The ISP Column
A monthly column on things Internet

February 2008

Geoff Huston

IPv6 Transition Tools and Tui

In every ISP's engineering group there invariably lurks a list of those tasks that lie just a little a bit beyond the normal day to day activity of reacting to events as they happen. For many the item "IPv6??!! has been on this "to do" list for some years, if not for the entire lifetime of the ISP itself! This particular task falls into the category of being large enough that there's never normally enough time to put aside to work on it in between all the other day to day tasks, but its not important enough on any day to push the task to the top of the priority stack. So instead it resides on this "to do" list for year after year. This seemingly endless deferral of core engineering for IPv6 appears to be a pretty typical scenario at the moment across the Internet. A look at the IPv6 inter-domain routing table at the moment reveals some 900 unique Autonomous System numbers (AS's), or 900 networks that are publicly routing IPv6, while there are some 27,300 equivalent AS entries in the IPv4 Internet. So there are still a fair number of networks for whom turning on IPv6 remains something to do tomorrow, or possibly the day after.

But is switching on IPv6 support in the network really such a hard task? What does it take to turn on IPv6 in your network? There have been a number of presentations on this topic in various operational for a in recent months, and valuable efforts at collecting practical information that will assist ISPS in assessing what is required to enable IPv6 on their network.

> The ARIN IPv6 Wiki: http://www.getipv6.info/index.php/Main_Page
>
> A collaborative effort coordinated under the auspices of the Internet Society to assemble a collection of useful tools and perspectives on IPv6 deployment issues: http://www.civil-tongue.net/clusterf
>
> A blog by Randy Bush of his service migration experience in setting up a dual stack environment: http://rip.psg.com/~randy/ipv6-westin.html

These list of IPv6 deployment tasks can be quite daunting at the outset. There's the address deployment plan, the routing protocol setup, the issues of setting up IPv6 transit and peering, the determination of IPv6 routing policy, figuring out traffic engineering options, what load balancing may be required for service platforms, enabling existing service platforms to support dual stack access, there's the DNS, mail services, web services, other customer-visible services and facilities, billing, Customer Premises Equipment, firewalls and the security framework, operations and management tools, and application support, to name but a few areas of activity that sit within the broad brush picture of IPv6 deployment. In many ways it might be better not to think of IPv6 as some kind of version upgrade to IP at all, but consider it more as a rollout of completely new protocol infrastructure and start from scratch in terms of the network and service engineering tasks. Whichever way you want to look at it, its not a small task, and considering that there is scant customer demand for IPv6, scant funding for the activity and an incomplete set of services and tools available for IPv6, then its no surprise to see the IPv6 task languish on the list of ISP activities that are continually deferred until some suitable tomorrow.

However, deferring the task because there is too much to do is probably not the best course of action either. The timetable of looming exhaustion of the unallocated IPv4 address pool is one that has inexorable momentum, and while IPv4 NATs might buy some further time in some circumstances, there is still the commonly held view that we shouldn't try to confuse such stopgap measures for a stable and viable platform for the world's communications needs for the next few decades. So if IPv6 really is an inevitable component of our networked future, then is there something an ISP could do today, within the scope of a day or two of effort, that is less ambitious than the full dual stack deployment across the entire network, yet enables some degree of useful and working IPv6 support for its customers? Or, to put it another way, is there a small step that would at least kick start an ISP into the area of IPv6 support?

I suppose I wouldn't pose the question unless I had something in mind, and in this case I'm thinking about an approach that I found quite interesting with Nathan Ward's Tui work. Nathan described his perspective on IPv6 transition at the recent NZNOG 08 workshop in Dunedin, New Zealand (http://2008.nznog.org). This unit is a neatly packaged IPv6 tunnelling relay to allow an IPv4 ISP a quick and potentially painless start in gaining some experience in supporting IPv6 services for its customers.

In this article I'd like to take a look at the various approaches to providing IPv6 services across an IPv4 Internet substrate, and then return to this TUI unit and its capabilities.

IPv6 Tunnelling

The most common scenario of IPv6 deployment today is still one of isolated islands of IPv6 in an ocean of IPv4. The relative isolation of these IPv6 networks means that the only practical way of supporting a connected IPv6 Internet is to treat the IPv4 network as a lower layer connectivity tool for IPv6 and tunnelling IPv6 across the IPv4 "undernet".

