Project Number:	**IST-2001-32603**
Project Title:	**6NET**
CEC Deliverable Number:	**32603/WWU(JOIN)/DS/233/A1**
Contractual Date of Delivery to the CEC:	31st December 2003
Actual Date of Delivery to the CEC:	15th March 2004
Title of Deliverable:	Updated IPv4 to IPv6 transition cookbook for end site networks/universities
Work package contributing to Deliverable:	WP2
Type of Deliverable*:	R
Deliverable Security Class**:	PU
Editors:	Christian Schild (WWU/JOIN), Tina Strauf (WWU/JOIN), Tim Chown (UoS)
Version:	1.0
Contributors:	Stig Venaas (UNINETT), Pekka Savola (CSC), Christian Strauf (WWU/JOIN), Chris Edwards (ULanc), Martin Dunmore (ULanc), Octavio Medina (ENST), Jérôme Durand RENATER), Feico Dillema (Invenia), Kostas Koumantaros (GRNET), Dimitrios Kalogeras (GRNET), Athanassios Liakopoulos (GRNET), Thorsten Kersting (DFN), Michael Mackay (Ulanc), Kamal Deep Singh (ENST), Tommi Saarinen (University Oulu)

Abstract:

This is the third version of an IPv4 to IPv6 transition cookbook for end site networks and/or universities. After an introduction to the basics of transitioning from IPv4 to IPv6 (which presently is generally done by moving to dual-stack networking) and a brief description of each mechanism on a theoretical basis, we give a description of some example scenarios to give the reader an idea of where and when to employ certain transition methods and how different mechanisms work together and complement each other. The next part of the document then focuses on installation and configuration examples. The deliverable is a "living document" and as such will be updated and revised whenever there is new or different material available, up until the point of the final version due in December 2004 (Deliverable D2.3.4).

Keywords: IPv4 to IPv6 transition, transition mechanisms, IPv6 site transition, dual-stack networking, IPv6-only networking

Executive Summary

As IPv6 grows in maturity in terms of standards and implementations, and as understanding of its benefits grow, deployments, both pilot and production, will also increase in number.

Within the 6NET project, many of the participants have already made a significant investment in deploying early IPv6 services, mainly dual-stack, but a small number IPv6-only. In this document, we present a cookbook of theory and practice for IPv6 site deployment.

The cookbook includes information on the theory of various transition methods that have been proposed to date, as well as examples using popular IPv6 operating systems such as FreeBSD, Linux, Solaris and Windows and router platforms such as Cisco IOS, Juniper JunOS and Zebra.

This cookbook will be updated as and when major new material is added during 2004, and then be replaced by a subsequent, more complete and final Deliverable (D2.3.4) in December 2004. The final version will also include more detailed migration scenarios/reports (including case studies that are being undertaken jointly with the Euro6IX project), which this version cannot yet present, as well as a more comprehensive list of implementations.

This document contains little to no coverage on routing issues and the tasks one needs to perform to set up a network with internal or external routing protocols. The reader is instead referred to D3.1.2, where these issues are addressed and covered in detail (e.g. considerations for using IS-IS for IPv4 and IPv6).

The authors welcome input on this cookbook, which should be directed to the principal editors of this document (join@uni-muenster.de).

Table of Contents

Table of Figures

1. Introduction

IPv6 support is now widely available for all common host and router platforms. There are methods readily available to configure and run IPv6 on a single host or within a small site.

With an IPv6 host or local network configured, getting connectivity to the global IPv6 Internet is vital if you wish to communicate with other IPv6 systems. Today, this is usually accomplished either natively or, more commonly, with the IPv6-in-IPv4 tunneling technique using either manual or automatic tunnel configuration methods.

In this document, we will describe configurations for single systems, and how to plan the deployment of IPv6 in some example scenarios. The academic 6NET project participants are, on the whole, fortunate enough to have native connectivity to their National Research and Education Networks (NRENs) and from there to a globally connected native IPv6 network (spanning 6NET, GÉANT, and links to Abilene in the US and WIDE in Japan). Other sites may not be so lucky; for them a tunneling mechanism is the only realistic option for IPv6 connectivity.

In D2.3.1 we described the IPv6-only site deployment at Tromso. IPv6-only deployments are rare, especially in Europe, but are an interesting exercise with a view to the end game of IPv6 deployment. However, the practical reality is that sites deploying IPv6 will not migrate to IPv6-only, but transition to a state where they support both IPv4 and IPv6 (dual-stack). The dual-stack environment then allows IPv6-only devices to be introduced, as a site slowly phases out IPv4. For this reason, translation mechanisms between IPv4 and IPv6 systems are less frequently required; however we discuss transition mechanisms, from view of the network, transport and application layers at which translation may be applied.

Expanding IPv6 functionality from a small infrastructure to a large site network is a complex and difficult venture. For a large site there are lots of different needs, and different conditions which make it necessary to employ various transition mechanisms according to the peculiarities of, for example, a given subnet, wireless or mobile environment or dial-in technology. In this cookbook we want to explain in detail which potential options and techniques exist to integrate IPv6 into a site network, which solution is appropriate for any special kind of network infrastructure and of course how exactly one has to set up and configure these techniques. Where possible, we also point to existing (current) problems and interoperability issues, for example in running IPv4 and IPv6 in parallel or having IPv6-only hosts which still need to be able to communicate with IPv4-only hosts on occasion. In the scenarios section, we describe the components identified for transition for a medium sized academic network (1,500 users, up to 1,000 systems).

In this document we differentiate between three general ways of integrating IPv6 into a network, namely dual-stack, IPv6-in-IPv4 tunneling mechanisms, and an IPv6-only infrastructure (where translation techniques are deployed). In the second chapter we describe each of these scenarios in more detail focusing on the general peculiarities and difficulties one has to face when trying to implement one or the other in any part of a network.

The third chapter covers the possible transition mechanisms and methods on a theoretical basis. It describes the way they work as well as where they are best deployed and if there are any known security and interoperability issues pertaining to a given mechanism in general. This chapter is also divided in three sections covering mechanisms for dual stack, tunneling and translation in turn.

Before the more technical part of this document begins with chapters five to seven the fourth chapter contains descriptions of some “real world” transition scenarios. These scenario descriptions are meant for the reader to get an idea of how different transition mechanisms can be used together to migrate a whole network to IPv6. The next three chapters (five to seven) make up the heart and most important part of this document and contain the example configurations for various implementations of the previously described transition mechanisms. Once the reader is familiar with the theory of a certain mechanism these chapters should enable them to deploy it on any platform covered in this document. Where relevant, these chapters will also contain information about known security and interoperability problems where they are specific to a certain implementation of a transition mechanism.

In Appendix A, the reader will find a list of available implementations of transition tools. The list might even contain a few implementations that are not (yet) covered in chapters five to seven. There is no guarantee that this list is ever going to be complete, of course, as the development work is still going on for a lot of platforms.

Appendix B shows how to make platforms and operating systems ready for IPv6. It is not yet clear if this section will be included in the final release of the deliverable as the information contained here might become totally unnecessary in the near future where more and more platforms are IPv6 enabled by default. However if the section is present in the final revision of the deliverable it will focus mostly on operational issues and experiences related to switching on IPv6 on a host or router.

2. General Overview

IPv6 can be introduced to a site in three basic ways, categorized into how connectivity of each system to the IPv6 Internet is achieved and how the network and hosts themselves have to be enhanced to make IPv6 possible. We call these categories “dual stack”, “additional IPv6 infrastructure” (which generally involves IPv6-in-IPv4 tunnelling) and “IPv6-only networking”. In a large-scale network one will never only use one technique or another. It is much more likely that the best way for getting systems connected within a site has to be decided for each subnet or even each host anew, which leads to the deployment of many different techniques according to the different demands and peculiarities of a certain part of the network. This Chapter focuses on the theoretical description of each or the three general transition scenarios “dual stack”, “additional IPv6 infrastructure” and “IPv6-only networks”. Descriptions of special transition mechanisms as they come into place within those scenarios are included in the next Chapter.

Once the internal networking is determined, the next step is to arrange external connectivity for the whole site, which involves external routing issues and is really only possible either natively or via some tunnelling mechanism. With IPv6, the choice of external connectivity method will determine the IPv6 addressing prefix within the site. A site prefix is usually (as recommended by the RIRs) a /48 prefix, which allows 16 bits of subnet address space for the /64 subnets.

We discuss security issues in this section. An overview of security issues in transition is available as an Internet Draft produced from 6NET experience [TRANSSEC].

2.1. Dual Stack

One of the conceptually easiest ways of introducing IPv6 to a network is called the “dual stack mechanism” [RFC2893]. Using this method a host or a router is equipped with both IPv4 and IPv6 protocol stacks in the operating system (though this may typically be implemented in a hybrid way). Each such node, called an “IPv4/IPv6 node”, is configured with both IPv4 and IPv6 addresses. It can therefore both send and receive datagrams belonging to both protocols and thus communicate with every node in the IPv4 and IPv6 network. This is the simplest and most desirable way for IPv4 and IPv6 to coexist and is most likely to be the next step in a network’s evolution in general, before a complete transition to an IPv6-only Internet can be achieved worldwide.

There are no real transition mechanisms to use within the dual stack scenario, as “Dual Stack” is a method to integrate IPv6 itself.

One challenge in deploying an IPv6/IPv4 Dual Stack network lies in configuring both internal and external routing for both protocols. If one has for example used OSPFv2 for intra site routing before adding IPv6 to the Layer 3 network one will either have to make the transition to OSPFv3 or IS-IS necessary or one will at least be forced run one of these IGPs in addition to OSPFv2. Over the course of writing this document there has been a deliverable released by WP3 within the 6NET project. This deliverable (D3.1.2) is also a cookbook and focuses on configuration examples for IPv6 routing both intra- and inter-domain. Since configuring IPv6 routing in a dual stack network is usually completely independent from the configuration of IPv4 routing the reader is asked to refer to this cookbook for most of the issues concerning basic routing setup. This cookbook will focus exclusively on the issues arising when doing both IPv4 and IPv6.

Another challenge lies in the interaction of the two protocols, and how this interaction is managed, given that a dual-stack network will (in an early stage of worldwide IPv6 deployment) generally be interacting with IPv4 external networks. An example is the deployment of email servers for SMTP, and how the MX servers are provisioned for both protocols by offering IPv4 or IPv6 reachability, and how failover is handled between the protocols. These issues will be discussed in the scenarios section.

2.2. Additional IPv6 infrastructure

By additional IPv6 infrastructure we mainly mean tunnelling techniques that one can use on top of the present IPv4 infrastructure without having to make any changes to the IPv4 routing or the routers. This method is often used where parts of or the complete infrastructure is not yet capable to offer native IPv6 functionality. Therefore IPv6 traffic has to cross the existing IPv4 network, which is possible with several different tunnelling techniques described in the following Chapter. These techniques are often chosen as a first step to test the new protocol and to start integration of IPv6.

Tunneling is also called encapsulation. It is a process by which information from one protocol is encapsulated inside the packet of another protocol architecture, thus enabling the original data to be carried over the second protocol. This mechanism can be used when two nodes that use the same protocol want to communicate over a network that uses another network protocol. The tunneling process involves three steps: encapsulation, decapsulation, and tunnel management. It requires two tunnel endpoints, which in the general case are dual-stack IPv4/IPv6 nodes, to handle the encapsulation and decapsulation. There will be performance issues associated with tunneling, both for the latency in en/decapsulation and the additional bandwidth used, though the latter is usually marginal.

