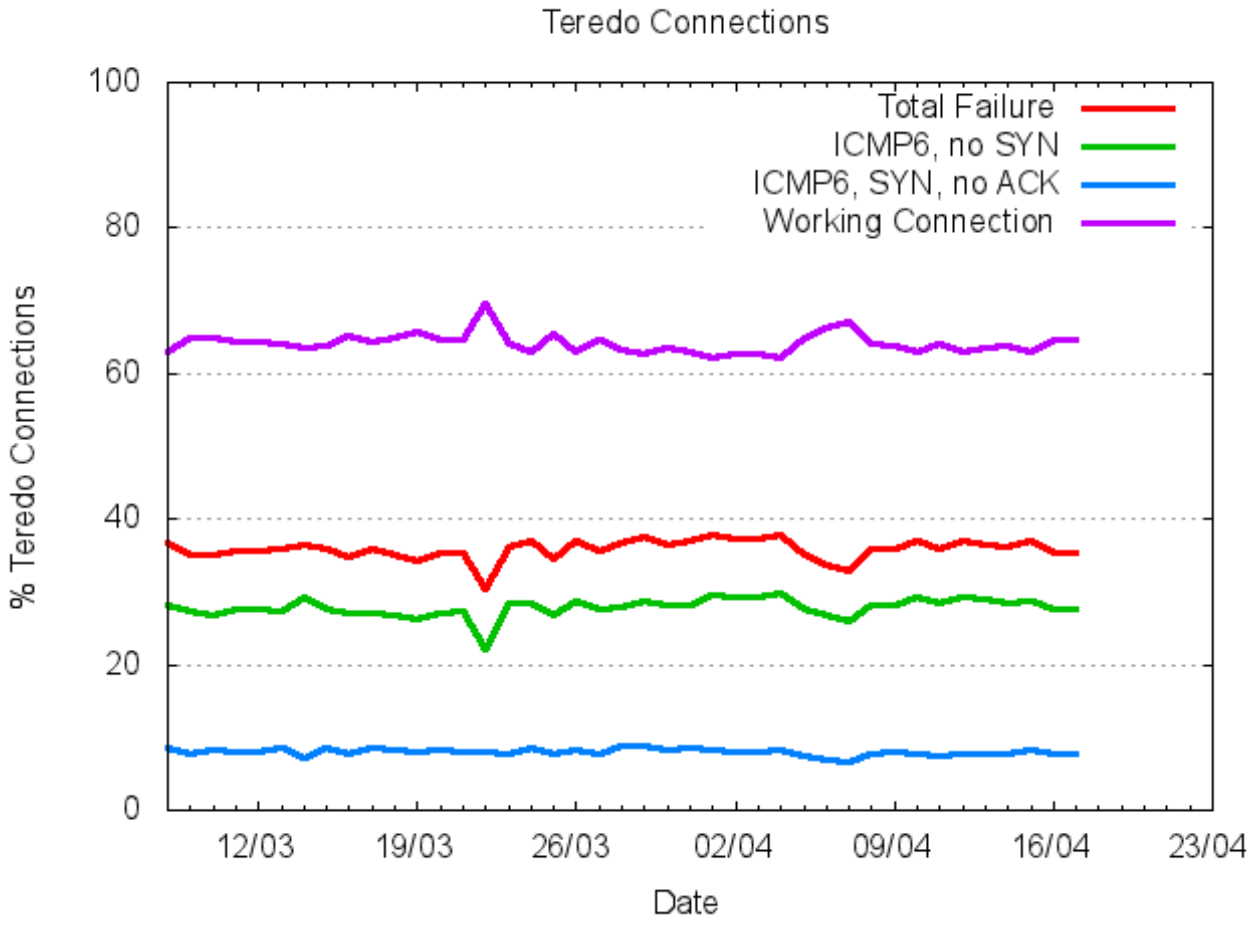


*Figure 9 – Teredo Failure Rates*

It appears that in comparison to IPv4, IPV6 over Teredo is extremely slow and highly prone to failure.

