IPv6 support for embedded devices

OpenFortress*
digital signatures

Abstract:

Most desktop computers are dual-stack, and networks are moving as well. Embedded devices, limited in resources as they are, seem to be the next important step to take. This proposal discusses why and how embedded devices should be made IPv6-only.

Document descriptive information:

Title:	IPv6 support for embedded devices
Author:	Rick van Rein
Version:	1.0
Date:	February 10, 2011
ASN.1 oid:	N/A
Release:	Public
Legal status:	Informational, OpenFortress assumes no liability

Contents

1 Embedded devices are different **4**

2 Current tunnelling techniques **5**

3 Proposed tunnelling technique: 6bed4 **6**

4 Help wanted **9**

A Application: SIP over IPv6 **10**

1 Embedded devices are different

The most important desktop systems of today support IPv6 as well as IPv4, and have IPv6 enabled by default. This means that the Internet as a whole can now make the transition to IPv6, without the risk that users will be thrown out. Indeed, this is slowly happening.
Embedded devices however, show a vastly different situation. There is no support whatsoever for IPv6 in many devices that we use on our networks, and many are not likely to support it for a long time to come. This means that a vital part of the transition is lagging behind.
The underlying problem appears to be the limited resources available to most devices. The limitations are not only technical issues such as the available memory space, but also pragmatic realities like lack of development time and the cost efficiency of support on complex solutions.
For this reason, we expect that IPv6 support in embedded devices will be single stack for as long as they can get away with it. And that may actually be very long; as long as IPv4 is working on networks everywhere, it is likely that embedded devices will maintain their position, not to be changed until IPv6 is omnipresent. This may lead to a situation where IPv4 will never fully disappear, which is not desirable because it implies more complex management duties for network administrators.
To alleviate this situation, it would be practical to support IPv6-only embedded devices everywhere on the Internet. That includes supporting them on IPv4-only networks. And as a consequence, it implies tunnelling techniques. We investigated several such techniques, and found none that is very useful to convince programmers of embedded systems to transit from IPv4-only to IPv6-only. Because of this, we propose an alternate tunnelling technique based on an existing one.

```
err = try_autoconfig6 (&cfg);
if ((err != 0) || managed_flag_dhcp6 (&cfg)) {
   err = try_dhcp6 (&cfg);
   if (err != 0) {
      err = try_dhcp4 (&cfg);
      if (err == 0) {
         use_tunnel (&cfg);
         err = try_autoconfig6 (&cfg);
      }
      if (err) {
         panic ();
      }
   }
}
```

Figure 1: Example code for IPv6-only network bootstrapping.

Figure 1 shows a procedure for automatic network bootstrapping on IPv6 if it is

available, with a tunnel fallback based on IPv4 if this is needed. Not shown here are manually configured IPv4 addresses to be used if the network is IPv4-only, but the idea of the relative simplicity of this setup should be clear. It should not be a strain to use this configuration procedure to bootstrap networking for an IPv6-only embedded device.
Various tunnelling techniques may be tried, but for embedded devices it will be useful if a single technique can be rolled out on all networks.

2 Current tunnelling techniques

If an embedded application is to become IPv6-only, it should be able to fall back to a tunnelling technique to cross an IPv4-only network, including home and office networks at end users. This includes passing through NAT, possible through an uncooperative router.
In contrast with these requirements, most current tunnelling techniques are designed with a cooperative router in mind; the idea being that they can be incorporated into such routers. For an embedded device that sits behind any router that has been built in the past years, this luxurious position does not apply; it will have to cope with the router that it is plugged into.
Following is a list of criteria for tunnelling techniques that are necessary to win over the programmers of embedded devices:

- **Standard.** It is good to follow a clear standard for tunnel mechanisms. That way, there should not be situations where some tunnels do, and others do not support certain facilities.

- **Simple.** The most appealing issue for embedded applications is that a tunnel should be as simple as possible. Combined with the straightforward NAT traversal of IPv6, the step to IPv6 can actually be easier than using IPv4, and thus attractive for newly created devices.