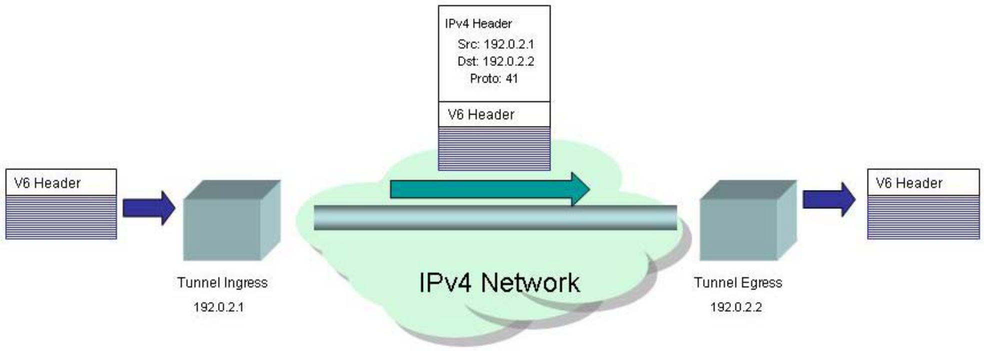
Tunnelling is a very flexible tool, and can be used in a number of ways. A tunnel can be configured as a virtual interface on an individual host system, allowing individual hosts to configure a hop over the local network to make a virtual direct connection to a remote access point. A tunnel may also be configured as a virtual interface on a router, allowing a network to use this virtual connection to directly connect to another network, or act as a remote access concentrator for hosts or end sites.

There are many types of tunnels, ranging from the simple extension of the IP datagram concept of prefixing the original IP packet with an additional IP packet header, using in IP-in-IP (as in the case with the IPv6 tunnelling protocol, where the outer header indicates the presence of a tunnel payload by using IP protocol 41), the Generic Routing Encapsulation (GRE) approach, which places a control field between the outer IP header and the inner packet payload to allow packet sequencing as well as a number of additional control functions, the Layer 2 Tunnelling Protocol (L2TP), the Point-to-Point Tunnelling Protocol (PPTP), IPSEC, and of course various permutations of using TCP or UDP as the lower level transport layer and placing the inner IP packet in the transport protocol payload, such as SSH, TLS or SOCKS.

However, the story is not completely in favour of tunnelling, and in the IPv4 network tunnelling is most typically the domain of VPN applications, and operates on an edge-to-edge basis without the direct involvement of an ISP to broker the VPN. Generically tunnels can be seen as another layer of indirection, and its often the case that adding a layer of indirection can also add a further layer of complexity. Failure within a tunnel may generate an ICMP notification to the tunnel endpoint, but this notification may not be sufficient, or even appropriate to relay back to the original inner packet's source. For example, an ICMP message relating to a failure in the IPv4 network's reachability may make no sense to the sending IPv6 host, for example, while notification of MTU failure may be essential information for the original IPv6 endpoint. Tunnels may be challenging when used with security firewalls, as the identification of the payload and the application port addresses are now embedded within the payload of the outer packet. There are issues with MTU size, fragmentation, and performance characteristics of the tunnel endpoints. Tunnels require various forms of per-packet processing, and while there may be ample processing capability at the edge of the network where packet rates are relatively low, the same cannot be said for the core of the network, where high packet throughput is obtained only via minimal per-packet processing overheads. Fragmentation should be avoided due to the overheads of packet reassembly at the tunnel egress, but if the tunnel traffic uses the DF flag to prevent fragmentation of the tunnel traffic then correct handling of the ICMP messages relating to fragmentation is then necessary. Tunnelling needs some thought in terms of the environment of deployment, particularly in relation to using the tunnel to support routing, where the potential to create damaging circular interdependencies is always a risk. Tunnelling should with a large cautionary label: "Handle with Care!"

With that in mind, what I would like to do here is look at a number of tunnelling approaches that can support tunnelling IPv6 over IPv4, allowing IPv6 "islands" to interconnect using the underlying IPv4 network as the tunnelling substrate.

