352: IPv6 Investigation -- Final Report

Current status (2016-09-07): All managed Linux subnets have now had IPv6 enabled.

This project was to conduct an initial investigation into the issues and likely effort involved in bringing in IPv6 support. It was expected that a number of follow-on projects would result to cover specific larger work items which were identified, though "small" items would just be done by this project, particularly if it would aid with evaluating a later phase. The project's home page has links to working documents and a number of useful references.

The project's title was "IPv6 investigation", and the motivation and general description were initially set out as follows: "*IPv4 space is running out, and while there's no suggestion that we'll lose what we have of it, it is increasingly likely that there will be IPv6-only sites out there who want to speak to us or to whom we would like to speak. This project is (a) to identify what we will require to do, (b) do such of it as is small-scale or fundamental to any deployment, and (c) fork off such other projects as are required.*" The project was approved with "strategic" priority, reflecting the increasing importance that IPv6 was expected to acquire in the School's plans going forward.

The project has achieved these (necessarily rather vague) goals. We now have a network infrastructure which actually is capable of supporting IPv6 use on managed machines. We have global IPv6 routing. IPv6 is enabled on all of our managed-Linux subnets. DNS entries are created automatically, where appropriate, or manually as required. A number of technical discussion and maintenance documents have been produced (linked from the project's index page).

Several things were also explicitly deferred:

1. Student labs were not IPv6-enabled, as the switches currently in use for the Forrest Hill decant are too old to provide the necessary infrastructure support.
2. DHCP for IPv6 was forked as a separate project. (Whether this will require a full-blown DHCP implementation or just enough for O-bit use will be for that project to determine.)
3. Rolling out as a full service for self-managed VLANs was also forked as a separate project, with dependencies on DHCPv6 and a switch upgrade (specifically for RA-guard).
4. Managed Windows desktops had a few problems in testing, feedback was provided to IS, and we await their response in due course.

Although this was strictly speaking a "Development" project, there was necessarily some operational impact, and so it was decided to include a standing "IPv6" section in the Infrastructure Unit's reports to the Operational Meetings in addition to the Development Meeting presentations noted below. This provided useful bi-directional feedback.

An initial systems blog post announced the start of the project, with an update shortly before external routing was brought up and tested. A third posting, shortly before IPv6 was enabled on the Forum DICE wire, summarised the position at the end of August. As well as the first presentation, mentioned below, there was a second Development meeting presentation in December 2015 to give a progress update, and a "where next" discussion in April 2016.

The final "managed" subnet (Forum DICE) had IPv6 enabled on it in September 2016. This marked the essential completion of the project, with just a few loose ends to be tidied up afterwards. There is a time-line section below which sets out the main events of the project.

Note that it is not being suggested that IPv4 should be phased out any time soon!

Initial thoughts and expected issues

The first step of the project was to look at the issues which were expected to arise. This was then "condensed" down into a presentation and discussion for the September 2nd 2015 development meeting. Some of these turned into real issues, addressed below, while others were less important than initially expected, or disappeared in the light of experience, or indeed just didn't arise as a result of adopted practice.

Address allocation, DNS

At that development meeting it was decided that there was no need to try to link our IPv4 and IPv6 address allocation in any particular way. (With hindsight this was an eminently sensible decision, and trying to link the addresses really wouldn't have made a lot of sense in practice.) Accordingly, the existing (IPv4) `makeDNS` tool was left totally unchanged and a new tool (`makeDNSv6`) was written to populate the IPv6 zone fragments, which could then be assembled along with the IPv4 fragments in the `dns/` maps' Makefile. The specific Informatics IPv6 address map to rfe is `dns/inf6`. The tools can be found in the `makeDNS` package in the lcfg svn repository. Experience gained during testing suggests that many (most?) machines will automatically generate a MAC-based address, further decoupling them from any possible link with IPv4 addresses but greatly simplifying IPv6 address provisioning. Our existing lcfg-based MAC handling makes this straightforward to deal with, of course.