A tunnel can be configured in four different ways:

1. Router to router, which spans one segment of the end-to-end path between two hosts.
2. Host to router, which spans the first segment of the end-to-end path between two hosts.
3. Host to host, which spans the entire end-to-end path between two hosts.
4. Router to host, which spans the last segment of the end-to-end path between two hosts.

Depending on what kind of setup is used, a tunnel might be “configured” (both sides need to be configured accordingly), “semi-configured” (only one side has to be configured, the other side acts as a gateway) or “automatic”, where nearly nothing needs to be done for the two hosts to communicate via a tunnel.

2.3. IPv6-only Networks

In IPv6-only networks communication between nodes is just that: IPv6-only. Communication between a node on the IPv6-only network and a remote node reachable only over IPv4 is not possible, because the hosts can only communicate using IPv6 at the network layer. This is where translation techniques come into place, which can operate at many different layers. We categorize translation techniques by the layer they may appear in, that is the network layer, the transport layer or the application layer.

In the network layer the header of a datagram is translated from IPv6 to IPv4 (or vice versa), which happens in the operating system of the originating host. In the transport layer the general mechanism is the use of a relay, which the data has to pass through. This relay is commonly a dual stack device that will translate and pass datagrams between the different networks. In the application layer an "application layer gateway" (ALG) is used, e.g. a web proxy. While in the network and transport layer translating and relaying IPv4/IPv6 datagrams is mostly application independent, application layer gateways have to be set up for each and every application or service one wants to offer.

3. Description of Tools and Mechanisms

This chapter will describe the different transition mechanisms that can be used within each of the scenarios “dual stack”, “IPv6 tunnelling techniques” and “IPv6-only networks” in theory. Where necessary also general operational as well as security issues are addressed for each mechanism.

3.1. Dual Stack

As mentioned previously there are not really any transition mechanisms or tools involved as dual stack is a method to integrate IPv6 itself. To make a node a dual stack node one just has to switch on IPv6 on most platforms. This way the node becomes a “hybrid stack” host, given that elements of the protocol stacks are likely to be shared in the implementation.

A network or backbone becomes dual-stack if the routers and switches building the network not only route and handle IP(v4) but also route and handle IPv6.

3.1.1. Security Considerations for Dual-Stack Networks

Making a network/host dual-stack and providing this host/network with IPv6 connectivity to the rest of the IPv6 Internet does not pose a security threat in itself. It will just add a completely separate IP infrastructure for which the same or similar measures have to be taken for it to become as secure as the IPv4 network.

For example, if the parallel IPv4 network (or a single host) is protected by a (personal) firewall with certain IPv4-sepcific rule sets it should be taken care that this firewall is either extended with IPv6 support and corresponding rule sets for IPv6 or a separate IPv6-only firewall should be implemented that performs the task of securing the hosts and network the same way for IPv6 as the IPv4 firewall does for IPv4.

Another example is access-lists. If IPv4 access-lists are in place protecting parts of the network by preventing access from or to other parts of the network, IPv6 access-lists have to be created accordingly to implement the same restrictions when hosts communicate via IPv6.

3.2. IPv6 Tunneling Methods

In this section we describe methods for carrying IPv6 over existing IPv4 networks, which invariably means using some kind of tunnelling mechanism, either manually configured or automatically configured.

3.2.1. Configured Tunnel

Configured tunnelling is defined in RFC 2893 [RFC2893] as IPv6-over-IPv4 tunnelling where the IPv4 tunnel endpoint address is determined by configuration information on the encapsulating node. Therefore the encapsulating node must keep information about all the tunnel endpoint addresses. These kinds of tunnels are point-to-point and need to be configured manually. For control of the tunnel paths, and to reduce the potential for tunnel relay denial-of-service attacks, manually configured tunnels can be advantageous over automatically configured tunnels.

Configured tunnels are best employed when providing external IPv6 connectivity to a whole network. There are not yet many providers who offer IPv6 in any way but if one has the possibility to get initial IPv6 connectivity from another site, one of the easiest, most stable and secure ways to get the IPv6 traffic routed to the yet unconnected site is via an IPv6-in-IPv4 tunnel. One can even set up a BGP peering on that link, although if the tunnel is the only off-site link, BGP is not required unless the connecting site is interested in seeing the BGP routing information of its upstream provider.

Within a site, configured IPv6-in-IPv4 tunnels can also be used if there is a part of the network that can (for whatever reason) not be natively connected to the rest of the IPv6 topology. Since these kinds of tunnels have to be configured by hand this makes only sense if there is only a requirement for just a few of those tunnels. There is no point in connecting a lot of different isolated hosts by a configured tunnel or a lot of different isolated subnets. There are other tunnelling methods specifically designed for this purpose, such as ISATAP, a tunnel broker or 6to4.

Note that IPv6-in-IPv4 tunnels may be used for OSPFv3 routing but not with IS-IS as IS-IS is based on Layer 2 while IPv6-in-IPv4 tunnels are completely Layer 3.

3.2.1.1 Security Considerations for IPv6-in-IPv4 Tunnels

If a IPv6-in-IPv4 tunnel should be set up to a host behind an IPv4 firewall it is necessary to open that firewall for protocols 41 (IPv6) and 58 (ICMPv6) at least for the IPv4 address of the host at the remote end of the tunnel, which will be the source of the incoming IPv4 traffic that contains the IPv6 packets. Ideally it should be possible to restrict this opening to only those IPv4 packets destined for the local tunnel endpoint, limiting the connection to the agreed endpoints. This will leave the site relatively protected concerning IPv4.

In addition to possible attacks through IPv4 though, now also security attacks against IPv6 must be considered for all hosts, which will eventually be connected via IPv6 to the tunnel endpoint. Firewalls with IPv6 support have separate rule sets for IPv6 and IPv4. Few though, if any, are designed to unroll the tunnelling protocol (this applies also to tunnelling with 6to4, Teredo, ISATAP, etc.) and thus apply the rules directly to the encapsulated traffic. The problem with that is, that to an IPv4 firewall rule set, SIT (normal IPv6-in-IPv4 tunnels) as well as 6to4 traffic looks like nothing more than protocol 41 on IPv4. To an IPv6 firewall rule set, SIT and 6to4 do not exist. So neither rule set applies directly to the tunnelled traffic beyond switching protocol 41 on or off. Bitmapped rules, which mask against encapsulated payloads, are a difficult and error prone workaround.

In the case of setting up configured IPv6-in-IPv4 tunnels or 6to4 for external connectivity of a site/subsite it is best to separate the two tasks of securing IPv4 and IPv6. This can be done by employing IPv6 security mechanisms after the IPv6 packets have been decapsulated (ideally immediately at the local tunnel endpoint). For example special IPv6 access lists and IPv6 firewalls can then be easily used to extend the same level of security to IPv6 as is present at the site border for IPv4 traffic.

3.2.2. Tunnel Broker

Instead of manually configuring each tunnel endpoint it is possible to use executable scripts instead. This “automatic” alternative is called a “tunnel broker” and is presented in RFC 3053 [RFC3053].

Like manually configured tunnels, the tunnel broker is useful where a user has a dual-stack host in an IPv4-only network, and wishes to gain IPv6 connectivity. The basic philosophy of a tunnel broker is that it allows a user to connect to a web server, (optionally) enter some authentication details, and receive back a short script to run and establish an IPv6-in-IPv4 tunnel to the tunnel broker server.

The operation of a typical tunnel broker service is illustrated in Figure 3-1. The provider of a tunnel broker service needs to provide a web server (available over IPv4) and a (dual stack) router device capable of accepting setup commands to create new tunnels to client endpoints. It is possible that both functions can be served from one machine.

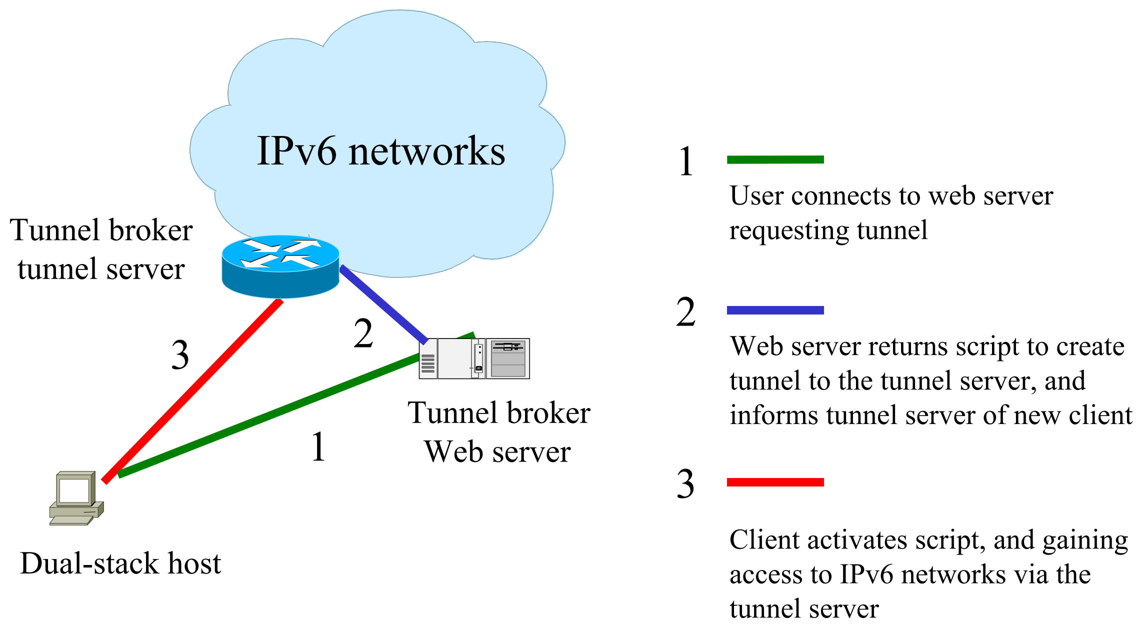

IPv6 networks
Tunnel broker tunnel server
3
2
1
Tunnel broker Web server
Dual-stack host
1
User connects to web server requesting tunnel
2
Web server returns script to create tunnel to the tunnel server, and informs tunnel server of new client
3
Client activates script, and gaining access to IPv6 networks via the tunnel server

Figure 3-1: Tunnel broker components and set-up procedure

A tunnel broker can be implemented in many ways. The requirement for the service is that it needs to keep track of the tunnels created and whom they belong to. Ideally it should have some authentication to grant access to the service, but in practice early implementations have not required this. The Freenet6 service (http://www.freenet6.net) is perhaps best known, but being based in Canada is not ideal for use in European networks (the first hop to any destination would be thousands of miles away); an ISP should offer local tunnel broker facilities for its users where no native IPv6 service is present.

Tunnel brokers can serve subnet tunnels, as well as single host tunnels. In such cases the host obtaining the tunnel is in reality a router, and the mechanism for obtaining the tunnel can be more generic (using for example TSP, the tunnel setup protocol [TSP]), and may need specific functions to activate or deactivate the tunnel.

