The ISP Column
An occasional column on things Internet

August 2007

Geoff Huston

Transition to IPv6

Last month's ISP Column looked at the exhaustion of the IPv4 unallocated address pool and the state of preparedness in the Internet to grapple with this issue. It concluded with the observations that:

> It appears that this industry has driven itself into a rather messy place right now, and extracting itself from this state in a sane and rational fashion, and without excessive levels of blame shifting as we do it, is indeed quite a challenge!
>
> We've got around two years left of IPv4 address distribution as we knew it, and then that particular world comes to an end.
>
> So what should we be doing now?

It appears that the immediate options we face are to make more intense use of NATs in the IPv4 world and continue to wait to see how high the pressure to transition will get, or to just do it, and finally get serious about IPv6 deployment in the Internet.

There has been a considerable volume of discussion in various IPv6 and address policy forums across the world about how we should respond to this situation in terms of development of address distribution policies. Is it possible to devise address management policies that might both lessen some of the more harmful potential impacts of this forthcoming hiatus in IPv4 address supply, and also provide some impetus to industry to move in the originally intended direction to transition into an IPv6 network? Judging by the volume of contributions to these mailing lists in recent weeks, its clear that there are many ideas on how this might be achieved and no clear indication that any single approach is commonly preferred. This is often the case when the problem itself is both important and challenging.

In this column I'd like to look at the approach to transition to IPv6.

Some Views on Transition

If some form of transition to IPv6 is what is needed here, then why should this be such a hard topic? Indeed some folk do not see this as being an insurmountable issue, and, as an example, there is a recent draft document submitted to the IETF titled "An Internet Transition Plan" [draft-jcurran-v6transitionplan], where the author, John Curran, essentially proposes that we all just agree on some target dates for various phases of this work, and then just do it!

If only transition were that simple!

While the brevity of the document has a certain appeal, I can't help thinking that the underlying issues have as much to do with the interplay of regulatory policies and business interests as they do with technologies.

A presentation at the IEPG meeting at IETF 69 in July on IPv6 Transition and Operational Reality by Randy Bush exposed a different perspective, where the current situation was characterized as having no coherent transition plan, no realistic understanding of costs and no support from the customer base. The presentation made the point that transition was a difficult topic, and part of the issue here was that there were no simple, useful, scalable translation or transition mechanisms that could be readily used. The deployment models of transition do not easily admit models of incremental rollout, and the extent of existing deployment of any form of IPv6 transition environments is extremely limited.

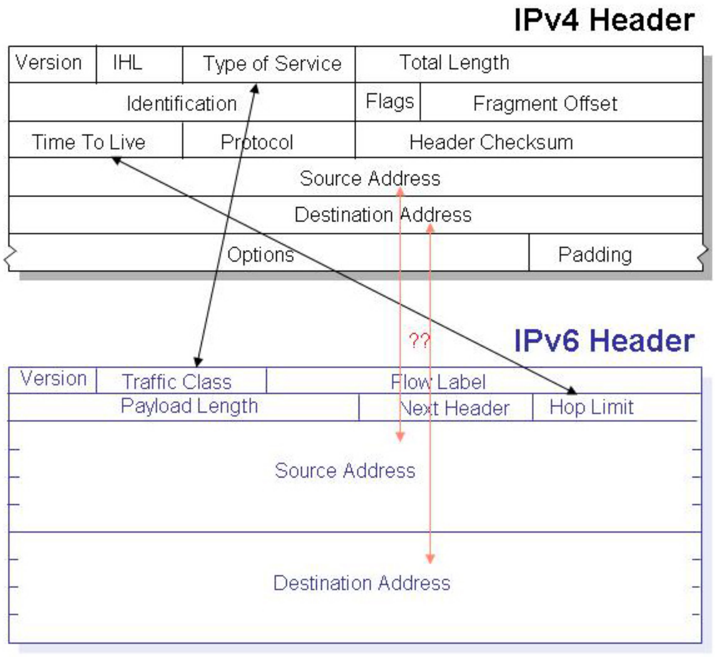
Does transition really have to be that hard a task?

Its not as if the Internet is a stranger to various forms of technology transition over the years. Over the past couple of decades we've raised the basic speed of the underlying circuitry from kilobits per second to gigabits per second, changed the media framing protocols at regular intervals, changed the routing protocols, changed our management tools, completely changed the family of applications, changed the network's architecture and completely changed the customer base. It appears that the technology of the Internet has been in a state of constant flux for the past two decades, and if we are able to learn from experience we should be in a pretty good position to take on this IPv6 transition.