A separate document discusses IPv6 addresses in general, with some suggestions as to when each type might be most appropriate.

The syntax adopted for `makeDNSv6` was as follows:

- Blank lines and anything after a '#' are ignored.
- Lines are tokenised at whitespace.
- At present there is no continuation between lines. This could be added later if required.
- Any line which begins with %subnet is used to introduce a new subnet, within which all of the following lines (up to the next %subnet) are located.
- Any line which begins %slaac causes a *forward* SLAAC-style address to be generated, based on the lcfg MAC data.
- Any line which begins %slaacFile points to a list of hostnames for which SLAAC-style forward and reverse address-generation should be attempted. These files are generated as below. Any line which begins %slaacReverseFile has addresses generated for the reverse direction only.
- All other lines are expected to begin with a valid IPv6 address fragment ("interface ID"), which can be up to four chunks in length.
- The whole lot is contained within a global prefix which is set as a configuration option to the tool.
- Unless explicitly added, :: is assumed to be on the left-hand end of the current fragment.
- All subsequent tokens on the line are names which should be associated with the first-token address.
- A SLAAC-style *reverse* entry is also generated for the "base" name.
- Our usual conventions regarding aliases and names with hyphens apply.
- Once the whole source file has been read in and parsed successfully, a "forward" zone fragment and a number of "reverse" zones are written out.

Because of the way `bind` loads and merges RR sets from its input zonefiles, names are shared between the IPv4 `dns/inf` and the IPv6 `dns/inf6` maps. This is, of course, what we would want and expect to happen. Secondary nameservers then just AXFR all of the IPv4 and IPv6 content as usual. For example:

```
march.inf.ed.ac.uk.     2401    IN      A       192.168.49.58
march.inf.ed.ac.uk.     2401    IN      A       129.215.33.1
march.inf.ed.ac.uk.     2401    IN      A       129.215.64.1
march.inf.ed.ac.uk.     2401    IN      A       129.215.27.239
march.inf.ed.ac.uk.     2401    IN      A       129.215.160.2
march.inf.ed.ac.uk.     2401    IN      A       129.215.42.1
march.inf.ed.ac.uk.     2401    IN      A       192.168.64.5
march.inf.ed.ac.uk.     2401    IN      AAAA    2001:630:3c1:64::1
march.inf.ed.ac.uk.     2401    IN      AAAA    2001:630:3c1:33::1
[... and other non-address content ...]
```

After a bit of experience it looked as though it would be useful to be able to add IPv6 SLAAC-style addresses for all (IPv4) names on *selected* subnets. This was implemented in two stages:

1. A separate tool for simplicity and clarity (`extractBySubnet`, still part of the same `makeDNS` package) was written which scans a hosts-file-format input file and outputs lists of names under the control of a configuration file containing subnet numbers, netmasks and filenames.
2. The makeDNSv6 tool was then enhanced to add a `%slaacFile` keyword, with filename as parameter, which would direct the tool to read the list and add corresponding SLAAC-style addresses for all those names which had matching lcfg MAC translations.

This allows generation to be configured by subnet, as for "client" wires there was a good case for generating entries for all machines, whereas for "server" wires it was felt that it would be much better for this to be under the explicit control of machines' managers. Decoupling the tools in this way makes each simpler and clearer, and less likely to be buggy, and has the useful side-effect of leaving intermediate state files which can be used to help track down subsequent address-generation issues or as input to other tools.

A further extension to the mechanism (`%slaacReverseFile`) results in only reverse entries being generated. This is somewhat useful in that tools such as `dig -x` can look up the address to find the hostname, but security-aware code will want to find a matching forward mapping before trusting the result.