The tunnel broker service is generally easy to use for the client, but there are some concerns with the deployment of server systems, e.g. in security of access, and in re-allocation of tunnels where clients use dynamic IPv4 addresses (as is typical behind commodity dialup). As with other tunnel methods, any intervening firewall must pass Protocol 41 to/from the tunnel server. Where that is not required, a site administrator may be blissfully unaware of users on their site who use tunnel brokers, thus not creating any site demand for "proper" IPv6 deployment.

A tunnel broker is an important transition aid; it enables easy-to-use IPv6 network access, and we expect to see a number of supported brokers used in the 6NET environment during the project. The tunnel brokers may be deployed by sites (universities) or by NRENs (as not every university will wish to run its own broker). If no broker is available to a national participant, remote brokers may be used, but doing so will naturally reduce the efficiency of the tunnelling, since the first IPv6 hop for the client will be in the (distant) remote network, even if the target is relatively local.

3.2.3. Automatic Tunnels

This type of tunnel mechanism uses IPv4-compatible IPv6 addresses on the tunnel endpoints. The address of the recipient node is specified by the packet that is being encapsulated. This method can only be used on router-to-host and host-to-host communication since these are the only schemes where one tunnel endpoint is also the recipient. Due to the use of particular addresses it only works on IPv6 over IPv4 tunnelling and not vice versa.

It is currently widely felt that automatic tunnelling should be deprecated. One reason for that lies in the ad-hoc nature of connectivity that results, lacking structure in the IPv6 domain; solutions such as ISATAP or 6to4 (see below) are generally considered preferable. IPv4-compatibe addressing is being written out of the update to RFC2893.

3.2.4. 6to4

The transition mechanism known as 6to4 [RFC3056] is a form of automatic router-to-router tunneling that uses the IANA-assigned IPv6 TLA prefix 2002::/16 to designate a site that participates in 6to4. It allows isolated IPv6 domains to communicate with other IPv6 domains with minimal configuration. For that an isolated IPv6 site will assign itself a prefix of 2002:V4ADDR::/48, where V4ADDR is the globally unique IPv4 address configured on the appropriate interface of the domain's egress router (see Figure 3-2). This prefix has exactly the same format as normal /48 prefixes and thus allows an IPv6 domain to use it like any other valid /48 prefix. In the scenario where 6to4 domains wish to communicate with other 6to4 domains, no tunnel configuration is needed. Tunnel endpoints are determined by the value of global routing prefix of the IPv6 destination address contained in the IPv6 packet being transmitted which includes the IPv4 address. In this scenario, an arbitrary number of 6to4 domains may communicate without the need for any tunnel configuration. Furthermore, the 6to4 routers do not need to run any exterior IPv6 routing protocol as IPv4 exterior routing performs the task instead.

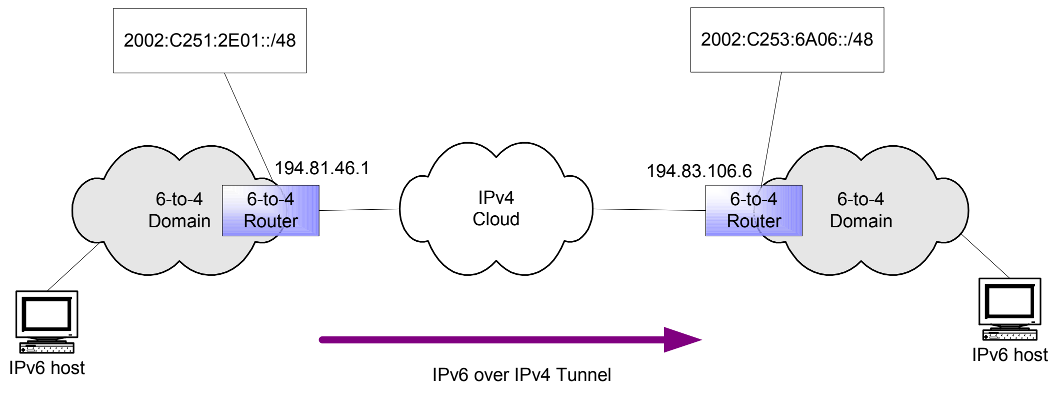


Figure 3-2: 6to4 service overview

When 6to4 domains wish to communicate with IPv6-only domains however, the situation is a little more complex. In this case, connectivity between the domains is achieved via a relay router, which is essentially a router that has at least one logical 6to4 interface and at least one native IPv6 interface. Unlike with the previous scenario, IPv6 exterior routing must be used. The relay router advertises the 6to4 2002::/16 prefix into the native IPv6 routing domain. In addition the relay router may advertise native IPv6 routes into its 6to4 connection. The relay router can be discovered using IPv4 anycast, as described in RFC 3068 [RFC3068].

Most ISPs/NRENs only advertise their 6to4 relay within their own network. There are very few “public” relays, in part due to the security concerns described below. Despite such concerns, and concerns over provision of Multicast over 6to4, the protocol does offer a very convenient, automatic way to gain IPv6 connectivity to an IPv4-connected site.

The general use of 6to4 is as a mechanism for an IPv6 site border router with only IPv4 external connectivity to establish automatic connectivity to the IPv6 public Internet. Other methods (e.g. ISATAP, or native IPv6 networking if available) can then be used inside the site. It can also be used on a host, but such usage is (we expect) rather less common and not originally intended.

3.2.4.1 Security Considerations with 6to4

There are concerns about the security of 6to4 relay devices (see deliverable 6.2.2 [D6.2.2]), which may be used for remote denial-of-service attacks. 6to4 is meant to provide an automatic way of connecting a site to any outside 6to4 site without special configuration, so on a 6to4 host incoming 6to4 traffic will be accepted and decapsulated from any source from which regular IPv4 traffic is accepted. Using an IPv4 firewall to restrict incoming packets with protocol 41 to only specific IPv4 source addresses is therefore not possible (see also section 3.2.1.1 on security issues with manually configured IPv6-in-IPv4 tunnels.). If the firewall provides this functionality one can however still restrict the traffic so only packets to specific destination addresses, specifically to the addresses of those routers which use 6to4 is let through.

If IPv6 address spoofing is felt to be a threat, a plausibility check on weather the encapsulating IPv4 address is consistent with the encapsulated 2002::/16-address can be used on the 6to4 routers. If such a check is applied, exceptions to it must be built in to admit traffic from relay routers. 2002::/16-traffic must also be excluded from checks applied to prevent spoofing of “6over4” traffic (see section 3.2.5.1). In any case, 6to4 traffic whose source or destination address embeds an IPv4 address which is not in the format of a global unicast address must be discarded. Specifically, this means that IPv4 addresses defined in RFC 1918, broadcast, subnet broadcast, multicast and loopback addresses are unacceptable. For more information on 6to4 security issues please also refer to the current version of [6TO4SEC].

3.2.5. 6over4

6over4 is defined in RFC 2529 [RFC2529]. It interconnects isolated IPv6 hosts in a site through IPv6-in-IPv4 encapsulation without explicit tunnels. It uses IPv4 addresses as interface identifiers and creates a virtual link using an IPv4 multicast group with organization-local scope. IPv6 multicast addresses are mapped to IPv4 multicast addresses to allow neighbour discovery. The 6over4 method has fallen out of favour due to a number of reasons, including the general lack of IPv4 multicast support in site/ISP networks.

There have been a small number of implementations, including those by 3Com and Cisco, but practically no adoption. We thus do not consider 6over in any detail, as the method seems (in effect) deprecated.

3.2.5.1 Security Considerations with 6over4

If 6over4 is used there is a possible spoofing attack in which spurious 6over4 packets are injected into a 6over4 domain from outside. Thus boundary routers must discard multicast IPv4 packets with source or destination multicast addresses of organization local scope (should be 192.X.X.X, see [RFC2365] for details), if they arrive on physical interfaces outside that scope. To defend against spurious unicast 6over4 packets, boundary routers must discard incoming IPv4 packets with protocol type 41 from unknown sources (see exceptions for 6to4 in section 3.2.4.1). Unless IPsec authentication is available, the recommended technique for this is to configure boundary routers only to accept protocol type 41 packets from source address within a trusted range or ranges.

3.2.6. ISATAP

An alternative to 6over4 is ISATAP (Intra-Site Automatic Tunnel Addressing Protocol) [ISATAP]. ISATAP also uses the site's IPv4 infrastructure as a virtual link, but it does not use IPv4 multicast, so the link is NBMA (Non-Broadcast Multiple Access).

ISATAP, like 6over4, creates an interface identifier based on the interface's IPv4 address. ISATAP supports both autoconfiguration and manual configuration of addresses, but the IPv4 address of the interface will be embedded as the last 32 bits of the IPv6 addresses. As with 6over4, the IPv4-addresses need not be globally unique.

Usually multicast is used for neighbour discovery operations like address resolution and router solicitations or advertisements. Since the IPv4 address is always embedded in the IPv6 address, address resolution is trivial. For router solicitations to work, the host must somehow have learned of IPv4 addresses of possible ISATAP routers (through DHCP, DNS, manual configuration etc)., and will then send solicitations as unicast. The router always sends advertisements as unicast and only as a reply to a host's solicitation. Each ISATAP host will regularly send solicitations to the ISATAP routers it knows of.

3.2.6.1 Security Considerations with ISATAP

In addition to the security considerations which where mentioned for neighbour discovery and stateless address autoconfiguration, site administrators must ensure that lists of IPv4 addresses representing the advertising ISATAP interfaces of potential router list (PRL) members are well maintained. The PRL provides the policy for trusting router advertisements, which is problematic considering that the PRL itself could possibly not be trusted. That is why administrators must ensure that lists of IPv4 addresses representing the advertising ISATAP interfaces of PRL members are well maintained. If the PRL is compromised, many different attacks are possible. An analogy can be found in rough DHCP servers being deployed on a network.

3.2.7. Teredo

Teredo [TEREDO] (formerly known as Shipworm) is designed to make IPv6 available to IPv4 hosts through one or more layers of NAT [RFC1631] by tunnelling packets over UDP. It uses two entities: a Teredo server and a Teredo relay. The server listens for requests from the clients, responding with an IPv6 address for them to use. It forwards the IPv4-encapsulated IPv6 packets sent from the clients to the relay, and also forwards the IPv6 packets received from the Teredo relay. The relay acts as an IPv6 router. Teredo is very much a "last resort" tool, only designed to be used where no other method will work (e.g. in an enterprise network using RFC1918 addresses and offering no IPv6 transition support mechanisms). The Teredo method is complex, and cannot be guaranteed to work due in part to varieties in NAT implementations.

3.2.7.1 Security Considerations for Teredo

The threats posed by Teredo can be grouped into four different categories:

1) Opening a hole in the NAT
2) Using the Teredo service for a man-in-the-middle attack
3) DoS of the Teredo Service
4) DoS against non-Teredo nodes

These four types of threats as well as possible mitigating strategies are addressed below.

Opening a Hole in the NAT

As Teredo is designed to make a machine reachable via IPv6 through one or more layers of NAT, the machine using the service consequently give up any firewall service that was available in the NAT box. All services opened for local use will become potential targets for attacks from the entire IPv6 Internet. It is recommended to use a personal firewall solution, i.e. a piece of software that performs the kind of inspection and filtering locally that is otherwise performed in a perimeter firewall as well as the usage of IPv6 security services such as IKE, AH, or ESP.