- **Any router.** A tunnel technique that works for embedded devices should not make any assumptions on the routers it passes, or in other words it should pass through all types of router, including those who rely on symmetrical NAT.

- **No config.** Although not a technical requirement, it is best if no configuration is needed to get a tunnel going. This is because configurations block plug-and-play devices, and because any form of configuration leads to helpdesk calls meaning that a device is considered less appealing. What this means is that the tunnel mechanism should work without supply of credentials.

- **Traceable.** Even without provision of credentials, a tunnel should not become a portal for network abuse. It should therefore be possible to isolate a misbehaving network node and take corrective measures. One way of making IPv4-registered users traceable in IPv6 is by revealing their IPv4 address as part of their IPv6 address.

- **Stateless.** If possible, a stateless mechanism for the tunnel service would be ideal. This would make it easy to handle downtime on an instance, because traffic can be diverted elsewhere. Also, there is no need to pass through the same server on bidirectional traffic; this perfectly matches the stateless routing properties of IP-level traffic.
- **Anycast.** If possible, it would be ideal to have well-known addresses for the tunnel service, so they can be anycasted by BGP4. This leads to a network service that is resilient and that can do load-sharing among instances. A requirement for this to work would be that the tunnel service is stateless.

Most tunnelling techniques do not implement all these requirements and desires (desires start with *if possible*). Instead, some techniques are useful for one reason and some for other reasons. This explains the current variety in techniques.
If an IPv6-only embedded device is to work on any network, its fallback tunnelling technique should support at least the required points, and the chances of success increase if the desires are also fulfilled.

Goal	6in4	6to4	Softwire	TSP	Teredo	AYIYA	6bed4
Standard	√	√	√	±	±	×	√
Simple	√	√	×	√	×	√	√
Any router	×	×	√	√	×	√	√
No config	×	√	×	√	√	×	√
Traceable	×	√	√	?	?	√	√
Stateless	√	√	×	×	×	×	√
Anycast	×	√	×	×	√	×	√

Figure 2: Comparison of tunnelling techniques.

Figure 2 compares current tunnelling techniques for these properties, and it adds a last column for an alternate proposed here. As can be seen, no implementation is completely suitable for embedded devices, with the exception of our proposal of 6bed4, which can be considered an extension of 6to4 with NAT traversal. The table serves as an explanation of the need of yet another tunnelling technique.

3 Proposed tunnelling technique: 6bed4

The proposed tunnelling technique is 6bed4:

- 6bed4 is IPv6 in UDP in IPv4
- UDP is used to easily pierce through NAT
- IPv4 is used as the fallback connectivity protocol
- Tunnel addresses use a /112 prefix

- Routeable addresses are requested through Router Solicition
- Routeable addresses are provided through Router Advertisement
- Expired sender addresses are replaced through Router Advertisement

The 6bed4 protocol is not a standard in itself, but merely a usage profile for existing standards. On the IPv4 side, this tunnel runs over UDP, which suffices to pass through all kinds of NAT that are in common use in today's World. The only requirement is to keep the NAT translation open, which involves a regular use of the port, possibly for a simple `ping` over ICMPv6.
The 6bed4 protocol has no facilities for accounts and login. This is sensible for an embedded application, because this makes possible the use of the tunnel without configuration of the device. The train of thought enabling this goes as follows:

1. **Not anonymous.** The tunnel user should not be anonymous so abusers can be traced. This is easily achieved by incorporating a public IPv4 address in any IPv6 addresses that is usable through the tunnel.
2. **No registration.** Given that a tunnel user can always be traced back to the IPv4 address that they are using, it is possible to handle tunnel requests without registration of user accounts, or login. The tunnel service will not aggravate problems of address spoofing if it verifies that the IPv6 addresses sent out do indeed match the sender's IPv4 address. That way, egress filtering on IPv4 applies to the IPv6 addresses as well.
3. **Stateless.** Given that there is no registration, a simple check on traffic from the IPv4 side to the IPv6 side is needed, and a reconstruction of IPv4 and UDP headers when traffic returns from IPv6 to IPv4. If the UDP information (notably, the outgoing UDP port of what will often be a NAT router) is also present in the IPv6 addresses used, then the tunnel service has no need for storing any state about the tunnel. It is a simple packet-at-a-time translation service.
4. **Anycast.** Stateless services that only perform simple translations of headers can easily be made into anycast services. Indeed, allocating a /64 on the IPv6 side and a /32 or /24 on the IPv4 side would suffice. Broadcasting these routes from many BGP speakers and hosting them locally in many networks should not be any problem. Interestingly, if this approach becomes more popular, it is likely that local providers will host these anycast addresses on their local networks before they have completed the transition to IPv6 for all their customers. This gives them the benefit of local traffic, leading to lower expenses for backbone traffic.

The IPv6 addresses that can be used through the tunnel are therefore a generic /64 prefix, followed by 32 bits of the IPv4 address as shown to the Internet, as well as the UDP port shown to the Internet for the device that setup the tunnel. Figures 3 and 4 demonstrate the address format and the translation mechanism.

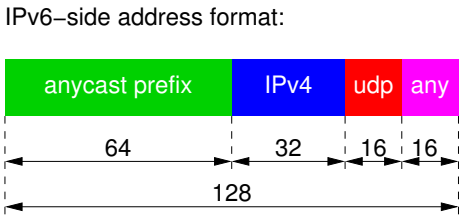

Figure 3: The format of tunnelled IPv6 addresses.

The only matter that remains is how to obtain the IPv6 addresses that can pass through the tunnel. This is achieved with autoconfiguration, using the fact that the boundary between the routeable prefix and the interface identifier can be anywhere. In case of 6bed4, the prefix is 112 bits. The remaining 16 bits are the interface identifier, and can be freely used by the embedded device, provided that identifier 0 is reserved for the tunnel server annex upstream router. The remaining 65535 possible client addresses are plenty for any embedded application — 64k ought to be enough for anyone...

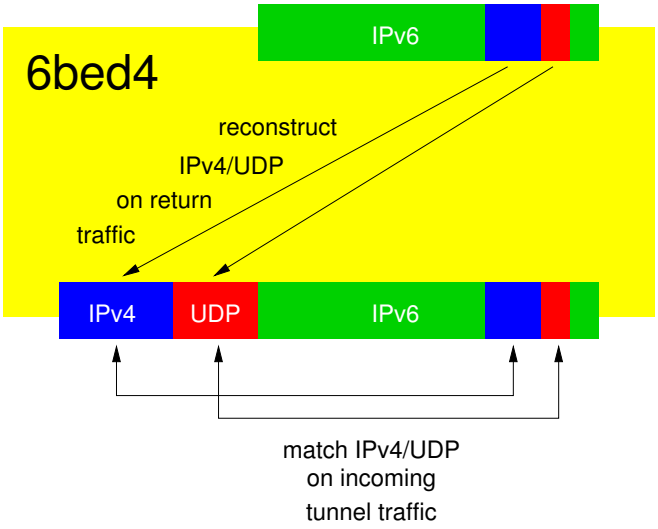

Figure 4: Checks and translation in the 6bed4 service.

Autoconfiguration means that only a single exchange is required between the tunnel client and tunnel server. A Router Solicitation is sent from the client to the server, and a Router Advertisement is replied with the routeable /112 prefix included.

Since there is a one-on-one connection between the tunnel client and the server, even if the server is stateless and so highly predictable, it is possible to set fixed interface identifiers. This is why the router or tunnel server is always assigned identifier 0, and the client can choose anything else. Being in absolute control, there is nothing to stop the client from hard-coding the client address.