However, most users do not experience these problems with Teredo. While Teredo is enabled by default in Windows Vista and Windows 7, it is not normally invoked by applications.

This leads to the observation that while Teredo clients are widely deployed, and the population of Teredo end clients numbers in the tens or even hundreds of millions of units, tracking the deployment of Windows Vista and Windows 7 end systems, it is to all practical purposes an unused mechanism. While the performance of Teredo is crippling and unusable, there is little benefit in trying to fix this given that that this DNS behaviour implies that it really is largely an unused protocol.

So is this really the case? How widely used is Teredo?

## Seeking Teredo

In a related experiment in looking at "dark" traffic in IPv6, it was surprising to see the profile of connection attempts being made to unadvertised IPv6 addresses. Here's a quick glimpse of one hundredth of one second of traffic that has been captured as part of that study:

```
00.080499 2001:0:4137:9e76:837:232d:xxx  > xxx:563a  ICMP6 echo req
00.081248 2002:dba7:xxx::dba7:xxx.52791  > xxx:37ca.27653: TCP [S]
00.081374 2001:0:4137:9e76:8e7:3d9f:xxx  > xxx:ff79: ICMP6 echo req
00.081744 2001:0:4137:9e76:346c:383:xxx  > xxx:70be: ICMP6 echo req
00.082123 2001:0:4137:9e76:3832:35ce:xxx > xxx:6f5f: ICMP6 echo req
00.082245 2001:0:4137:9e76:20e6:1dcb:xxx > xxx:84c1: ICMP6 echo req
00.083247 2001:0:4137:9e76:304a:fbff:xxx > xxx:ab99: ICMP6 echo req
00.083251 2002:dc96:xxx::dc96:xxx.51138  > xxx.34a1.36959: TCP [S]
00.083372 2001:0:4137:9e76:1c32:d51:xxx  > xxx:a609: ICMP6 echo req
00.084496 2001:0:4137:9e76:201a:37d4:xxx > xxx:5908: ICMP6 echo req
00.084500 2001:0:4137:9e76:2c3a:377e:xxx > xxx:5277: ICMP6 echo req
00.084997 2001:0:4137:9e76:87a:fbff:xxx  > xxx:856c: ICMP6 echo req
00.085371 2001:0:4137:9e76:2454:1bc2:xxx > xxx:cdb1: ICMP6 echo req
00.085496 2001:0:5ef5:79fd:20c2:3761:xxx > xxx:d562: ICMP6 echo req
00.085995 2001:0:4137:9e76:18ec:1a66:xxx > xxx:fbcf: ICMP6 echo req
00.087120 2001:0:5ef5:79fd:2cd8:228:xxx  > xxx:2292: ICMP6 echo req
00.087495 2001:0:4137:9e76:b1:266e:xxx   > xxx:5038: ICMP6 echo req
00.088120 2001:0:4137:9e76:2c65:18f:xxx  > xxx:cc80: ICMP6 echo req
00.088994 2001:0:4137:9e76:3cac:3497:xxx > xxx:5aa1: ICMP6 echo req
00.089868 2001:0:4137:9e76:14d3:5ca:xxx  > xxx:ceee: ICMP6 echo req
```

This is a typical extract of the traffic seen in this study, where 18 of the twenty packets captured in this one hundredth of a second were Teredo ICMP connection attempts, and the other two were 6to4 connection attempts.

Clearly, there is a significant set of IPv6 traffic that does not use the local client's DNS library as the rendezvous mechanism. Furthermore, it appears from an examination of the TCP port usage in this "dark" traffic that most of this traffic is attributable to various form of peer-to-peer networking, where IPv6 is being used deliberately by the peer-to-peer application.

Given the current preference of most client operating systems to prefer to use IPv4 over any form of locally auto-tunnelled IPv6, then this apparent peer-to-peer use of IPv6 may well be the major application of IPv6 today, and if this is the case then it may also be the case that Teredo is actually one of the major forms of IPv6 traffic in today's Internet!