So why is this particular transition looking to be such a tough one?

Transition, Coexistence and (In)compatibility

Even though IPv6 was designed to be a successor protocol to IPv4, its not a backward compatible technology. IPv6 doesn't just push the address size out from 32 to 128 bits in length, but also a uses completely different set of packet header fields and a packet header design, and the result is incompatible. An instance of the IPv6 protocol cannot receive an IPv4 packet header and make any sense of it. Similarly, an IPv6 protocol stack cannot generate a form of IPv6 packet header that would make sense to an IPv4 protocol instance. They are different and incompatible protocols, and while the two protocols can coexist on the same network, they simply cannot interoperate. While most of the IPv4 header fields can be mapped into either the equivalent IPv6 field or some form of extension header in one fashion or another, the problem comes when attempting to map the source and destination address values between the two protocols.

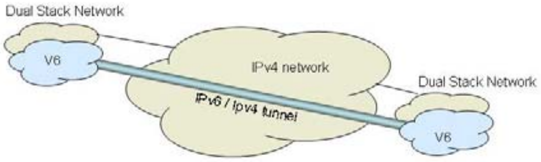


A host using only an IPv4 stack cannot directly communicate with a host using an IPv6 stack. Maybe it could've been different, and perhaps in retrospect we should've looked harder at alternative approaches when working on IPv6 that deliberately eased the burden of transition. But such speculation is not all that helpful given that the agenda before us now is pretty clear: just how can we undertake this transition?

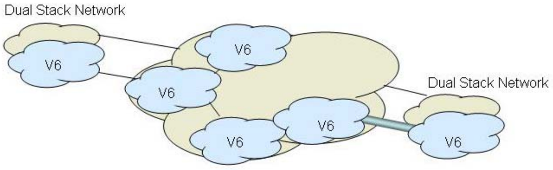
Given that we can't bind the IPv4 and IPv6 worlds together in a seamless fashion at the packet header level, then the starting point of the transition is that the underlying network infrastructure needs to support two distinct protocol families: IPv4 and IPv6.

Consequently, for the past 12 years or so what we've seen is that most of the plans for this transition to an IPv6 network pass through an intermediate stage of progressive dual protocol stack deployment where IPv4 and IPv6 are both deployed. The general idea is that initially new deployments would be both "conventional"

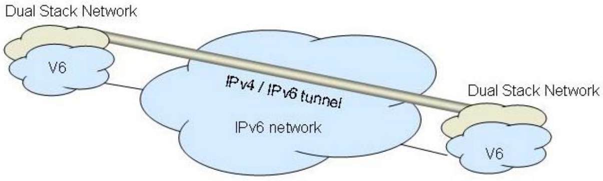
IPv4 and also IPv6, with IPv6 network support being provided possibly via tunnelling across IPv4 infrastructure.



Progressive stages of the transition would see these IPv6 “islands” interconnect directly, and also see the commencement of the older IPv4-only networks converting to a dual stack environment.



It might be the case that at some stage there would be a phase when network infrastructure may be IPv6 only, in which case there would also be some use of tunnelling IPv4 over IPv6 to ensure continuation of ubiquitous connectivity.



Presumably, we’d know that we were in the final stages of this transition when there are new network deployments using only IPv6, at which stage the continued need for IPv4 would’ve finished.

This broad plan raises more questions than it answers, particularly relating to the nature of the drivers that would see existing deployments of IPv4 infrastructure convert to a dual stack environment, and to the drivers that would see the culmination of the transition and termination of the dual stack environment. There is no particular timetable here, and no particular means of coordination of activity, so that knowing when we’ve completely finished with IPv4 may not be an obvious call.

This dual protocol deployment effort is predicated on the constraint that IPv4 hosts simply cannot talk directly with IPv6 hosts. So instead we have the dual stack negotiated behaviour that involves behaviours on the part of the application as well as the host’s protocol stack. The approach is that the application first attempts to expose the protocol capabilities of the other end by asking the DNS for both IPv6 and IPv4 addresses that are associated with the other end’s DNS name. Assuming that the DNS returns both address forms, then the application initially attempts to start up a session with one protocol, and if this fails, then it falls back to the other protocol

Here's an example using the Windows XP operation system and a web browser application. The local network is configured with both IPv4 and IPv6, but to expose the behaviour of dual stack I've turned off IPv6 transit.