Man-in-the-Middle Attacks

The goal of the Teredo service is to provide hosts located behind a NAT with a globally reachable IPv6 address. There is a possible class of attacks against this service in which an attacker somehow intercepts the router solicitation, responds with a spoofed router advertisement and provides a

Teredo client with an incorrect address. The attacker may have one of two objectives: a) it may try to deny service to the Teredo client by providing it with an address that is in fact unreachable, or b) it may try to insert itself as a relay for all client communications, effectively executing a man-in-the-middle attack. It is not possible to use IPv6 security mechanisms such as AH or ESP to ward of these kinds of attacks since the cover only the encapsulated IPv6 packet but not the encapsulating IPv4- and UDP header. In fact it is very hard to find an effective signature scheme to prevent such an attack since the attacker does not do anything else than what the NAT legally does. The Teredo Client should systematically try to encrypt outgoing IPv6 traffic using IPsec. That will at least make spoofing of the IPv6 packets impossible and prevent third parties from listening in to the communication. By providing each client with a global IPv6 address Teredo enables the use of IPsec.

Denial of the Teredo Service by Server Spoofing or an Attack of the Servers

Spoofed router advertisements can be used to insert an attacker in the middle of a Teredo conversation. The spoofed router advertisements can also be used to provide a client with an incorrect address pointing to either a nonexistent IPv4 address or to the IPv4 address of a third party. The Teredo client will detect the attack when it fails to receive traffic through the newly acquired IPv6 address of the so-called Teredo server. Using authentication encapsulation this attack can be prevented.

Other than confusing clients with false server addresses the Teredo service can of course also be disrupted by mounting a Denial of Service attack against the real Teredo servers and relays sending a huge number of packets in a very short time. Since Teredo servers are generally designed to handle quite a large amount of network traffic this attack most likely will have to be quite brute force, if it should work at all. The attack is mitigated if the Teredo service is built redundantly and the clients are ready to “failover” to another server. That will of course cause the clients to renumber.

If a Teredo relay is attacked in such a way it should stop announcing the reachability of the Teredo service prefix to the IPv6 network. The traffic will be picked up by the next relay.

Denial of Service against non-Teredo Nodes

There is a widely expressed concern that transition mechanisms such as Teredo can be used to mount denial of service attacks by injecting traffic at locations where it is not expected. These attacks fall into three categories: a) using the Teredo server as a reflector in a denial of service attack, b) using the Teredo server to carry a denial of service attack against IPv6 nodes and c) using the Teredo relays to carry a denial of service attack against IPv4 nodes. A common mitigating factor in all of these cases is the “regularity” of the Teredo traffic which contains highly specific patterns such as the Teredo UDP port or the Teredo IPv6 prefix. In cases of attacks these patterns can be used to quickly install filters and remove the offending traffic.

3.2.8. Tunnel Setup Protocol (TSP)

The Tunnel Setup Protocol (TSP) [RFC3053] is a general method designed to simplify the setup of (authenticated) IPv6 tunnels over IPv4 networks. The TSP method was initially applied to tunnel brokers, but the scope of the TSP applicability is much wider.

3.2.9. DSTM

DSTM (Dual Stack Transition Mechanism) [DSTM] is a tunneling solution for IPv6-only networks, where IPv4 applications are still needed on dual-stack hosts within an IPv6-only infrastructure. IPv4 traffic is tunnelled over the IPv6-only domain until it reaches an IPv6/IPv4 gateway, which is in charge of packet encapsulation/decapsulation and forwarding between the IPv6-only and IPv4-only domains. The solution proposed by DSTM is transparent to any type of IPv4 application and allows the use of layer 3 security.

Usually with a tunnelling scheme, one IPv4 address is required for every host wishing to connect to the IPv4 Internet. DSTM reduces this constraint by allocating addresses only for the duration of the communication making it possible for several hosts to share the same address on a large time scale.

DSTM can be implemented if a network infrastructure only supports IPv6, but some of the nodes on the network have dual-stack capability (and IPv4-only applications). DSTM consists of three components:

1. An Address Server,
2. A DSTM Gateway or TEP (Tunnel End Point) and
3. A Dual-IP node (called a “DSTM node”) wishing to communicate using IPv4.

For the sake of simplicity, we have decided to present the server and the gateway as different equipment, but in actual deployments, these two functionalities are mostly to be performed by the same host. Figure 3-3 presents the interaction between these three elements.

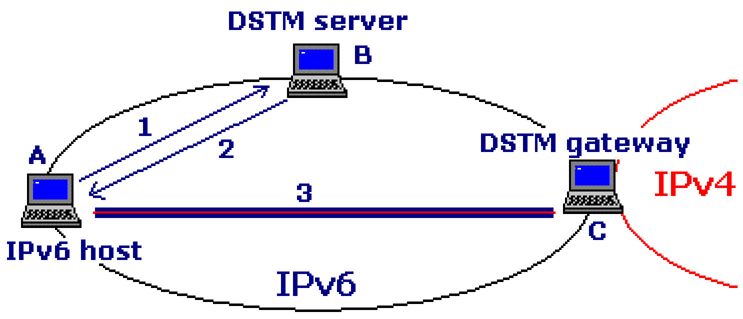


Figure 3-3: DSTM Architecture

As long as communications can take place in native IPv6, none of the capacities of DSTM are required. This applies to protocols like http or smtp, where the use of ALGs (Application Level Gateways) is to be preferred. When a DSTM node detects the need of an IPv4 address, by a query to the DNS resulting in an IPv4 destination address or an application opening an IPv4 socket, the DSTM process is launched.

When the first IPv4 packet needs to be sent, the DSTM client asks the server for an address (1). A number of protocols (DHCPv6 [DHCPv6], TSP [DSTM_TSP], RPC) have already been proposed to perform this task. Native IPv6 transport is the only restriction in this matter.

Following an address request, the server asks the DSTM gateway to add a Tunnel End Point (TEP) for the requesting DSTM node. It is the server who controls the IPv4/IPv6 address mapping performed at the DSTM gateway. Initial versions of DSTM considered that the gateway would build its IPv6/IPv4 mapping table dynamically by observing packet headers, but this approach is now obsolete due to security concerns.

If the end point for the new tunnel is successfully created, following the answering message from the gateway, the DSTM server (who manages an IPv4 address pool) replies to the host with the following information:

- The allocated IPv4 address,
- The period over which the address has been allocated *and*
- IPv4 and IPv6 addresses of the TEP.

This information is used by the node to configure an IPv4-over-IPv6 tunnel towards the DSTM gateway (3). At this point, the DSTM node has IPv4 connectivity and, if it obtained a global IPv4 address, it will be able to connect to any external host.

In DSTM, the period of allocation can be configured based on address availability. Nodes are required to ask for allocation renewal before allocation time expires. Depending on local policy and node behaviour, the DSTM server may accept or deny to extend the allocation. In normal operation, requests for allocation renewal are periodically sent until the address is no longer needed by the host. As long as the address allocation is extended, the DSTM server is not required to update the IPv4/IPv6 mapping table at the gateway. However, when the allocation expires, the gateway must be informed in order to update its tables and allow other nodes to re-use the TEP for that IPv4 address.

The DSTM gateway is in charge of packet forwarding between the IPv6-only domain and IPv4 networks. It performs packet encapsulation/decapsulation using an IPv4/IPv6 mapping table. For successful bi-directional communication, it is very important to allow IPv4 forwarding at the gateway and to make sure that, for any IPv4 packet coming from the outside, the route to DSTM nodes points to the TEP.

DSTM is to be used in a network domain where IPv6 routing is enabled and ALL nodes within that domain are able to communicate using IPv6. In this case, IPv4 support can be turned off. Thus the burden of maintaining an IPv4 addressing plan and supporting IPv4 routing is removed. However, given the huge number of IPv4-only hosts and applications in the Internet, a number of hosts inside IPv6-only domains will still require IPv4 connectivity.

DSTM can be deployed where no other solutions, such as ALGs, can be implemented. DSTM allows Dual IP-layer nodes to obtain an IPv4 address and offers a default route (through a 4over6 tunnel) to an IPv4 gateway. Any IPv4-only application can run over an IPv6-only network if such a scheme is used and, if DSTM is configured to allocate Global IPv4 addresses, hosts inside that domain will be able to communicate with any other host on the Internet.

DSTM may be deployed in several phases. As a first step, IPv4 connectivity may be assured by manually configuring tunnels from Dual-IP nodes to a Tunnel End Point (TEP). In a second phase, when address allocation or tunnel set up protocols become available (DHCPv6, TSP), it would be possible to dynamically assign an IPv4 address to requesting nodes. In this phase, the address may be allocated for the whole lifetime of the requesting node, reducing the complexity of address management. Finally, when IPv4 address availability becomes a problem, DSTM may be configured to allocate addresses only for small periods of time, based on the real needs of requesting hosts.

Since the address allocation process in DSTM is triggered only when IPv4 connectivity is strictly necessary, the size of the IPv4 address pool required by the mechanism should decrease with time (as more hosts and applications become IPv6 aware). However, if the lack of IPv4 address space continues, DSTM may be extended to include the 'ports option' [DSTM_DHCPv6], allowing simultaneous use of the same address by several hosts, but increasing complexity.

3.2.9.1 "The VPN-Scenario"

An alternative use of DSTM concerns what has been called "the VPN scenario" [DSTM_VPN]. It concentrates on the situation where a DSTM node is outside its home domain. Supposing that the node can easily obtain an IPv6 address on the visited network but no IPv4 configuration is possible, the DSTM node can negotiate with its home DSTM server and TEP for IPv4 connectivity. If authentication succeeds and the nomad node obtains an address, the node's IPv4 traffic will be sent to the TEP at its home network using a 4over6 tunnel. Even if the path is not optimal, the node obtains access to private IPv4 resources in its home domain and may obtain global IPv4 connectivity.

3.2.9.2 Security Considerations with DSTM

The DSTM mechanism can use all of the defined security specifications for each functional part of its operation.

E.g. for DNS, the DNS Security Extensions/Update can be used.

Concerning address allocation, when connections are initiated by the DSTM nodes, the risk of Denial of Service attacks (DOS) based on address pool exhaustion is limited in the intranet scenario. With the intranet scenario, if DHCPv6 is deployed, the DHCPv6 Authentication Message can be used for security. When using TSP for address allocation, the SSL encryption and authentication can be used since TSP messages are in plain text.

When exchanging the DSTM options using DHCPv6, the DSTM Global IPv4 Address option may be used by an intruding DHCP server to assign an invalid IPv4-mapped address to a DHCPv6 client in a denial of service attack. The DSTM Tunnel Endpoint option may be used by an intruding DHCP server to configure a DHCPv6 client with an endpoint that would cause the client to route packets through an intruder system. To avoid these security hazards, a DHCPv6 client must use authentication to confirm that it is exchanging the DSTM options with an authorized DHCPv6 server. The DSTM Ports option may be used by an intruding DHCP server to assign an invalid port range to a DHCP client in a denial of service attack. To avoid this security hazard, a DHCP client must use authenticated DHCP to confirm that it is exchanging the DSTM options with an authorized DHCP server.

The main difference between the intranet scenario and the VPN scenario of DSTM is security. In the VPN scenario, DHCPv6 must not be used for address allocation but TSP (tunnel set up protocol) with SSL encryption can be used for this purpose.

In the VPN scenario, the DSTM server must authenticate the outside DSTM client. This authentication cannot rely on the IPv6 address since the address depends on the visiting network but can be based on some shared secret.