Our `gai.conf` configuration also required checking over. We had added gai configuration support to the lcfg-dns component back in 2011 in an effort to make it prefer "local" addresses, our previous "sortlist" approach which we had used since before 1999 having been rendered ineffective by `getaddrinfo()`. Prior to IPv6 we configured things so that (IPv6-mapped) IPv4 local and EdLAN addresses would be preferred, in that order, in conjunction with the default table from RFC 3848. We're now in an IPv6-enabled world, and RFC 3848 has been obsoleted by RFC 6724.

Fortunately, after a bit of thought, it became apparent that we wouldn't actually have to make any changes to our existing setup. In particular, we don't need to add any specific new IPv6 support. IPv6 addresses will be preferred, using the default rules (as augmented by our IPv4 additions), and the choice of which IPv6 address to pick in multi-homed cases happens automatically as the longest-matching-prefix code uses the entire /64 prefix including the subnet number. (In the IPv4 case the prefix match stops before the actual mapped address; and in any case we need to take account of non-/24 subnets, which the component code does automatically.) The IPv4 priorities we chose for our local additions are also still fine, as although the priorities overlap with those of RFC 6724 the associated labels and scopes are different.

Edge switches

(These enhancements also apply to the core switches.)

The following enhancements were made to the `pcScan` tool (from our `ProCurve` package):

1. Enhance the `manager-addrs` configuration option so that it knows how (and when) to use the IPv6-capable MIB entries, and to ignore for now any IPv6 addresses it finds which have been set by hand. In practice we don't expect any edge switches to have IPv6 management addresses, at least in the near future. *In due course it should be further enhanced so as to be able to manage the IPv6 addresses completely.*
2. Set Router Advertisement (RA) parameters per VLAN, as controlled by a new RA configuration-file option. These are enabled for switches which are doing IPv6 forwarding, and disabled for all switches which aren't even if they are potentially capable of doing so. As only a few core switches will be doing IPv6 forwarding, RA will generally be set to "off" in practice. For convenience and consistency, each site's RA parameters are collected into a `<site-prefix>/VLANs.RA` rfe map.
3. Enable "RA guard" on all untrusted ports, and disable it on all trusted ports.
4. *MLD-snooping where possible still to come...*

Several of these were made more complicated by the need to cope with varying levels of IPv6 support in our older-model switches!

Core switches

In addition to the above for the edge switches, all of the core switches have been given static IPv6 addresses on appropriate VLANs, following our separate instructions, and have had OSPFv3 enabled, again following those instructions. This was straightforward, though it turned out not to be possible to do authenticated OSPFv3 as the switch firmware doesn't appear to have a way to set any IPsec options (specifically AH or ESP). As we only speak OSPFv3 on a small number of subnets, each of which has only managed machines on it, this may not be too much of an issue in practice. (In fact, it turns out that the current EdLAN core switches aren't licenced for AH or ESP in any case.)

(Note that the IPv4 and IPv6 manager-addr lists are not currently synchronised with each other. In particular, only wire-B IPv6 addresses will be accepted. If this is a problem in practice, either use ssh's "`-4`" flag or specify an IPv4 address explicitly.)

One interesting feature showed up on the first reboot of any of our OSPFv3-enabled switches: the addresses on the "link" interfaces were labelled as "tentative", and so OSPFv3 didn't start. This appears to have been DAD-related, and so our instructions now contain a section to check and turn DAD off if necessary. (This should be the *only* place where we do this on the Informatics network.)

The `pcScan` tool now reports OSPFv3 parameters alongside the OSPF(v2) ones.

Also not initially implemented by the switches were any of the "new" RA extensions, in particular Priority, RDNSS and DNSSL. Fortunately the default priority is "medium", so a workaround was to use BIRD on a Linux router to provide these using low-priority RA. (This was fixed in later firmware versions, but the BIRD solution may turn out to be more flexible in the long run, as it's lcfg-configured. Alternatively the switches might be configurable through SNMP, in which case it would make sense to add the capability to `pcScan`.)