The simplest form of tunnelling IPv6 packets over an IPv4 network is the IP-in-IP approach. Here an IPv6 packet is simply prefixed by a 20 octet IPv4 packet header. In this IPv4 packet header there is the source address is the IPv4 address of the tunnel ingress, the destination address is the IPv4 address of the tunnel egress and an IP protocol field with the value 41, indicating that the payload is an IPv6 packet. The packet is passed across the IPv4 network from tunnel ingress to egress using conventional IPv4 packet forwarding, and at the egress point the IPv4 IP packet header is removed and the IPv6 packet is routed in an IPv6 network as before. From the IPv6 perspective the transit across the IPv4 network is a single logical hop.



So lets look at a number of approaches to tunnelling IPv6 over an IPv4 network.

Static Tunnels

The simplest tunnel configurations are those where the tunnel endpoints are manually configured. The tunnel configuration takes the IPv6 addresses of the tunnel endpoints, the local and remote IPv4 addresses and the tunnel encapsulation method to be used. The assumptions here are that the endpoints are both dual stack systems and that the endpoints are mutually reachable via the public IPv4 network. This form of tunnel also requires explicit configuration of IPv6 routing. For simple end-site connection tunnels static routing is often used, due to its simplicity of configuration. The end-site points its IPv6 default route to the tunnel point, and the remote end sets up an explicit route for the end site address prefix for the tunnel.

End Site

```
interface Tunnel0
  no ip address
  ipv6 address 2001:DB8:0:1::2/64
  tunnel source 192.0.2.1
  tunnel destination 192.0.2.2
  tunnel mode ipv6ip

IPv6 route ::/0 2001:DB8:0:1::1
```

Tunnel Hub

```
interface Tunnel20
  no ip address
  ipv6 address 2001:DB8:0:1::1/64
  tunnel source 192.0.2.2
  tunnel destination 192.0.2.1
  tunnel mode ipv6ip

IPv6 route 2001:DB8:1::/48 2001:DB8:0:1::2
```

However, this approach of using statically configured tunnels is not always appropriate. If one of the tunnel endpoints lies behind a NAT in the IPv4 network context then the tunnel cannot be configured in this way. Also when firewalls are present on the tunnel path, it is likely that the firewall will block all tunnelled traffic. It is still uncommon to have a firewall that can perform inspection and apply filter rules on the tunnel payload, so allowing the tunnel through in this context often produces the contradictory outcome of having firewall filtering on IPv4 packets, and no firewall filtering whatsoever on IPv6. But the major issue with tunnels are scaling issues. Scaling affects both the amount of per-packet processing at the tunnel endpoints, the number of tunnels that can be configured at the IPv6 tunnel relay points that are acting as access concentrators, the management overhead of the configuration elements, and the amount of traffic that can be handled in this fashion.

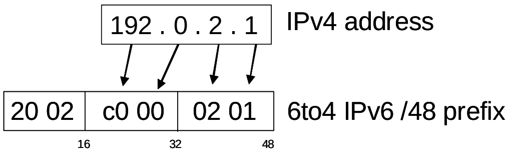
Of course one answer to these scaling issues with static tunnels is to “go native,” and if you are attempting to push gigabits per second of IPv6 traffic through a set of static tunnels on a small tunnel processing platform then the option of migrating your IPv6 connectivity to a combination of a direct IPv6 connection and a high speed router probably makes a lot of sense. But if you have not got to the point of being overwhelmed by the IPv6 traffic levels, is there some alternate approach to tunnel configuration that could reduce some of the administrative workload associated with these manually configured IPv6 tunnels?

6to4

The 6to4 approach is described in RFC3056, RFC 3068 and RFC3964. 6to4 is a framework of automated point-to-multipoint tunnels that addresses the same general scenario as the manually configured tunnel environment, namely a collection of “islands” of IPv6 that are interconnected by the public IPv4 network.

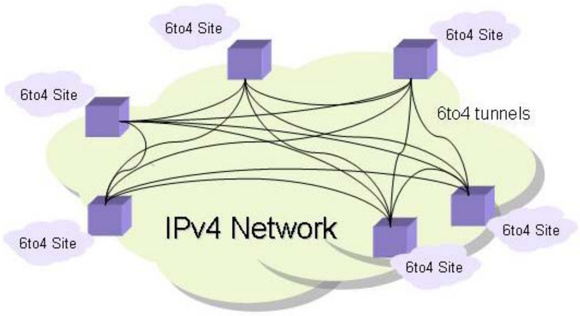
The change with the 6to4 approach is that each site is no longer configured with a single tunnel to some tunnel hub, but is in effect configured with the local end of a point-to-multipoint tunnel, so that each 6to4 site is capable of sending and receiving packets from the entire set of 6to4 end sites using the same local tunnel endpoint. In effect all 6to4 sites form a fully meshed overlay network with automatically managed 6to4 tunnels to every other 6to4 site. Obviously each site does not attempt to create these tunnels, but instead the approach is based on encoding an IPv4 address into an IPv6 site address prefix. The local site’s IPv4 tunnel endpoint address is used to create an IPv6 /48 end site address prefix, so that 6to4 provides both a tunnelling mechanism and a local end site IPv6 address prefix.