In the VPN scenario, the mapping between the IPv4 and the IPv6 address of the DSTM node in the TEP is also a security concern. If the mapping is established dynamically (no configuration by the DSTM server), it could be possible for every intruder knowing a valid temporary IPv4 address to use the TEP as an IPv4 relay or to access internal IPv4 resources. So, in the VPN scenario, the mapping in the TEP must be managed by the DSTM server which authenticates the DSTM host and its IPv6 address. This is an important requirement that avoids the use of IPv4 resources by non authorized nodes.

Finally, for IPv4 communications on DSTM nodes, once the node has an IPv4 address, IPsec can be used since DSTM does not break secure end-to-end communications at any point. The tunnel between the DSTM host and the TEP can be ciphered, but it is our view that this is more of an IPv6 feature (like the use of IPv6 mobility) than a DSTM feature.

3.2.10. OpenVPN-based tunnelling solution

This method is being developed by JOIN within 6NET and will be described in detail in the next iteration of this document.

It is based on the OpenVPN project: http://openvpn.sourceforge.net/. It is basically a Layer 3 tunnel over UDP/IPv4 or TCP/IPv4, so it should prove possible to set up a router that uses the tunnel-interface to route IPv6. It should be capable of traversing NATs and can, if needed, use strong encryption over SSL.

This method is a solution for those calling for a more robust way to offer IPv6 connectivity in a NAT environment.

3.2.10.1 Security Considerations with the OpenVPN scenario

To be included in the next version of this document.

3.3. IPv6 Translation Methods

Translation methods are deployed where IPv6-only devices wish to communicate with IPv4-only devices, or vice-versa and no IPv4-in-IPv6 is used.

3.3.1. SIIT, NAT-PT and NAPT-PT

A translator located in the network layer in the protocol stack is called a "header translator". Such mechanisms translate IPv4 datagram headers into IPv6 datagram headers or vice versa. A model that can be used when implementing this mechanism is presented in RFC 2765 [RFC2765]: "SIIT – Stateless IP/ICMP Translation Algorithm".

Network Address Translation with Protocol Translation (NAT-PT), defined in RFC 2766 [RFC2766], is a service that can be used to translate data sent between IP-heterogeneous nodes. NAT-PT translates an IPv4 datagram into, as far as possible, a semantically equivalent IPv6 datagram or vice versa. For this service to work it has to be at the interconnection point between the IPv4 network and the IPv6 network.

Just like existing NATs in the IPv4 world translate between (usually) private IPv4 addresses and globally routable IPv4 addresses, the NAT part of NAT-PT translates between a globally routable IPv4 addresses to a IPv6 address or vice versa as well as from a private IPv4 address to an IPv6 address. The PT-part of the NAT-PT handles the interpretation and translation of the semantically equivalent IP headers, either from IPv4 to IPv6 or from IPv6 to IPv4. Like NAT, NAT-PT also uses a pool of addresses which it dynamically assigns to the translated datagrams.

Dual-stack and tunnel-based mechanisms do not alter any of the data contained in the IP datagram. This is true both for IPv4 and IPv6 since the communication is end-to-end using only one protocol. NAT-PT (and NAPT-PT as described below) on the other hand translates the header of the datagram from IPv6 to IPv4 or vice versa. The result is a new header which is semantically equivalent to the original header but not equal. It's therefore likely that some of the information has been lost during translation. For example that a service, which is only available in one protocol, is lost when converted to another protocol.

RFC2766 also specifies a service called Network Address Port Translation + Packet Translation (NAPT-PT). This service enables IPvX nodes to communicate transparently using only one IPvY address. NAPT-PT maintains a set of port numbers, which it dynamically assigns to sockets located on the recipient side of the NAPT-PT node.

NAT-PT shares many of the problems the TRT mechanism has, e.g. handling or rather failing to handle IP addresses embedded in application protocol payloads.

3.3.1.1 Security Considerations with NAT-PT

End-to-End Security

As noted in RFC2766, NAT-PT and end-to-end security do not work together. When IPv6 only node X initiates communication to IPv4-only node Y the packets from X have an IPv6 source as well as an IPv6 destination address which are both used in IPsec (AH or ESP) and TCP/UDP/ICMP checksum computations. Since NAT-PT assigns X with an IPv4 address that has no relationship to X's IPv6 address, there is no way for recipient Y to determine X's IPv6 address and in that way verify the checksums.

Prefix Assignment

RFC2766 does not explain how the IPv6 nodes learn about the prefix that is used to route packets o the NAT-PT box. If the prefix is pre-configured in IPv6 nodes, the IPv6 node would prepend the preconfigured prefix do the address of any IPv4-only node with which it wants to initiate communications. However, with a prefix, there might be a reachability problem if the NAT-PT box were to shut down unexpectedly. If an attacker would somehow be able to give the IPv6 node a fake prefix, the attacker would be able to steal all of the node's outbound packets to IPv4 nodes. Even though this is not specified in RFC2766, DNS servers and DNS-ALGs can be used in outgoing connection to return the prefix information to the IPv6 node as a means to avoid the problem of a statically preconfigured prefix. When an IPv6-only node wishes to initiate communications with an IPv4-only node, its resolver would send an AAAA query. This query can be passed through the DNS-ALG which itself looks for an A record. In this case the DNS-ALG can prepend the appropriate prefix for NAT-PT itself and thus return a full AAAA record to the IPv6-only node. The DNS-ALG can also monitor the state of a number of NAT-PT boxes and use only the prefixes of those that are running. The method by which a DNS-ALG determines the state and validity of a NAT-PT box must of course also be secure. The DNS-ALG and each NAT-PT box should be configured with a pairwise unique key that will be used for integrity-protected communications. Note that messages from DNS-ALG are not integrity-protected and can therefore be modified. To prevent such a modification, a DN-ALG can sign its packets. The DNS-ALG's public key can be maide available like that of any other DNS server (see RFC2535) or presented form of a certificate that has a root CA that is well know to all nodes behind NAT-PT. A shared-key technique may not be as practical.

Security Issues Arising when Using a DNS-ALG

A DNS-ALG is required when IPv4-only nodes should be allowed to initiate communication within a NAT-PT scenario. Since the DNS-ALG will translate simple “A record” requests into “AAAA record” requests and vice versa DNS-SEC will not work in this case. However, as pointed out in draft-durand-v6ops-natpt-dns-alg-issues, if the hosts sets the “AD is secure”-bit in the DNS header, it is possible for the local DNS server to verify signatures. Also another option to increase security is for the DNS-ALG to verify the received records, translate them and sign the translated records anew. A third option would be if the host had an IPsec security association with the DNS-ALG to protect DNS records.

Source Address Spoofing Attack

There are two cases in which an attacker will use NAT-PT resources, one where the attacker is in the same stub domain as the NAT-PT box and the second where the attacker is outside the NAT-PT stub domain.

Suppose that an attacker is in the same stub domain as NAT-PT and sends a packet destined for an IPv4-only node on the other side of the NAT-PT-gateway, forging its source address to be an address that topologically would be located inside the stub domain. If the attacker sends many such packets, each with a different source address, then the pool of IPv4 addresses may quickly get used up, resulting in a DoS attack (or rather Address depletion attack). A possible solution to this attack as well as to similar attacks like resource exhaustion or a multicast attack is to perform ingress filtering on the NAT-PT box (which is the border router). This would prevent an attacking node in its stub domain from forging its source address and thus from performng a reflection attack on other nodes in the same stub domain. The NAT-PT box should also drop packets whose IPv6 source address is a multicast address. Address Depletion attacks can be prevented by employing NAT-PT in a way that it translates the TCP/UDP ports of IPv6 nodes into the corresponding TCP/UDP ports of the IPv4 nods/addresses. However, sessions initiates by IPv4 nodes are restricted to one service per server. Of course IPsec might be used to further increase security.

Suppose now that an attacker outside the NAT-PT domain sends a packet destined for an IPv6-only node inside the NAT-PT domain and forges its (IPv4) source address to be an address from the IPv4 address pool used for NAT-PT. The same attacks are then possible as in the scenario above. Again filtering can be used to prevent this. The NAT-PT gateway should drop all packets whose IPv4 source address is a broadcast/multicast address. It should also filter out packets from outside that claim to have a source address from inside the NAT-PT domain.

3.3.2. BIS

The Bump in the Stack (BIS) [RFC2767] (see Figure 3-4) translation mechanism is similar to taking the NAT-PT approach with SIIT and moving it to the OS protocol stack within each host. Unlike SIIT however, it assumes an underlying IPv6 infrastructure. Whereas SIIT is a translation interface between IPv6 and IPv4 networks, BIS is a translation interface between IPv4 applications and the underlying IPv6 network (i.e. the network interface driver). The host stack design is based on that of a dual stack host, with the addition of 3 modules, a translator, an extension name resolver, and an address mapper.

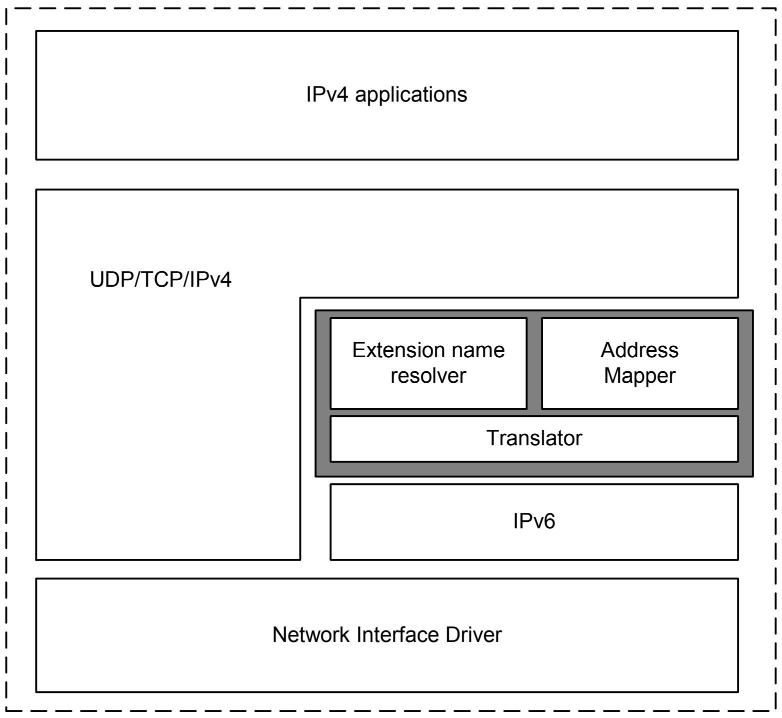


Figure 3-4: The BIS Protocol Stack

The translator rewrites outgoing IPv4 headers into IPv6 headers and incoming IPv6 headers into IPv4 headers (if applicable). It uses the header translation algorithm defined in SIIT. The extension name resolver acts as the DNS-ALG in the NAT-PT mechanism. It snoops IPv4 DNS queries and creates another query asking to resolve both 'A' and 'AAAA' records, sending the returned 'A' record back to the requesting IPv4 application. If only 'AAAA' records are returned, the resolver requests the address mapper to assign an IPv4 address corresponding to the IPv6 address. The address mapper maintains a pool of IPv4 addresses and the associations between IPv4 and IPv6 addresses. It will also assign an address when the translator receives an IPv6 packet from the network for which there is no mapping entry for the source address. Because the IPv4 addresses are never actually transmitted on the network, they do not have to be globally unique and a private address pool can be used.