Why does Teredo perform so badly? Or perhaps we should rephrase the question: can Teredo performance be improved?

## Improving Teredo?

One possible factor in the incredibly poor performance of Teredo as observed in this experiment was the decision to configure the dual stack server in the same way as any other dual stack server. I configured the server with an IPv6 unicast service, and a local 6to4 relay, but in the first instance I did not add a Teredo Relay (Miredo) to the server configuration. Instead, I was relying on some other party to perform this translation from IPv6 to Teredo on my behalf.

This reliance on some unknown third party benefactor is not unusual in the Internet, as examined by Jonathan Zittrain in a TED presentation: "The WEB as Random Acts of Kindness" (http://www.ted.com/talks/jonathan_zittrain_the_web_is_a_random_act_of_kindness.html).

Where is my Teredo relay?

```
$ traceroute6 2001:0::1 traceroute6 to 2001:0::1 (2001::1) from
  2401:2000:6660::f001, 64 hops max, 12 byte packets
1 2401:2000:6660::254  0.989 ms  0.872 ms  0.831 ms
2 as4826.ipv6.brisbane.pipenetworks.com  1.300 ms  1.339 ms  1.344 ms
3 ge-0-1-4.cor02.syd03.nsw.VOCUS.net.au  13.376 ms  13.260 ms  13.276 ms
4 ten-0-2-0.cor01.syd03.nsw.VOCUS.net.au  13.388 ms  13.366 ms  13.388 ms
5 ge-0-0-0.bdr02.syd03.nsw.VOCUS.net.au  13.492 ms  13.499 ms  13.482 ms
6 2001:de8:6::3:71:1  13.140 ms  13.361 ms  13.258 ms
7 bbr01-p132.nsyd02.occaid.net  17.896 ms  18.293 ms  17.191 ms
8 bbr01-v240.snjs01.occaid.net  201.156 ms  201.126 ms  202.061 ms
9 bbr01-g2-3.elsg02.occaid.net  212.161 ms  212.165 ms  211.755 ms
10 bbr01-p1-2.dlls01.occaid.net  255.956 ms  255.453 ms  256.101 ms
11 dcr01-p4-0.stng01.occaid.net  304.695 ms  304.917 ms  304.531 ms
12 bbr01-p2-1.nwrk01.occaid.net  481.964 ms  311.408 ms  386.187 ms
13 bbr01-p2-0.lndn01.occaid.net  382.204 ms  381.888 ms  382.026 ms
14 neosky-ic-8241-lon.customer.occaid.net  411.731 ms  412.085 ms  445.658 ms
15 tunnel105.tserv17.lon1.ipv6.he.net  353.714 ms  352.686 ms  352.968 ms
16 gige-g4-18.core1.lon1.he.net  352.854 ms  352.690 ms  353.583 ms
17 6to4.lon1.he.net  352.481 ms  352.469 ms  351.669 ms
18 * ^C
```

Oops! The server itself self is located in Brisbane, but the "closest" Teredo relay to this server is located in London, which is over a third of a second away in terms of round trip time! That's impressively bad!

Why did the server in Australia choose a Teredo relay in London as the "closest" server. A clue as to why can be see in hop 7 of the traceroute reproduced above. The transit service used by the local IPv6 service is the Occaid service (http://www.occaid.org). This is a "virtual" IPv6 provider who provides numerous points of presence scattered all over the Internet, interconnected by overlay tunnels in many cases. The default routing situation is that each AS has a single exit policy, so when faced with a number of peers advertising a routing to 2001:0::/32, the AS will pick a single exit which represents what it sees as the "best" path. In this case Occaid is selecting an egress for 2001:0::/32 located in London. For remote clients of