The 6to4 mapping is performed by using a 6to4 16 bit address prefix, 2002::/16, then appending a 32 bit IPv4 address to the prefix to form a 6to4 site address. For example the IPv4 address 192.0.2.1 is mapped by 6to4 into the IPv6 address prefix of 2002:c000:201::/48.



The assumption in the 6to4 design is that all traffic to and from the local 6to4 site is passed through the local 6to4 gateway router. Outgoing IPv6 traffic is encapsulated in an IPv4 header, while all incoming 6to4 traffic directed to the gateway is stripped of its IPv4 header and the IPv6 packet routed within the site.

This allows the point-to-multipoint tunnel overlay to form. Any IPv6 traffic from a 6to4 site to any other 6to4 site can be sent directly from 6to4 gateway to 6to4 gateway via a 6to4 tunnel without the assistance of any relays or intermediate IPv6 routing. In essence 6to4 treats the entire IPv4 network as a non-broadcast multi-access network.



What remains is the necessary configuration to allow IPv6 communication between the 6to4 site and the broader IPv6 network. If the local 6to4 gateway is already connected to the IPv6 network then the gateway needs to route 2002::/16 through the 6to4 tunnel interface and route the IPv6 default route via the existing IPv6 connection. If a site only has 6to4 connectivity then it needs to arrange default connectivity via some other 6to4 gateway and use this default route provider’s 6to4 gateway address as their default route and configure this into the local 6to4 gateway. Or a site can elect to take a simpler path and just use the reserved IPv4 6to4 anycast address of 192.88.99.1 as the 6to4 default gateway address (or set up 2002:c058:6301::1 as your IPv6 default route). A local gateway needs only to configure this address as an alias on the 6to4 gateway, and announce a route for 192.88.99.0/24 in the

IPv4 world and 6to4 sites will by default use the closest instance of this default gateway as their default route.

In the 6to4 world the default actions of an isolated 6to4 gateway are to encapsulate outgoing IPv6 packets in a IPv4 packet header. The source address of the IPv4 packet is the IPv4 address of the local gateway. The IPv4 destination address is extracted from bits 17 to 48 of the original IPv6 address if the IPv6 address is drawn from the 6to4 2002::/16 prefix, or just use the address 192.88.99.1 for any other IPv6 destination address.

The actions for incoming 6to4 packets are also simple. The outer IPv4 packet header is removed and the destination IPv6 address is checked against the local IPv6 routing table. If there is a match then the IPv6 packet is forwarded as per the local IPv6 routing state.

But for 6to4 to work you need a public IPv4 address on the “outward” facing interface of the site’s gateway. This is no longer a safe assumption for much of the deployed Internet these days. If you are able to place the 6to4 gateway process on the same unit as a site’s NAT then 6to4 will be able to work. But if you have a NAT between your 6to4 site and the Internet then 6to4 is simply not an option for your site, and you need to look at a tunnelling protocol that includes explicit support for NAT traversal. Which means that you are probably looking at Teredo.

Teredo

“Teredo Navalis” is the Latin name of the shipworm, the creature that bores wormholes into the wooden hulls of ships. In the IPv6 over IPv4 world Teredo is an appropriate description of the mechanism to drill an IPv6 wormhole through the IPv4 network fabric. Teredo is described in RFC 4380.

Like 6to4, Teredo is both an address assignment and tunnelling technology.

Teredo uses 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, possibly behind a NAT. A *Teredo server* is an address and reachability broker that is located in the public IPv4 Internet, and 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 tunnel payload.

The first step in initializing a Teredo connection is the NAT capability detector. Teredo uses a simple categorisation of NAT behaviours into three basic types.