The BIS mechanism may be useful during initial stages of IPv4 transition to IPv6 when IPv4 applications remain unmodified within IPv6 domains. However, BIS is limited in its translation capabilities. It allows IPv4 to IPv6 host communication but not vice versa. It does not send or receive any IPv4 packets to/from the network. Thus, even an IPv4 application attempting communication with another IPv4 application using BIS, will fail without additional translation mechanisms somewhere in the communication path. As with NAT-PT and SIIT, BIS will not work for multicast communications and will not work for applications that embed IP addresses in their payloads. An ALG is required for any application that exhibits this behaviour.

The BIS method is not widely used.

3.3.3. BIA

The Bump in the API (BIA) [LEE02] translation mechanism is very similar to that of BIS. However, instead of translating between IPv4 and IPv6 headers, BIA inserts an API translator between the socket API and the TCP/IP modules of the host stack (see Figure 3-5).

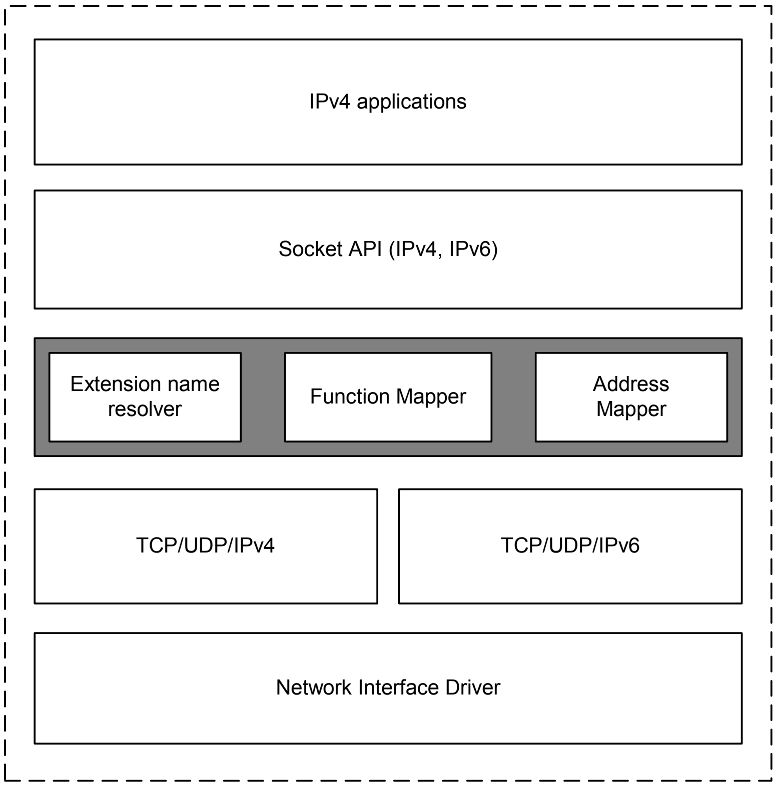


Figure 3-5: The BIA Protocol Stack

Thus, IPv4 socket API functions are translated into IPv6 socket API functions and vice versa. In this way, IPv4/IPv6 translation can be achieved without the overhead of translating every packet header. Like BIS, BIA is based on the addition of 3 modules: the extension name resolver, the function mapper and the address mapper. Both the extension name resolver and the address mapper modules operate in exactly the same way as the corresponding modules in BIS. The function mapper is the entity that maps IPv4 socket calls to IPv6 socket calls and vice versa. The function mapper does this by intercepting IPv4 socket API function calls and invoking corresponding IPv6 socket API function calls in their place. These IPv6 socket function calls are used to communicate with the peer IPv6 host and are transparent to the IPv4 application that invoked the original IPv4 socket function calls.

The BIA mechanism is intended for systems that have an IPv6 stack but contain applications that have not been upgraded to IPv6. Thus, BIA may be useful in early stages of transition when there are many unmodified IPv4 applications within IPv6 domains. BIA allows IPv4 to IPv6 host communication, but does not specify the reverse case. However, it could be easily extended to cater to IPv6 to IPv4 host communication (this is also applicable to BIS). The advantage BIA has over BIS is that it is independent of the network interface driver and does not introduce the overhead of per-packet header translation. However, BIA exhibits similar limitations to BIS. It will not support multicast communication without some additional functionality in the function mapper module, and it will not work for applications that embed IP addresses in their payloads.

The BIA method is not widely used.

3.3.3.1 Security Considerations with BIA

Security issues with BIA mostly correspond to those of NAT-PT (see section 3.3.1.1). The only difference is that with BIA address translation occurs in the API and not the network layer. The advantage here is that, since the mechanism uses the API translator at the socket API level, hosts can utilize the security of the underlying network layer (e.g. IPsec) when they communicate via BIA with IPv6 hosts using IPv4 applications.

Another security issue NAT-PT and BIA have in common stems from the use of address pooling, which may open a denial of service attack vulnerability. One should employ the same sort of protection techniques as mentioned fore NAT-PT in this regard.

Note that since there is no DNS ALG necessary with BIA as it is with NAT-PT, there is no interference with DNSSEC when using this transition mechanism.

3.3.4. Transport Relay

A translator located in the transport layer is called a transport relay. The relay is located somewhere between the communicating nodes and enables IPv6-only hosts to exchange traffic (UDP or TCP) with IPv4-only hosts. If two nodes – for example a client and an application server – use different protocol stacks, they couldn’t communicate directly with each other, and traffic has to be passed over a relay. In case of TCP the relay terminates the IPv6 transport protocol session from the client, and thus acts as a transport destination endpoint to the client. At the same it originates a second IPv4 transport session with the server, and copies received data from each session to the other. In case of UDP the datagram is just translated and forwarded to the target node.

3.3.4.1 TRT

The Transport Relay Translator (TRT) [RFC3142] enables IPv6-only hosts to exchange traffic (TCP or UDP) with IPv4-only hosts. No modification on hosts is required, the TRT system can be very easily installed in existing IPv6 capable networks.

A transport relay translator which runs on a dual-stack node can use one protocol when communicating with the client and one protocol when communication with the application server. In such a setting the relay is able to translate in the transport layer all data sent between the client and application server. For TCP such a translation includes recalculating the checksum, keeping the necessary state information about which client socket is connected to witch server socket and removing this state when the client ends its communication. With UDP the checksum is mandatory when using IPv6 but not when using IPv4.

UDP is a connection-less protocol and in theory it is impossible for a relay to know which UDP datagrams belong to the same session. A UDP relay implementation will typically assume that a UDP datagram that follows another with the same source and destination within a certain time interval, belong to the same session.

The TRT system can work with most of the common Internet applications: HTTP, SMTP, SSH, etc. The transition mechanism operation is relatively simple:

The IPv6 host uses a DNS-ALG as its nameserver to resolve its DNS queries. The IPv6 host, when asking its nameserver for the IPv6 address (AAAA record) of an IPv4-only host, will receive from the DNS-ALG an IPv6 address (AAAA record) specially constructed from the IPv4 address (A record), instead of an error message with the answer that no IPv6 address could be found corresponding to the query. The constructed addresses consist of a special network prefix associated with the transport relay and a host ID (the lower 64 bits) that embeds the IPv4 address of the remote host.

The network is set up such that packets destined for addresses with the special network prefix are routed to the TRT relay node. The TRT then intercepts transport sessions and acts towards the client node as destination endpoint of an IPv6 session and acts towards the server node as source for an IPv4 session, copying all data it receives from each session to the other.

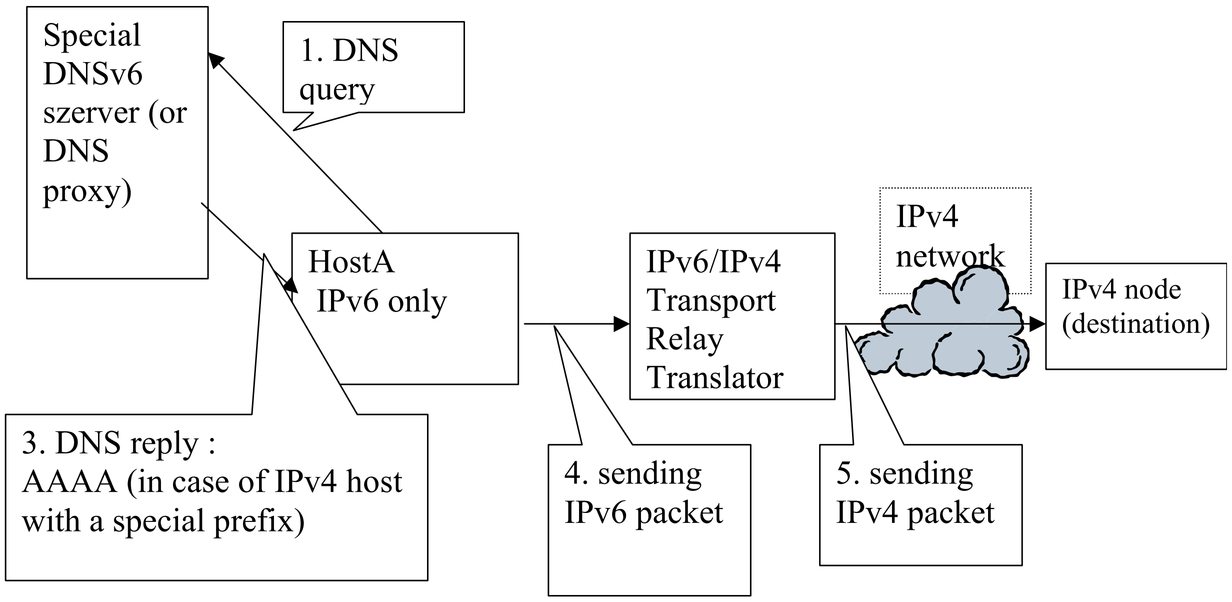

Special DNSv6 szerver (or DNS proxy)
1. DNS query
HostA IPv6 only
IPv6/IPv4 Transport Relay Translator
IPv4 network
IPv4 node (destination)
3. DNS reply : AAAA (in case of IPv4 host with a special prefix)
4. sending IPv6 packet
5. sending IPv4 packet

Figure 3-6: Transport Relay Translator in action

A UDP relay can be implemented in similar manner as a TCP relay. An implementation can recognize a UDP traffic pair like a NAT systems does, by recording address/port pairs into a table and managing table entries with timeouts.

There are both advantages and disadvantages in employing the TRT mechanisms. The advantages include:

- There is no problem with fragmentation. If different fragmentation has to be used in the IPv6 and IPv4 parts of the TRT "connections", there is no problem: the Path MTU discovery algorithm or fragmentation mechanism of the TRT relay server can handle the situation.

- There is no problem with ICMP packets. If any error occurred in any part of the TRT connections the ICMP/ICMPv6 packet is sent back to the TRT relay server, where the error can be handled properly or be reported towards the other end of the "connection".
- It is not necessary to modify the IPv6 stack of the initiating host, neither is it necessary to modify the application to support relaying.
- It is relatively easy to setup.
- It can be enough to have only one TRT relay server for a whole site. And this router has to have only one global IPv4 address.