Linux: iptables

The lcfg-iptables component had already had IPv6 support mostly added some time ago. There were only a few additional minor changes required: to pass a list of IPv6 addresses into "generating" rule-scripts to allow them to emit appropriately; and for some SL7-port changes. The base component's ruleset fragments were all checked, and "g." versions of some were created to handle non-IPv6-safe cases. There were only a few of these.

The "g." scripts in dice-iptables, however, took a bit more thought and effort. These are all more tailored to the Informatics or EdLAN environment, and as a result are generally more complicated. As part of this process they were converted from using the lcfg-iptables package's `_iptables_getaddr` helper utility (for shell scripts) or tkined's `Tnm::dns` (for tcl scripts) to using bind-utils's `dig` tool for both shell and tcl, as this was the most straightforward way to allow them to look up AAAA entries. This also removes a package dependency on tkined, though of course adds one on bind-utils in its place. There's nothing special about shell and tcl, of course; all that's required is to write some rules to stdout. C or perl or python or ... would work just as well.

Although the component does allow for chains' IPv4 and IPv6 rules to be different, in all cases the "g." scripts were modified in such a way that they could be used for both IPv4 and IPv6 rulesets. This was done by testing the "wires" list which is passed into these scripts by the component as the first command-line parameter. The component is careful to pass in only addresses of the correct type. The scripts are then able to generate IPv4 or IPv6 versions of the rules, as appropriate, or to omit rules for cases where addresses or restrictions of the appropriate type are not present for one or other version. IPv6 tests have to be sufficiently general to allow link-local as well as global addresses, so only match against the top chunk and the first colon, though as the component only passes in well-formed addresses this is sufficient. IPv4 tests, on the other hand, generally look for the entire dotted-quad, as there isn't an equivalent to IPv6's "elide zeros to '::'" feature.

The lcfg header files have been modified in such a way that it is still possible for different sets of rules to be used for IPv4 and IPv6. In fact, the default rulesets are identical, but some machines then add IPv4-specific rules in a few cases.

One interesting, and unexpected at the time but in hindsight entirely reasonable, effect of configuring IPv6 filtering was as follows. The `dice/options/iptables.h` header defaults to DROP for the INPUT chain policy, while enabling the IPv6 rules (at the time by setting the `IPTABLES_DOING_IPV6` macro) adds all the expected rules, including ACCEPTing traffic from link-local addresses. In particular, this allows through Router Advertisement (RA) packets on those VLANs where we had it enabled at the time. This then resulted in "global dynamic" addresses being created for the interfaces on those wires, and useful IPv6 default routes being installed. Although this didn't appear to cause any problems for the machines involved, it was decided to delay enabling those rules more generally until after global routing was properly up and running.

IPv6 rules are now enabled by default.

Linux: routing

We opted for BIRD, as it appeared that quagga wouldn't do areas for IPv6 OSPFv3. (This does seem unlikely, but...) It also gives us a bit more diversity, and experience with alternative routing solutions.

A BIRD lcfg component has been written. This currently supports the "kernel", "direct", "ospf" and "radv" protocols, to at least a reasonable extent, and does appear to interact with the ProCurve and Cisco switches in the expected way. It has been written in such a way that it should support both IPv4 and IPv6, though only the latter has been tested to date. There is no OSPF authentication yet. The Status() method has been extended to take additional parameters (status, prot[ocol], ospf, interface, route) which give additional information on the functioning of the daemon and its peers and protocols.

There were no particular issues with this. There is a version of the daemon in one of the standard repositories, but we built from the SRPM from the BIRD web site so as to get a newer version. The component was written to generate configuration files in the standard places, to make it simpler to have init or systemd start things, though this has been written to be configurable and for testing purposes the component is starting the daemons itself.

One feature of the HP switches turned out to be that while they did implement the base Router Advertisement RFC, they hadn't implemented any of the "new" options from later RFCs. In particular, they couldn't set the router priority or advertise any nameservers. Fortunately BIRD can do both, so the workaround is to add BIRD routers with a low-priority as required to advertise IPv6 nameservers. (A new firmware revision did eventually add this capability.)