The general autocofiguration mechanism continues with neighbour discovery to learn whether someone else is using a prospective IPv6 address. The nature of 6bed4 makes this unnecessary, and the number of such neighbour discovery requests defaults to 0.
If the NAT translation changes due to not using the UDP port for some time, the next outgoing message will have to come from a different IPv6 address than is known to the tunnel. Upon receiving a message from what will then be a false sender address, the tunnel server will drop the message and respond with a Router Advertisement that holds the new prefix to use. The tunnel client can process this change in any way that it likes, as long as future messages are sent from under the new prefix. If higher-layer software resends the originally wrong network packet, it should use the new prefix and derived address.
In summary, a single request-respons interaction suffices to setup a 6bed4 tunnel, and a similarly simple interaction serves to correct for any dynamic changes. The initial IPv6 message sent can be a static message of only 56 bytes long, leading to 98 bytes on the wire; the response IPv6 message is only 88 bytes long, good for 130 bytes on the wire. The mechanism has been implemented and is known to work well.

4 Help wanted

The mechanism introduced in the previous section is implemented in our 6bed4 tunnel server, which will be made freely available under an open source license. In its current version, the server application does not integrate with routing protocols, for instance to report whether a tunnel's link is up, but we are willing to consider requests of this nature.
As can be seen in the appendix, we also have a good example in the making, and we will distribute the embedded software as an open source application, ready for device vendors to use.
The one thing we cannot arrange ourselves, is the server-side hosting and connectivity of 6bed4. Ideally, this protocol would be locally implemented in many network places, but initially it could be setup by a few parties that are near the backbone and that want to help IPv6 advance. If you are such a party, we are hereby asking for your help.
If we can get this setup working as a reliable part of the infrastructure of the Internet, it will be straightforward for manufacturers of embedded devices to move over to IPv6. It is the omnipresence of IPv6 that makes it possible for these single stack implementations to move over, if instead we continue on the current path of transition it will take for a long, long time until this transition happens.
As soon as 6bed4 is active, embedded devices can start to support IPv6. Inasfar as those devices contact other hosts on the Internet, there will even be a rush to implement IPv6 on those too; this should not normally be a problem, given that any counter-parties of embedded devices are usually servers, which should easily be able to implement IPv6. There would actually be an acceleration in using IPv6 as a result of such IPv6-only embedded devices.

A Application: SIP over IPv6

The work presented in this document is actually something that we ran into while working on a specific application. We understood this to be a general problem that needed coverage in the Internet as part of a successful transition to IPv6.

The service that we are developing based on the aforementioned infrastructure is IPv6-only SIP telephony. SIP is an obvious example of a protocol that suffers from IPv4 and NAT, and pulling it up to an IPv6 service will greatly enable the end user facilities for calling over the Internet using a plethora of media formats. To do this well, the only path is to go IPv6-only, and that means a clean break with the current SIP technology. Still, given the rather enclosed roll-out of SIP to date, this is still possible.

We are developing open source firmware for embedded devices that do SIP in this IPv6-only fashion, and like the general class of embedded devices, this calls for tunnelling techniques. Like for general embedded devices, if such phones are to be plug-and-play, they need a public tunnel service that can be trusted to work on any network, regardless of the router employed.

With the ability of direct end-to-end calls, it is finally possible to setup calls through ENUM and similar mechanisms; and, it is finally possible to use services such as `freenum.org` for more than just university-to-university calls. Finally, being detached from RTP proxies means that the kinds of media that can flow may be much more diverse than with an RTP proxy that is hosted by a telecom operator. A phone could send along the IPv6 address of a TV on the LAN, for instance. Or a whiteboard. Computer helpdesk services can access a desktop over SIP.

In short, as soon as the practical barriers that currently hold back SIP have been removed, that is as soon as SIP runs over IPv6, SIP can become the platform of innovation for which it was originally designed.

Notes:

rick@openfortress.nl

http://openfortress.nl

OpenFortress*
digital signatures