The disadvantages include:

- There can be problems with applications using embedded IP addresses (e.g. FTP, H.323). The TRT relay has to be smart enough to "look inside" the packets if such an application has to be supported. In this case the TRT relay server becomes a kind of application proxy.
- It supports only unicast TCP/IP traffic, however it is theoretically possible to implement multicast support as well.
- TRT is more difficult to scale than the stateless translation methods. The TRT relay server has to keep track of all the "TRT" connections to properly handle all error conditions. The scaling problem can be eased using anycast technology to reach the closest TRT relay server.
- The TRT relay server can generate a major security problem, since it can be used as an intermediate hop to reach IPv4 servers. The served community of a TRT relay server has to be carefully controlled by packet filtering or access control lists. To reduce the problem site local addresses could be used for accepting incoming IPv6 packets (Note that within the IETF there are currently plans to deprecate site-local addresses and replace them with a new addressing scheme for “private” addressing within a site).
- TRT requires a specially configured DNS server to run.
- Due to the nature of the TCP/UDP relaying service, it is not recommended to use TRT for protocols that use authentication based on source IP address (e.g., rsh/rlogin).
- IPsec cannot be used across a TRT relay.

3.3.5. SOCKS

SOCKS [RFC1928] is another example of a transport relay but it is usually referred to as a "proxy protocol for client/server environments".

A SOCKS proxy works in a similar fashion as a traditional transport relay, but there are minor differences, which we will now describe.

When a client wants to connect to an application server it first sets up a connection to a well known, preconfigured proxy server using a special proxy protocol. The client informs the proxy about the IP address and port number of the application server it wants to communicate with. The proxy server is now responsible to set up a connection to the application server. As soon as this connection is up and running the proxy relays packet between the client and application server hiding the actual connection.

SOCKS include two primary components: a SOCKS server and a SOCKS client library. The server component is located in the application layer while the client component is located between the client application and the transport layer.

Before an application client can use SOCKS it has to be modified ("socksified"). This can be done in two different ways: If the source code is available it can be compiled together with the SOCKS client library using a set of pre-processor directives. If on has instead only precompiled binaries for an application, but the operating system supports dynamic linking of shared libraries one can change some environment variables in the operating system so that the client uses SOCKS instead of the default network libraries.

RFC 3089 [RFC3089] presents a SOCKS-based IPv6/IPv4 gateway mechanism that supports both IPv6 to IPv4 communication and IPv4 to IPv6 communication. This RFC also contains a link to two different implementation of the mechanism described; one from NEC and one from the KAME-project.

3.3.6. Application Layer Gateway (ALG)

An Application Layer Gateway (ALG) is a common mechanism to allow users behind firewalls or behind a NAT gateway to use applications that would otherwise not be allowed to traverse the firewall or NAT gateway. A common example for an ALG is a classical HTTP proxy like “squid” or “wwwoffle”.

The working principle of an ALG can easily be explained using an HTTP proxy as a running example. Normally, a web browser would directly open a connection to a web server if a direct connection between the client and the server can be established. However, when using an ALG, the client opens a connection to the ALG (in this case the HTTP proxy) which (if the required content is not already cached locally from a previous request) then itself establishes a connection to the web-server acting as a relay for outgoing requests and incoming data. In most cases, the use of an ALG is almost transparent for the user. Applications that use ALGs have to be configured to do so beforehand. A web browser has to be configured to use a certain HTTP proxy, for example. There are also applications that allow automatic configuration of ALGs.

In IPv6-only networks, the ALG functionality can be used to enable hosts in IPv6-only subnets to establish connections to services in the IPv4-only world and in some cases the other way around as well. This can be achieved by setting up ALGs on dual-stack hosts which have both IPv6 and IPv4 connectivity.

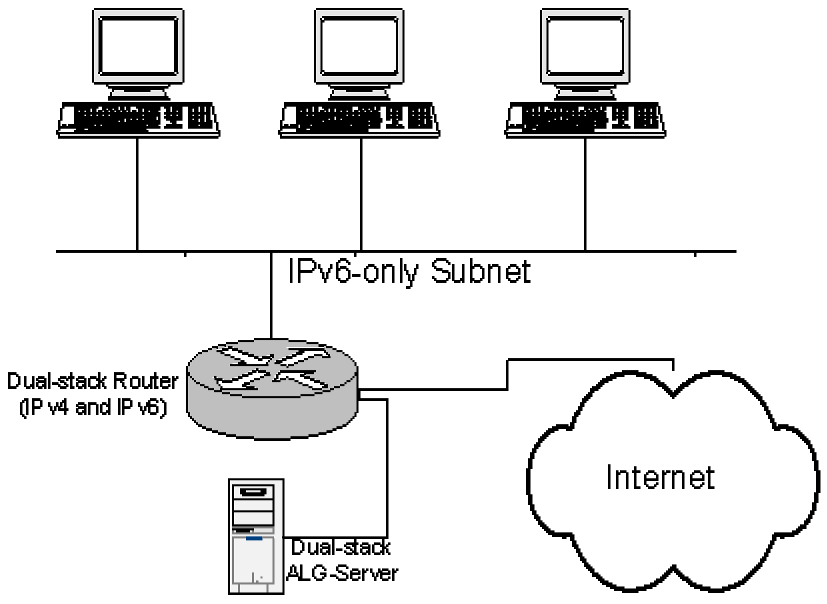


Figure 3-7: ALG Scenario

3.4. 6TALK

6TALK is a transition toolbox providing several functionalities of the mechanisms described above in one implementation. It has been developed by ETRI and, in principle, is based on the NAT-PT/SIIT mechanism and supports some ALG functions, such as a DNS ALG. The first implementation of 6TALK was made in 2001 using Linux with kernel v2.4.8. In 2002 it has migrated to a hardware box with an embedded Linux system on the MPC8260 platform. Regarding the line interfaces, the 6TALK system supports four ports of 10/100 Ethernet interface with one serial line port for internal configuration. Also, a web-based interface and command-line interface have been provided for easy configuration and management. In 2003, it is planned to extend the functionality to support additional features like the tunneling mechanisms 6to4, DSTM and DSTM extension functions.

4. Example Scenarios

This chapter will eventually contain reports on the IPv4 to IPv6 transition as they are taking place at 6NET-partner sites. At present, we cite three examples.

6NET partners authored a scenario-based review of IPv6 transition tools for the IEEE Internet Computing journal [IEEE-V6], published in May/June 2003.

The IETF has been undertaking extensive scenario analysis in the IPv6 Operations WG [V6OPS], in the areas of ISP networks, enterprise networks, unmanaged networks and 3G networks. The enterprise scenario document [ENT-SCEN] (still a draft document) and analysis (just underway) are most relevant to this section.

4.1. Campus IPv6 deployment (University of Münster, Germany)

The University of Münster is a large university with a widespread network that uses a large set of different hardware and network techniques. Thus several considerations had to be taken into account.

If one wants to integrate IPv6 into a network, the most desirable form of integration is always to run dual-stack mode on each and every interface and node. However, while support for IPv6 is now present in most of the products, there are still older hardware and technologies that do not easily support IPv6 capabilities or don't support them at all.

Especially in large sites, that have been in place for a long time, the network infrastructure has evolved over a number of years. Such networks often have a modern core, but still use old technology in some areas and on internal “stubby” edges. In such environments it is practically impossible to run full dual-stack mode. However, several of the transition methods described in this cookbook can be used to reach such areas.

In addition, network administrators often hesitate to introduce IPv6, because they fear to destabilize their IPv4 infrastructure or because they are unfamiliar with IPv6 and IPv6 management. To overcome these fears it is helpful to start with IPv6 just in a few parts of the network and to leave the IPv4 infrastructure untouched.

A good method for this is using VLAN technology (802.1q). VLANs are very common and often used in modern networks, and it is especially easy to integrate IPv6 in these networks. If a dedicated IPv6 router is used, which just participates in a VLAN group, the IPv4 network can remain unchanged, and all IPv6 traffic can be routed and managed over a different set of hardware. If no additional hardware is available, it might be sufficient to use only a small set of the existing routers to do IPv6 routing.

In the University of Münster, there is a single border router (Cisco 7206) with an uplink to the 6WiN (German academic IPv6 network) and the global IPv6 network. The same router is internally connected via a trunked Gigabit Ethernet interface to a Catalyst 6500 and the university’s Layer 2 infrastructure.

The Cisco 7206 now acts as an IPv6 router. It is a simple task to add a VLAN interface with a VLAN-ID of the already existing IPv4 VLANs. Via this interface our IPv6 router sends router advertisements to the VLAN clients and becomes the default router for IPv6 traffic. The standard IPv4 traffic still is routed over the default IPv4-only routers.

Since VLANs are spread throughout the whole university, it is possible to give IPv6 access to various areas. Still, there are some drawbacks. In those areas where no VLAN technology is available, but older remnants of, for example, ATM or FDDI infrastructure exist, other methods are needed to give IPv6 access to the hosts. This can be achieved with various tunnel technologies, but these are not yet deployed in our network. Also, if there is a "secured" area, one should consider carefully if IPv6 access should be added to such a VLAN, because when bypassing the IPv4 infrastructure those security mechanisms might get compromised. For example if there are ACL rules existent for IPv4, these should be applied also for IPv6. This is not always possible, because the two routing topologies are different. While the IPv4 network usually consists of a set of routers building a cascading/tree-like core network, our IPv6 "network" here is just a single router. If IPv4 ACL rules rely on some kind of hierarchical routing infrastructure, they probably cannot be rebuilt for IPv6 in this case, or at least not easily so.

Using VLANs is a smart and easy way to start IPv6 deployment, yet it is far away from a real production environment. The step to full dual-stack still takes a while to implement. Meanwhile the current setup can be improved; one of the next steps would be to add a set of two Routers (C7200 or Cat6500) to every VLAN, which offer redundancy for routing and addressing of clients. Just sending Router Advertisement could be replaced with DHCPv6 to assign addresses to hosts. Extending the number of IPv6 routers will require the setup of an internal routing protocol (e.g. IS-IS or OSPFv3).

Several tunnel technologies have to be deployed internally to reach remote IPv6 subnets (e.g. with configured tunnels or 6to4) or single IPv6 hosts (e.g. with ISATAP, configured tunnels, 6to4 or a tunnel broker). These technologies can also be used to reach e.g. Wireless LAN subnets and single Dial-In users.

These steps will be taken throughout the year 2004 and the experiences will be included in the next interim and final versions of this cookbook.

4.2. Small academic department, IPv6-only (Tromso, Norway)

This scenario is already described in Deliverable D2.3.1.

4.3. Large academic department scenario (University of Southampton)

In this section we begin by describing the systems components in this scenario of a large departmental network (1,500+ users, up to 1,000) hosts that wishes to transition to deploy IPv6. We describe the elements that need to be considered for the transition.

This scenario description assumes no IPv6 is deployed, although in reality the transition is underway; thus after a review of the systems components, we present an overview of the status to date, current plans, and also the major remaining obstacles that have been identified.

4.3.1. Systems Components

The components fall into categories:

- Network components

- Services
- Host and device platforms
- User tools

We discuss these categories below.

In the light of the IETF v6ops WG activity on studying IPv6 network renumbering [RENUMBER], we also cite components where hard-coded IP(v4) addresses may be found, that may need consideration in IPv6 networks (this is an incomplete list that will be updated in future releases).