The browser has been requested to fetch from http://www.arin.net, a site that is hosted on both IPv4 and IPv6.

```
Time (secs)  Action

 0.00        DNS Query        www.arin.net AAAA
 0.14        DNS Response     2001:500:4:1::80
 0.14        DNS Query        www.arin.net A
 0.16        DNS Response     199.43.0.202, 192.149.252.7
 0.16        V6 TCP SYN       2001:0DB8::1 ⇒ 2001:500:4:1:80
             [timeout]
 3.11        V6 TCP SYN       2001:0DB8::1 ⇒ 2001:500:4:1:80
             [timeout]
 9.15        V6 TCP SYN       2001:0DB8::1 ⇒ 2001:500:4:1:80
             [timeout]
21.22        V4 TCP SYN       192.0.2.1 ⇒ 199.43.0.202
21.50        V4 TCP SYN ACK   199.43.0.202 ⇒ 192.0.2.1
21.50        V4 TCP ACK       192.0.2.1 ⇒ 199.43.0.202
             [TCP session established]
```

The application fetches both the AAAA Resource Record (IPv6 address) and A RR (IPv4 address) from the DNS. The presence of both RRs in the DNS triggers the local application to try a dual stack approach, first attempting to use IPv6 then using IPv4. The application attempts to open a TCP session using IPv6. The first IPv6 TCP SYN packet receives no response, because I've deliberately turned off IPv6 transit in my local network. A second attempt is made after a 3 second timeout. This second packet also receives no response within a 6 second timeout, so a third packet is sent. After a 12 second timeout the application switches over to IPv4, and sends an IPv4 TCP SYN packet. This generates a response from the other end in the form of a IPv4 TCP SYN ACK packet, so the TCP handshake can now complete and the TCP session is opened.

This dual stack process can slow things down when IPv6 connectivity is impaired. As in the example above it takes a total of 21 seconds for the IPv6 connection attempt to be abandoned and for the local host to fall back to an IPv4 connection. More of a problem is the interaction between maximum packet size limitations (MTU). IPv6 over IPv4 tunnelling, and zealous filtering of ICMP "packet too large" error responses. In this case the local dual stack system will successfully perform the TCP handshake, as these are small packets, but as soon as a large packet is sent it generates a silent MTU error. In this case the local application may have cached the information that the remote site is IPv6 addressed and IPv6 reachable and then strongly resists efforts to clear this cache and revert to using IPv4. But these are, on the whole, relatively minor issues, and careful engineering of IPv6 connectivity, IPv6 tunnels and firewalls and filters can avoid the worst of these types of problems.

So, with help from various forms of tunnelling support to bridge over any protocol-specific transport continuity gaps, the basic idea of this transition is that we enter into an extended period where hosts need to use IPv4 to talk to IPv4 hosts and IPv6 to talk to IPv6 hosts. As long as hosts first try the IPv6 handshake, then, so goes the line of reasoning, we should see the overall traffic mix in this dual stack environment tend to move towards IPv6 as the legacy IPv4 infrastructure migrates to IPv6, and the dual stack nature of the deployment means that dual stack hosts can always fall back to IPv4 to speak to IPv4 legacy infrastructure.

As transition plans go, it's a reasonable plan, particularly considering that IPv6 is not a backwards-compatible protocol.

However there is one salient aspect of this dual stack transition that is significant. This approach to transition implies that for the duration of the transitional environment both IPv4 and IPv6 support is required in hosts, across the network, in the routing system, in switches and routers, and in all forms of infrastructure services such as the DNS, firewalls, security, management and rating systems.

This is not a very satisfying situation. This overall process of dual stack transition demands that, ultimately, the IPv4 legacy environment must migrate to support IPv6, while the characteristic of many forms of legacy environments is that they are strongly resistant to change. This process also requires new deployments to be equipped with legacy capabilities for as long as there are legacy IPv4 environments that have communications requirements. It appears that within such a transitional process we would not have moved very far from our current addiction to IPv4 addresses while at the same time attempting to push IPv4 legacy systems into upgrading for IPv6 support.