To EdLAN and beyond

Following discussions with IS (Sam) we settled on OSPFv3 as the most straightforward way to exchange IPv6 routing information. Routing was enabled on our "external" subnets (`2001:630:3c1:42::/64` and `2001:630:3c1:160::/64`) on Tuesday 2nd February 2016, and at that point we joined the global IPv6 Internet. This turned out to be entirely straightforward, as on our end it was just a case of adding a few more resources to the lcfg-bird configuration in the relevant header file to set the routing parameters. As with IPv4, all of Informatics is in area 64, and the ABRs are the EdLAN-AT and EdLAN-KB routers.

Initially only a couple of test routers were brought up, but after it became clear that things were running stably all of our standard edge routers were also IPv6-enabled, essentially giving IPv6 the same level of support as for IPv4.

Address-use auditing

It was evident early on that an IPv6 replacement for `arpwatch` would be required, and we picked `addrwatch` as the most useful alternative. It has been packaged up as an RPM, and a component written to manage it. The functionality is slightly different from arpwatch, and in some ways more convenient:

- The daemon just logs ARP and ND packets as they go by rather than trying to preserve a "current mapping" table, which allows for better "historical use" searches.
- The log files can be handled using logrotate in the usual way, rather than having to stop the daemon to run a purge tool over its database files.
- There's control over how long mappings are remembered, so that logs can be rate-limited and duplicates suppressed.

It doesn't, however, email out MAC-mapping changes in the way that arpwatch does, so the plan is to run *both* meantime so that we keep the (IPv4) emails while having the better history available through addrwatch.

Testing and rollout

Initial testing was done on a small number of SL6 machines, located on subnets where enabling features on the core switches would be unlikely to result in anything else trying to speak IPv6. Once the general framework was in place and working, global routing through EdLAN had been established, and most of our standard edge routers were suitably configured, IPv6 was declared open to COs at the 2016-02-10 Operational meeting.

For both SL6 and SL7, the process has generally gone smoothly. A couple of little niggles were identified along the way:

1. rsync likes to use IPv6 in preference to IPv4, but if the destination is using a "hosts allow" list and the source address is of a SLAAC form then there might not be a DNS reverse entry for this. Even if there is, there might also have to be a matching forward entry. Using the "-4" flag to prefer IPv4 works around this.
2. The DICE default was for sshd to listen only on IPv4. Setting

   ```
   !openssh.sshdopt_AddressFamily mSET(any)
   ```

   allows it to listen on IPv6 too. This actually speeds things along a bit, as it avoids the otherwise-inevitable initial attempt to use an IPv6 address, which will then fail and force an IPv4 retry. This has since been made the default on SL7 DICE.

A wiki page was created to allow other testers to add comments on their IPv6 experience.

Tests of the Windows 7 managed desktop prior to enabling IPv6 on wire M (212) produced mixed results. In particular, IPv4 DHCP appeared to be contingent on IPv6 RA having the M-bit set. This wasn't expected, and as a result the decision was made to bring forward migrating all of these machines to their own (IPv6-free) subnet. (RT#77349.) Once that was done, and following discussion at an Operational meeting and a warning email, wire M was IPv6-enabled on August 1st 2016. SLAAC-style IPv6 forward and reverse DNS entries are being automatically generated for all machines with IPv4 addresses on the subnet.

Another blog article was written, notifying the imminent enabling of IPv6 on the Forum DICE wire and giving a status update on (managed) servers and self-managed machines. The actual enabling of IPv6 on the DICE wire happened at lunch-time on Tuesday 6th September 2016. Other than a couple of trivial bugs in `makeDNSv6`, this went off smoothly. Once again, SLAAC-style DNS forward and reverse entries are being automatically generated, bringing us to well over 550 AAAA DNS entries.

As noted above, we need switch upgrades and DHCPv6 before we can open up to self-managed users, and this project has therefore come to a natural end-point here.