- **Cone** mode NATS operate their UDP bindings such that a binding of an internal address and port number is bound to an external address and port number when the first outbound packet passed through the NAT, and then any external IP address can send a packet to this mapped address and port number and the NAT will map the address and port fields back to the original host.

- **Restricted** mode NATS operate their UDP bindings such that the binding of an internal address and port number and externally facing address and port number is only accessible to the IP destination addresses and ports used by the local host

- **Symmetric** mode NATS operate their UDP bindings such that the binding of an internal address and port number and externally facing address and port number is only accessible to the IP destination address and port used by the local host to create the binding, and different destination addresses and ports will generate new bindings for the local address and port.

The Teredo client sends a Router Solicitation message to a Teredo server. This is a IPv4 UDP packet addressed to UDP port 3544 and sent to the DNS-discovered address of *teredo.IPv6.microsoft.com* (currently 65.55.158.102 and 65.55.158.80), or some other locally configured address of a Teredo server. The Router Solicitation message sets the “Cone flag” to detect whether the NAT operates its UDP bindings in “Cone mode”

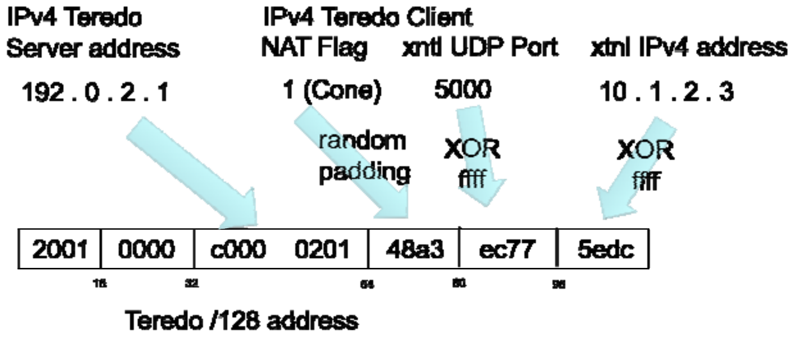
The Teredo server will respond with a Router Advertisement message, and because the Cone flag has been set, the IPv4 source address of this response packet will be an alternative IPv4 address of the Teredo server. If the client receives this message then it can assume that it is sitting behind a Cone NAT.

If the client receives no response then it clears the “Cone flag” and sends a second Router Solicitation message. In this case the Teredo server will respond using the same IP address. If the client receives a response it assumes that it is behind a Restricted or Symmetric NAT.

Finally, the client sends a Router Solicitation message to a secondary Teredo server with the “Cone flag” cleared. The Router Advertisement response contains the external IP address and port address of the NAT binding. If the second Router Advertisement response differs from the first then the client can determine that the NAT operates in Symmetric mode and the Teredo session is discontinued

As long as the NAT is either a Cone or Restricted mode NAT then the Teredo initialization process can continue. For symmetric NATs there are certain unspecified extensions to the Teredo approach that may provide services in some cases, but even so communications between a Teredo client behind a symmetric NAT with a Teredo client behind a restricted or symmetric NAT remains a challenge for as long as Teredo avoids the use of additional relays in the teredo Client-to-Teredo Client case.

The Teredo Router Advertisement packets include an *Origin Indication* field, which contains the NAT-mapped external IP address and UDP port number of the Teredo client. This information is used to construct the client’s Teredo IPv6 address. Like 6to4, Teredo uses its own address prefix, and all Teredo addresses share a common IPv6 /32 address prefix, namely 2002:0000::/32. The next 32 bits are the IPv4 address of the Teredo server. This implies that all Teredo clients of the same Teredo server share a common IPv6 /64 address prefix. The 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). The NAT type field is padded with a random pattern, and the port and IP address fields are XOR’ed with all 1’s to prevent gratuitous NAT translation of IPv4 addresses detected in the IP packet payload.