This form of the dual stack world does not alter the inherent demand for IPv4 addresses, as every communicating host that wants to maximizes its communications realm now needs the ability to communicate in both IPv6 and IPv4. An IPv6-only host can only communicate with other IPv6 hosts, while a dual stacked host can communicate with any other party.

If this is truly so unsatisfying, then this brings us back to the basic proposition in the dual stack IPv6 transition arrangements: is it impossible for an IPv4-only host to initiate, maintain and close a "conversation" with a IPv6-only host and vice-versa? If one allowed various forms of intermediaries, including protocol-translating NATs and various permutations of DNS servers, is this still “impossible”?

Probably not “impossible,” but it would go well beyond the conventional mode of packet protocol header substitution, and would call upon protocol header translation, cross protocol NAT bindings, DNS manipulation and various forms of application level gateways.

What would could such an intermediary approach of protocol header translation look like? It appears that the initial DNS query would need to be locally intercepted and processed using a DNS ALG, and a local cross-protocol binding installed in a protocol-translating NAT. The following is based on the mechanisms described in RFC 2766, that originally described a protocol-translating NAT.

Suppose an IPv6 host (“A”) wants to initiate a session with an IPv4 host (“B”), and neither is dual stack equipped. What would the middleware have to do to support such a scenario?

In this case the middleware would have to:

- Intercept A's DNS IPv6 AAAA address query for B and replace it with an IPv4 A address query
- Take this returned IPv4 address and a local-use IPv6 address (X) install a local NAT binding that binds the IPv6 X address to the IPv4 B address
- Return this IPv6 X address to A as the answer to its original DNS query.

When A sends a packet to X the middleware unit intercepts the packet

- Use a public-side IPv4 address (Y) and bind the IPv6 A address to the IPv4 Y address
- Perform a protocol transform on the IPv6 packet header to generate

an IPv4 packet header with source address of Y and destination address of B
- Forward the IPv4 packet towards B

When B sends a packet to Y the middleware has already created the bindings, so that the middleware can
- Perform a protocol transform on the IPv4 packet header to generate an IPv6 packet header with source address of X and destination address of A
- Forward the IPv6 packet towards A

A similar arrangement would be required to support an IPv4 host attempting to initiate a session with an IPv6 host.

This approach has its dark and ugly side in terms of cost, deployment complexity, service fragility and fractured application transparency. The IETF description of this approach, RFC2766, “Network Address Translation - Protocol Translation (NAT-PT)” was pretty much condemned in a more recent RFC, RFC 4966, which consigned NAT-PT from Proposed Standard to Historic status, with the comment that: “Accordingly, we recommend that: the IETF no longer suggest its usage as a general IPv4-IPv6 transition mechanism in the Internet, and RFC 2766 is moved to Historic status to limit the possibility of it being deployed inappropriately.”

It really looks like dual stack is the only transition mechanism that we are willing to use for the larger internet, implying also that our appetite for more IPv4 addresses continues largely unabated until we come out the other end of this transition to IPv6.

If the trigger for this industry to finally embark on this process of transition to IPv6 is the exhaustion of the current channels of supply of IPv4 addresses, then how is this dual stack transition expected to work ? And even if we attempt to head towards application level gateways and NAT-PT we still have a continuing appetite for IPv4 addresses to fuel the IPv4 side of these mechanisms. So how can any form of IPv6 transition be expected to work if we need a continuing supply of IPv4 addresses to drive the transition if the supply part fails?

The assumption behind this transition was that we'd all get moving with this IPv6 transition while there was still ample IPv4 addresses to fuel the continued growth of this dual stack world, or, minimally, to just enough addresses to support NAT'ted versions of IPv4 domains, and that we'd be done with dual stack long before anything in IPv4 seriously hit the wall of exhaustion. Some years back this probably seemed like a sensible assumption.

But the real question we face now is: "How do we support the growth of the V4 part of this dual stack Internet with a looming exhaustion of the V4 unallocated address pool?"

It appears that if we want a useful Internet to operate now and in the future, its not really a choice between IPv4 NATs and IPv6 at all. To support connectivity across this transition to IPv6 we simply need to continue to use IPv4 both in existing and new network deployments, and that implies we need to make far more intense use of NATs in IPv4. At the same time we need to finally get serious about IPv6 deployment in the Internet. Its hard to see an either/or option in this situation when looking at IPv4 NATS and IPv6. It now seems like we need both.

Paradoxically, the use of NAT in IPv4 is about to become very critically important to IPv6.

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