I thought this was an unusual case, so I tested this same prefix from the Australian Academic and Research Network:

```
$ traceroute6 2001:0::1 traceroute6 to 2001:0::1 (2001::1) from
2001:388:1:4007:20e:cff:fe4b:f987, 64 hops max, 12 byte packets
1 2001:388:1:4007:216:9dff:fe7a:8001  0.339 ms  0.233 ms  0.212 ms
2 ge-1-0-6.bb1.a.cbr.aarnet.net.au  0.600 ms  0.560 ms  0.605 ms
3 so-0-1-0.bb1.a.syd.aarnet.net.au  4.609 ms  4.650 ms  4.592 ms
4 tengigabitether2-1.pe1.c.syd.aarnet.net.au  4.827 ms  4.819 ms  4.856 ms
5 6453.syd.equinix.com  5.076 ms  5.051 ms  4.969 ms
6 2405:2000:ffb0::1  74.197 ms  73.910 ms  73.775 ms
7 POS12-1-1.core1.TV2-Tokyo.ipv6.as6453.net  125.301 ms  110.938 ms 112.921 ms
8 if-1-0-0.1649.core2.TV2-Tokyo.ipv6.as6453.net  361.133 ms 105.210 ms  105.382 ms
9 POS2-0-1.mcore3.PDI-PaloAlto.ipv6.as6453.net 425.874 ms  841.421 ms  341.499 ms
10 POS0-0-0.core1.DDV-Denver.ipv6.as6453.net  400.471 ms  364.301 ms 416.856 ms
11 POS1-0-0.core1.CT8-Chicago.ipv6.as6453.net  385.986 ms 386.174 ms  386.101 ms
12 if-12.core2.NTO-NewYork.ipv6.as6453.net  410.854 ms  410.541 ms 410.472 ms
13 if-9.core3.NTO-NewYork.ipv6.as6453.net  448.062 ms  421.904 ms  431.823 ms
14 if-xe.tcore2.AV2-Amsterdam.ipv6.as6453.net  499.548 ms  499.235 ms  500.164 ms
```

```
15 2001:5a0:200:100::39  600.599 ms  519.740 ms  514.790 ms
16 2001:5a0:200:100::1e  479.537 ms  486.377 ms  479.308 ms
17 xmr8k-nikhef1-ipv6.as39556.net  478.431 ms  478.995 ms  479.175 ms
18 teredo-relay.easycolocate.nl  479.191 ms  478.872 ms  479.805 ms
19 *
```

That's even worse! The closest Teredo relay from this Australian network is now located in Amsterdam, at a distance of some 479 ms of round trip time.

What happens if I install a local Teredo relay?

```
$ traceroute6 2001:0::1 traceroute6 to 2001:0::1 (2001::1) from
2401:2000:6660::f001, 64 hops max, 12 byte packets
1  bogong  0.222 ms  0.146 ms  0.139 ms
2 *
```

This improves the RTT performance to some extent, as shown by the cumulative distribution of the RTT incremental times with Teredo.