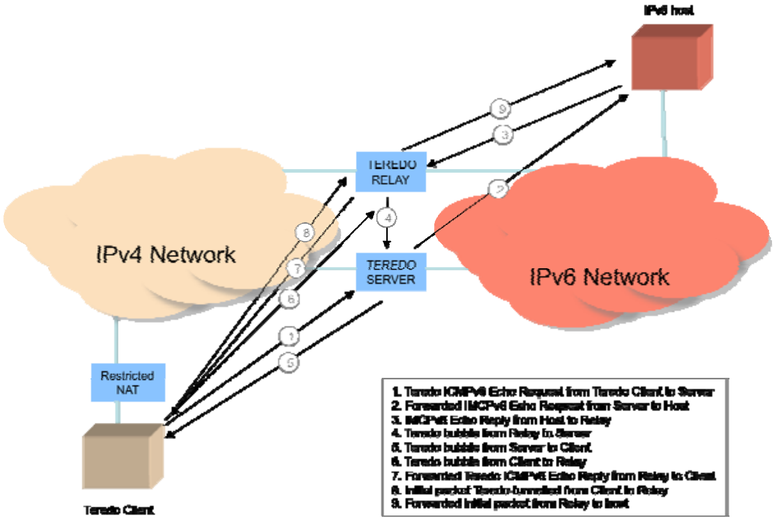


Teredo-tunnelled packets are encapsulated using an outer IPv4 header and a UDP header, and contain the IPv6 packet as a UDP payload. NAT bindings typically have an inactivity timeout, so to maintain the NAT binding there is also a teredo **bubble packet**, which is an outer IPv4 and UDP header set and an inner IPv6 packet header with the **Next Header** field set to *no payload* (59) to indicate that there is no IPv6 payload present.

Teredo packet forwarding involves more steps than 6to4 because of the presence of NATs and the requirement to ensure that there is an appropriate NAT state that allows a Teredo client to communicate.

For an IPv6 host to communicate to a Teredo client, the host will simply pass a packet addressed to the Teredo client into the IPv6 network. All Teredo Relays advertise a route for the IPv6 anycast prefix 2001::/32, so the packet will be passed to the ‘closest’ Teredo Relay. The Relay inspects the NAT Type field in the Teredo client address’s interface identifier field, and if the Cone flag is set then the Teredo Relay will forward the packet directly to the teredo client using Teredo tunnelling. This will correctly transit the NAT because the Cone NAT will allow all Teredo Relays access to the NAT binding for Teredo tunnels. Upon receipt of the Teredo packet the Teredo client will store a local binding of the packet's source IPv6 address to the IPv4 address and UDP port of the Teredo Relay, causing return traffic to the IPv6 host to transit the same Teredo Relay as the incoming packet. If the Teredo Relay detects that the teredo client is behind a Restricted NAT then an additional packet exchange is required before the Relay can access the Teredo NAT binding. In this case the Teredo Relay does not immediately forward the triggering packet, but instead sends a bubble packet to the Teredo Client, using the IPv4 address and UDP port of the Teredo Client's Server. The Teredo Server forwards this bubble packet to the client, but includes an Origin indicator of the IPv4 address and UDP port of the Teredo Relay. The Teredo Client sends this bubble packet back to the Teredo Relay, creating a source-specific NAT binding that the Teredo Relay can now use. The Teredo Relay can now release the queued packets that were addressed to the client. In both these cases the Teredo client responds to the initial IPv6 packet with an ICMPv6 Echo Request and awaits an ICMPv6 Echo Reply before proceeding with the communication.

For a Teredo client to communicate with an IPv6 host there are a number of cases: The host is a Teredo Client behind the same NAT, the host is a Teredo client behind a different NAT, the host is a Teredo client behind a teredo host-specific relay, of the host is not teredo client nor a Teredo host-specific relay, and from a teredo perspective, is therefore an IPv6 only host. In the case of Cone NATted Teredo clients communication with other Teredo clients can happen via direct exchange. In other cases an initial packet exchange between the teredo Client and its Teredo Server is used to establish whether a teredo Relay should be used, and whether additional NAT binding state should be set up. For example, if the remote host is an IPv6 only host, the Teredo client will initially send an ICMPV6 Echo Request packet via its Teredo Server, which will be forwarded onto the IPv6 host. The host's IMCPv6 response will be directed to the Teredo Client, but will arrive at the closest Teredo Relay to the IPv6 host. If the Teredo Client's NAT is a Cone NAT then the Relay will direct the IMCP echo Reply to the Teredo client, who will then set up a local binding of the IPv6 address with the Teredo Relay's IPv4 address and UDP port and complete the initialization of the communication state. If the Teredo Client's NAT is a Restricted NAT then the Teredo Relay will send a bubble packet to the Teredo Server, who will direct the packet to the Client with an Origin indication of the Teredo Relay. The Client will then forward the packet to the Teredo Relay, opening up a NAT binding for the Relay. The Relay will then forward the ICMPV6 Echo Reply to the Client. Receipt of the ICMPv6 Echo Reply will then complete the initialization of the communication state.