Time-line

As the project has been fairly drawn out, in real-time terms, it's useful to display a time-line showing the important events. The project's home page has links to related documents.

0. (March 2015: Sam's UNIX Day presentation)
1. 25th June 2015: project formally started
2. July, August: much reading and thinking...
3. September: all core switches given initial IPv6 addresses, and OSPFv3 enabled within Informatics; some network servers given initial IPv6 addresses; switch configuration tools taught (a bit) about IPv6
4. September: first Development meeting presentation; first systems-blog article
5. October: DNS entries generated; switch configuration tools taught how to do RA and RA-guard (where possible); iptables rules done but not yet in the default set
6. November: lcfg-bird; BIRD on Linux routers
7. December: Development meeting update; lcfg-bird enhancements
8. January 2016: second systems-blog article
9. 2nd February: global routing enabled!
10. 10th February: opened for general CO testing
11. March: iptables IPv6 rules in the default set; SLAAC-style DNS reverse addresses automatically generated alongside explicit forward and reverse entries
12. April: Development meeting "where next"
13. May: COtest wire set up; Windows managed platform tests; additional BIRD RA code and debugging
14. June: User with an IPv6 ISP surfaces! (RT#77905)
15. July: "AT" Windows desktops moved to their own subnet
16. 1st August: IPv6 enabled on wire M
17. August: lcfg-addrwatch, including daemon control and rrd logging
18. August: third systems-blog article
19. 6th September: IPv6 enabled on (Forum) DICE wire
20. October: take to Development meeting for sign-off

Conclusions

As was only to be expected, this has been a rather drawn out and episodic project, as shown in the time-line above. There were periods of fairly intensive development work, interspersed with periods of soak-testing, and not a little conceptual design work. There was also a fair amount of learning required, both from the RFCs and other other documentation, and from experience through the life of the project.

On the Linux side, things went generally smoothly, with most of the implementation work (e.g. converting iptables scripts, DNS tools, writing `BIRD` and `addrwatch` components) being straightforward, albeit somewhat mechanical at times.

Some of the underlying switch management turned out to be less simple, due mainly to the variety and ages of the switches we have in use, and how much or how little IPv6 support they offer (and indeed whether any two models respond in the same way to the same management input!). Generally, the newer the better, and if we are serious in our aspiration to roll out IPv6 throughout our network, and in particular in the student labs and for self-managed users, then we do still have quite a few older models which will have to be replaced.

The overall conclusion at the end of the project is that it is indeed possible for us to support IPv6, to at least some extent. We have a working IPv6 infrastructure, with global connectivity, suitable for use with managed machines. Self-managed machines are pending switch upgrades and DHCPv6.

Now that we have this in place, the focus shifts from the underlying infrastructure to the systems which we run on top of it. At this point it becomes a task for all COs, to look at the services they manage, assess their IPv6-readiness, and take action as appropriate.

The project has had 10 weeks of effort booked against it in total, spread over 16 elapsed months. That's arguably high for an "investigation", but there was necessarily a lot of implementation too so that later stages could be investigated properly. Quite a lot of it should perhaps have been put down as PD instead, though given the nature of the task it would have been very difficult to separate this out from "development" work.

Many thanks to those Informatics Computing and IS network staff who contributed to the project through their helpful discussions and testing effort.

Still to do

There are a few small things still to do, but nothing which impacts on the completion status of the overall project. They'll be swept up as misc-development in due course.

1. MLD snooping: for now multicasts just go everywhere
2. manager-addrs: for now we hard-wire one subnet (B), so that things aren't totally open, and use IPv4 addresses or `-4` as required.
3. %slaac and aliases: just a case of duplicating a few lines of code!

FinalReport.html,v 1.210 2016/10/03 14:42:49 gdmr Exp

Please contact us with any comments or corrections.
Unless explicitly stated otherwise, all material is copyright The University of Edinburgh