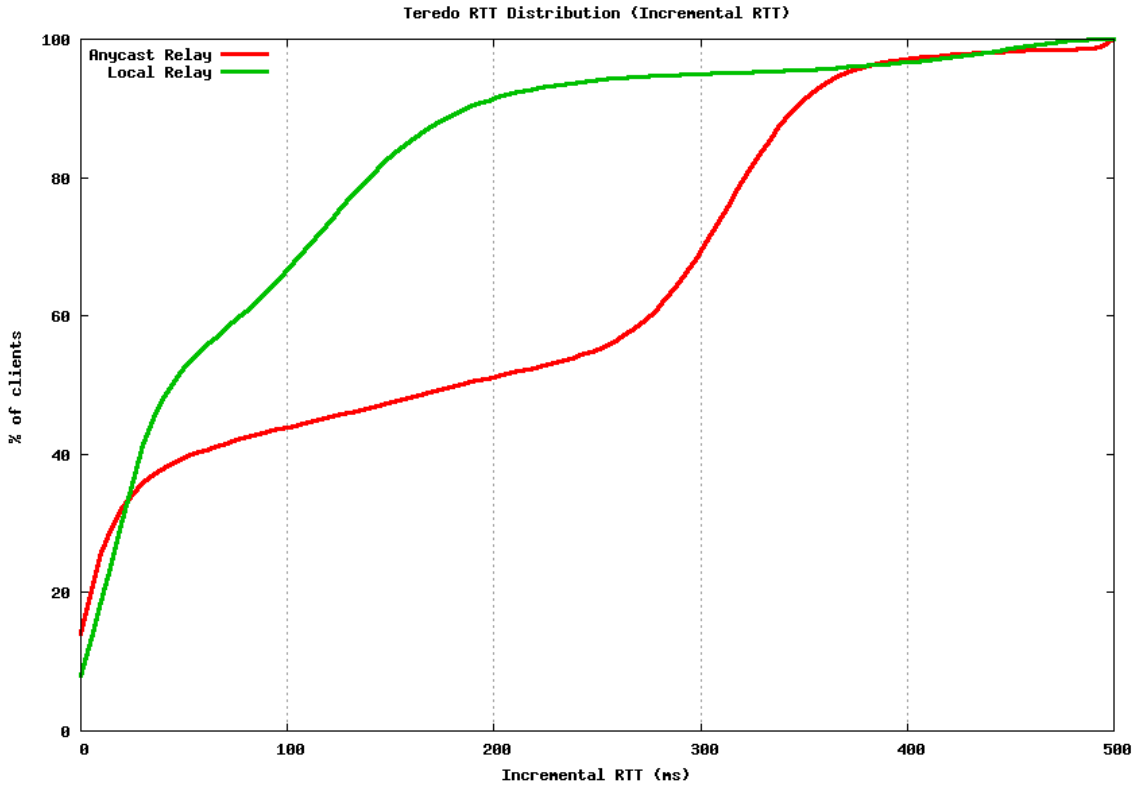


*Figure 10 – Teredo RTT comparison – local vs anycast Teredo Relay*

What is surprising to me is that the local anycast Teredo relay should result in a data path that is aligned to that of IPv4, whereas the experimental data reveals that 50% of clients still have a Teredo RTT that is 50ms or greater than their equivalent IPv4 RTT.

The Teredo setup time is also improved, but only to a minor extent, as shown in the following figure. Despite the significant difference in the RTT between the local Teredo Relay and the anycast Teredo Relay, the local Teredo Relay does not significantly alter the setup time for Teredo, with some 30% of clients still experiencing a Teredo setup time delay of longer than 2 seconds. Whatever is causing this delay in setting up a Teredo connection state, it is not just the RTT from the remote IPv6 host to its Teredo relay. Some other part of the setup operation appears to be at work here.