IPv6 host
TEREDO RELAY
TEREDO SERVER
IPv4 Network
IPv6 Network
Restricted NAT
Teredo Client
1
2
3
4
5
6
7
8
9
1. Teredo ICMPv6 Echo Request from Teredo Client to Server
2. Forwarded IMCPv6 Echo Request from Server to Host
3. IMCPv6 Echo Reply from Host to Relay
4. Teredo bubble from Relay to Server
5. Teredo bubble from Server to Client
6. Teredo bubble from Client to Relay
7. Forwarded Teredo ICMPv6 Echo Reply from Relay to Client
8. Initial packet Teredo-tunnelled from Client to Relay
9. Forwarded initial packet from Relay to host

It should be noted that this reliance on ICMP 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.

Teredo represent 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 path Teredo is a per-host connectivity approach, as compared to 6to4's approach which can support both individual hosts and end sites within the same technology. Also, Teredo is now a host-centric multi-party rendezvous application, and Teredo clients require the existence of dual stack Teredo servers and relays that exist in both the public IPv4 and IPv6 networks. Teredo is more of a connectivity tool than a service solution. The local IPv6 addresses generated for a Teredo client are based on the NAT IPv4 binding behaviour, and even in the case of the client sending a dribble of bubble packets to the Server to attempt to keep the binding in place the NAT may use an override timer that forces the NAT binding to be refreshed at regular intervals.

On the other hand if you are an isolated IPv6 host behind an IPv4 NAT, and you want to access the IPv6 network then 6to4 is just not for you, and you either have to set up static tunnels across the NAT to make it all work, or if your NAT is a Cone or Restricted type, you can simply turn on Teredo in your dual stack host and you should be able to establish basic connectivity.

Tui

So if you are an ISP and you'd like to offer some support for IPv6 clients, but you are not ready to do the full dual stack deployment exercise across your entire network, then what are your options? If all you want to do at the outset is move IPv6 packets and support the IPv6 connectivity services that are shipping on current host systems, then the initial shopping list can be relatively modest: at a minimum what you need is a dual stack gateway, an IPv6 router, a 6to4 relay, a Teredo server and relay and some idea of others who are willing to be IPv6 peers or offer you IPv6 transit services.

And all of this can be set up on tunnels to start with.

So here's where Nathan Ward's Tui unit looks so well suited to the task. A TUI unit is a dual stack processor that supports IPv6 in the modes of static tunnels, 6to4 and Teredo and supports IPv4 and IPv6 routing. A TUI unit is constructed as a standalone flash memory Soekris unit that is equipped with FreeBSD, IPv6, status tunnels, 6to4 relay services, a full routing suite including BGP and a Teredo server and relay. What you need to provide as a an IPv4 public address, and Ethernet port, an iBGP session and IPv6 transit, either in native mode or via a tunnel.

What the TUI unit will advertise within your network is a 6to4 outbound relay service via a local IPv6 advertisement of a route for 2002::/16, a Teredo relay service via a local IPv6 advertisement of a route for 2001::/32, a 6to4 default routing service via a local IPv4 advertisement of 192.88.99.0/24.

The collection of deployed Tui units also support a full mesh of 6to4 tunnels, and advertise their routes to Tui route servers, who, in turn redistribute all learned Tui routes to all the Tui units.

It seems to me to be a practical and low overhead way to get your ISP started in IPv6 and support some of your more enthusiastic customers with IPv6 services and also gain some practical experience in IPv6 transition services without disrupting your existing IPv4 network service platform. I'm impressed!

For more information on Tui, see http://www.braintrust.co.nz, or Nathan's presentation at NZNOG 08 at http://conference.nznog.org/presentations/20080124_04-6to4-teredo-tui_nathan-ward.pdf

Disclaimer

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

About the Author

GEOFF HUSTON holds a B.Sc. and a M.Sc. from the Australian National University. 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 is currently the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He was a member of the Internet Architecture Board from 1999 until 2005, and served on the Board of the Internet Society from 1992 until 2001.

http://www.potaroo.net