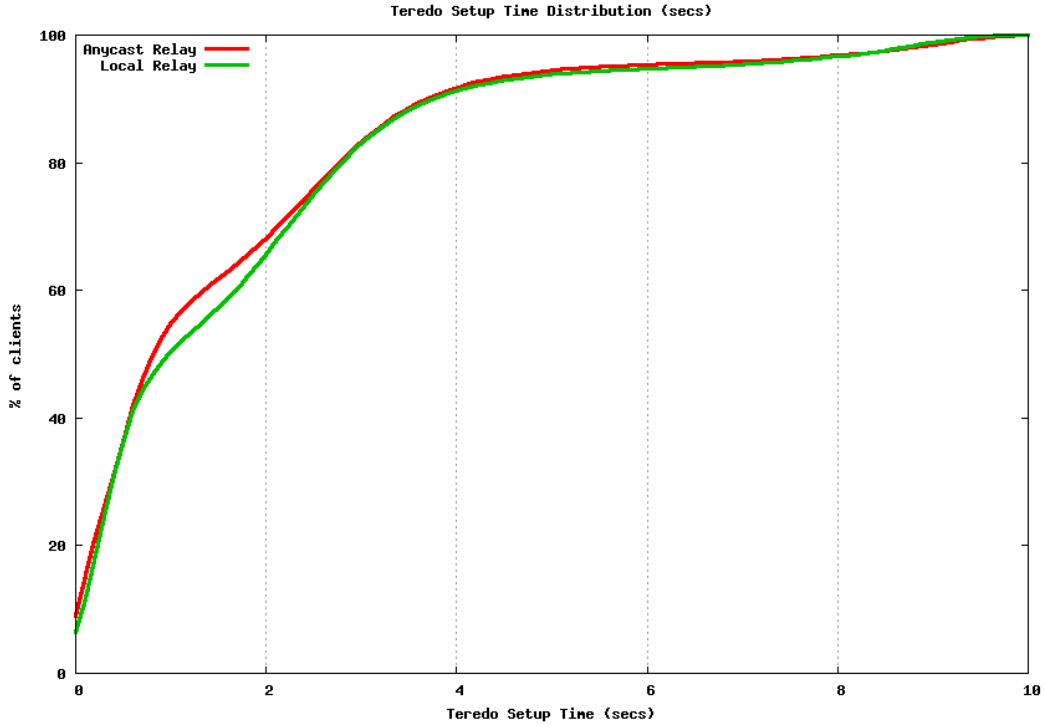


*Figure 11 – Teredo setup time comparison – local vs anycast Teredo Relay*

It also appears that failure of the Teredo connection is not directly related to the RTT time, nor the Teredo setup time. The Teredo connection failure rate is unaltered by the switch from the anycast Teredo relay to the local Teredo relay.

It seems that even with a local Teredo Relay the performance of Teredo is still relatively poor. The failure rate for Teredo-based connection attempts appears to be in excess of one in three, and there is a considerable performance penalty as compared to IPv4 for a large proportion of clients.

In the case of 6to4 the most obvious contribution to the poor performance was the high incidence of local firewalls or filters that blocked incoming packets using protocol 41. In the case of teredo, there is no such "obvious" cause. Teredo packets are conventional UDP packets from the perspective of the NAT device, so local filter or firewall setting would not be expected to interfere with the operation of a Teredo connection.

A more likely explanation lies in the operation of the Teredo packet exchange that attempts to establish the nature of the NAT device through which Teredo is attempting to operate. If this does not manage to correctly establish the behaviour more of the local NAT the consequent handling of the ICMPv6 packet exchange will fail because the Teredo Relay will be incapable of delivering the ICMP echo response to the Teredo client. Twice as many Teredo connections fail at the ICMPv6 exchange as the SYN exchange, pointing to a likely explanation that there are perhaps more forms of deviant NAT behaviour than the Teredo packet exchange is capable of reliably identifying!

The failure of the SYN exchange for some 12% of Teredo clients is not so readily explained. The inference here is that the initial Teredo exchange has succeeded and that Teredo relay has been able to pass an encapsulated packet back to the client with the encapsulated ICMPv6 response. The client is capable of responding to this by successfully sending a SYN packet, via the selected Teredo Relay, so it would appear that there is a bidirectional path through the NAT from the client to the Teredo Relay. Yet for these 12% of clients, the connection does not manage to reach completion in the TCP handshake.

## Teredo in Perspective

We've learned a number of axioms of networking in the decades we're been working with packet switched networks and the Internet in particular. Among them I would offer the following three:

> Tunnelling is really never a good answer.
>
> Stateful devices in the data path are invariably problematic.
>
> NATs are strange!

Teredo exercises all three of these, and it could be said that it is an achievement that it works at all! Expecting it to work reliably in all cases is perhaps just asking too much.

The default behaviour of Windows clients, who will avoid the use of Teredo in any form of communication that is initiated through a DNS name resolution appears to be a reasonable approach. On the other hand the data presented here makes a strong case that Teredo is perhaps best shipped "off" by default.

### Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

### Author

*Geoff Huston* B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He has been closely involved with the development of the Internet for many years, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector. He is author of a number of Internet-related books, and was a member of the Internet Architecture Board from 1999 until 2005, and served on the Board of Trustees of the Internet Society from 1992 until 2001.

*www.potaroo.net*