4.3.1.1 Network

Physical connectivity (Layer 2)

The technologies used are:

- Switched Ethernet
- Gigabit Ethernet
- Wireless networking (802.11b)

There is no use of ATM, FDDI or other "older" technologies. The network is purely an Ethernet one. VLANs are supported by the network equipment.

Routing and Logical subnets (Layer 3)

The hybrid Layer 2/3 routing equipment is Alactel OSR and Omnicore L2/L3, with approximately 15 internal IPv4 subnets (in effect, routed VLANs). There is no specific internal routing protocol used. There is a static route via the site firewall to the main upstream provider (academic) running at 1Gbit/s, and there is also a static route to the secondary (low bandwidth) link off-site (commercial).

Hard coded IP information:

- *The IPv4 address space assigned by academic provider*
- *There is hard-coded IP subnet information*
- *IP addresses for static route targets*

Firewall

The firewall is currently CheckPoint Firewall-1 running on a Sun Solaris platform, just migrating to a Nokia IP740 hardware platform. There is one internal facing interface, one external facing interface, and two "DMZ" interfaces, one for wired hosts and one for the Wireless LAN provision.

Hard coded IP information:

- *Names resolved to IP addresse sin FW-1 at "compilation" time*
- *IP addresses in remote firewalls allowing access to remote services*
- *IP-based authentication in remote systems allowing access to online bibliographic resources*

IDS

The Snort package is used for intrusion detection.

Management

Some network management is performend by SNMP; there is no specific package for this. There is a greater emphasis on monitoring than explictly in management.

Monitoring

A number of tools are used, to monitor network usage as well as systems availability, e.g. nocol, nagios and MRTG. The IBM AWM tool is used for network testing, along with iperf, rude and crude.

Remote Access

The components supporting remote access are:

- Livingston Portmaster 56K/ISDN dialup
- RADIUS server
- (Microsoft) VPN server

IPv6 access (e.g. for local testbed)

Due to lack of native IPv6 services from the regional network (LeNSE), a static IPv6-in-IPv4 tunnel is required to the JANET (NREN) IPv6 service.

Hard coded IP information:

- *IP address of both tunnel end points*

4.3.1.2 Services

The component services hosted by the departmental network are:

Email

There are three MX hosts for inbound email, and two main internal mail servers. Sendmail is the MTA. POP and IMAP (and their secure versions) are used for mail access, using the UW-IMAP open source code. There is an MS Exchange server used by up to 100 users (generally those wanting shared access to mail spools, e.g. professors and secretaries).

MailScanner is used for anti-spam/anti-virus. This uses external services including various RBLs for part of its spam checking.

Successful reverse DNS lookup is required for sendmail to accept internal SMTP connections for delivery.

Hard coded IP information:

- *Blocked SMTP servers (spam sources)*

Web hosting

Web content hosting is provided either with Apache 1.3.x (open source) or Microsoft IIS 5.0. Common components used to build systems with are MySQL, PHP 4 and Perl 5; these enable local tools such as Wikis to be run.

Hard coded IP information:

- *.htaccess and remote access controls*
- *Apache "Listen" directive on given IP address*

Databases

All database systems are presented via a web interface, including the financial systems. In some cases, e.g. student records, ODBC-like access is required/used in to/out from the department systems to the campus systems. Databases include: finance records, people, projects and publications (offered using ePrints).

Directory services

A number of directory service tools are in use:

- NIS (6 servers, all Solaris)
- LDAP
- Active Directory
- RADIUS

DNS

The three DNS servers have recently been upgraded to BIND9. A DNS secondary is held at another UK university site.

PKI

The department has at least 10 SSL certificates from Thawte, including Web-signing certificates. No personal certificates are supported by the department (though users may have their own).

NTP

The JANET NREN offers a stratum 0 NTP server. The department also has a GPS-based NTP server built-in to its own RIPE NCC test traffic server.

USENET news

The news feed is delivered using dnews.

Multicast

There is PIM-SM IPv4 multicast via a dedicated Cisco 7206 router. This supports applications including the IPv4 AccessGrid conferencing system. A number of bugs in the Alcatel equipment prevent heavy use of IPv4 Multicast within the department network (thus an IPv6 Multicast solution is highly desirable). An IPv4 Multicast beacon is used for monitoring Multicast.

Remote login

Remote login access is offered via ssh, with sftp for file transfer. Remote use of telnet and ftp is denied by the firewall.

File serving

The main file servers are SGI systems, hosting large (multi-TB) standalone RAID arrays. The files are offered via NFS and Samba to client systems.

The content distribution server is hosted on such a system (e.g. containing MS software licenced under the Campus Agreement).

4.3.1.3 Host and device platforms

Server platforms

The following server platforms are in use in the department:

- Windows 2003 server
- Windows 2000 server
- Windows NT
- Solaris 8
- Solaris 9
- RedHat Linux
- SGI Origin 300 (Irix 6.5.x)

Desktop/laptop platforms

The following client platforms are in use in the department:

- Windows 98, 2000, ME, XP
- Linux (various flavours)
- MacOS/X
- BSD (various flavours)

PDA platforms

The following PDA platforms are in use in the department:

- Windows CE/.NET
- PalmOS

- Familiar Linux
- Zaurus

4.3.1.4 User tools/systems

The following tools or systems are used by the department's user base.

Hardware

Various dedicated systems, for example:

- Networked printers
- Networked webcams

Mail client

Various, including:

- Outlook (various versions)
- Eudora
- Mutt
- Pine

Web browser

Various, including:

- MS Internet Explorer
- Mozilla
- Safari
- Opera

Conferencing systems

The following conferencing tools are in regular use:

- AccessGrid
- A dedicated H.323 system
- MS Netmeeting

Other collaboration tools

Collaboration tools in regular use include:

- IRC
- Jabber
- MSN Messenger

- cvs

USENET news client

Various, including:

- nn
- Mozilla

Host communications

Specific tools for remote host communications include:

- X11
- VNC
- PC Anywhere

4.3.2. Transition status

Having described the components, we now outline the steps already taken towards transition at the scenario site. The focus here is to provide increasing IPv6 functionality in a dual-stack environment, with the goal of allowing IPv6-only devices to be introduced.

Because the Alcatel switch/router equipment does not route IPv6, an alternative method was required to deliver IPv6 on the wire to existing IPv4 subnets. To enable this, IPv6 router advertisements were delivered using an IPv6 router supporting VLAN tagging; this BSD router is able to inject a different IPv6 prefix onto each IPv4 subnet, using congruent VLANs. This VLAN method is described by the authors in [VLAN-ID]. As traffic in the site grows, multiple routers can be dedicated to this task for internal routing, or a router with multiple interfaces. We currently have four routers with quad Fast Ethernet interfaces. BSD allows multiple-VLAN tagging per interface, so in light traffic conditions the interface count can be collapsed.

External IPv6 connectivity was acquired using a Cisco 7206, for unicast and multicast (SSM and PIM-SM), with IPv6 Multicast routed internally onto the VLANs using the BSD IPv6 Multicast support. The connection to the JANET IPv6 service is via an IPv6-in-IPv4 tunnel until the local regional network is IPv6-enabled (expected Summer 2004). JANET allocated us 2001:630:d0::/48.

The longer-term plan is to use IPv6 firewalling on the Nokia IP740; until then the firewall is an additional BSD system, on which ports are blocked by default. This is a partially stateful firewall.

Two IPv6-only DNS servers have been run in the past; now the main servers network is IPv6-enabled the department's primary BIND9 DNS servers are IPv6-enabled. This includes reverse delegation of our prefix under 0.d.0.0.0.3.6.0.1.0.0.2.ip6.int and 0.d.0.0.0.3.6.0.1.0.0.2.ip6.arpa (the .int is being phased out).

The main Linux login server is IPv6-enabled, with ssh logins and sftp file transfer available through the firewall. Once IPv6 is present on the wire, all that is needed is the firewall hole to be opened up, an IPv6 AAAA DNS entry added for the login server, and the sshd daemon with IPv6 support turned on. Offering only secure protocols (and not plain ftp or telnet) can be easier to do when starting afresh with a new protocol.

Some web sites have been made available using Apache 2, e.g. the IST IPv6 Cluster site, as operated for the 6LINK project.

The department's Wireless LAN (over 30 access points) is IPv6 enabled. Some Mobile IPv6 has been deployed and tested.

A dual-stack Jabber server is deployed.

An H.323 IPv6 conferencing system has been tested (GnomeMeeting for Linux).

4.3.3. Next steps for the transition

There is a lot to be done. Some imminent next steps include the following:

- A dedicated NTP device with IPv6 capability is being added. Also, the RIPE NCC test traffic server can act as an IPv6 NTP server. Thus two GPS-sourced IPv6-capable NTP servers will soon be online.
- The Sendmail configuration for SMTP handling will be migrated to support IPv6.
- The main department web server will be upgraded to Apache 2 as part of the main redesign already happening this summer. There is an IPv6-enabled e-Prints server already available.
- The IPv6-enabled Snort will be deployed on our IPv6 uplink. We will also provide reporting on traffic being blocked by the IPv6 firewall to see what kinds of probes are being performed on a typical IPv6 deployment.
- We will be pushing for a native IPv6 service on LeNSE, our regional network.

4.3.4. IPv6 Transition Missing Requirements

In the study for transition, we have identified a number of missing (unavailable) components for IPv6 transition, including:

1. No IPv6 Layer 3 functionality on the Alcatel OSR/Omnicore equipment (this will be worked around using the parallel VLAN method, until new IPv6-capable equipment is deployed);
2. Lack of NFS/Samba IPv6 support;
3. Lack of MS Exchange, Outlook or Eudora IPv6 support;
4. AccessGrid is IPv4-only (IPv6-enabling work is to be undertaken in 6NET);
5. Some Apache 2 modules lack Apache 1.3 functionality, hence migrating is a problem in a small number of cases;
6. No IPv6 support for Active Directory;
7. No IPv6 dnews, so one would have to use inn as a Usenet news server;
8. Lack of supported IPv6 for Windows 98/2000/ME;
9. Lack of supported IPv6 for Irix;
10. Lack of supporte IPv6 for various PDA platforms;
11. No method available to offer reverse IPv6 DNS for sendmail to verify autoconfiguring hosts (prepopulating a 64 bit subnet space is a problem, some wildcard method is required);
12. Lack of MLDv2 snooping in Ethernet switch equipment (thus IPv6 Multicast will flood subnets);

13. No available IPv6-enabled X11 (there is an xfree but it is encumbered by an unpopular copyright statement that most distributors find unacceptable);

This list will be updated in future revisions.

4.4. Other scenarios

We plan to add other scenarios in the next iteration, including:

- Scattered dual-stack devices in a university, with or without a dual-stack router on campus (early user experimentation), and/or a small testbed in a university, possibly IPv6 only (contained network);
- Some schools may get a single IPv4 address + NAT for broadband access to the school limiting the set of applications they could deploy. Adding IPv6 + Global addressing opens the way to run many applications;
- Home user networks (staff, students) wanting connectivity, possibly behind a NAT, possibly single machine, possibly dynamic IPv4 address.

The latter case may be a good example for use of the JOIN IPv6 OpenVPN access method.

4.5. Summary of unexpected results and unforeseen difficulties

The next iteration of this document will include a section on key lessons learnt from the transition experiences. This will be a focus also of the joint transition case studies with the Euro6IX project (including the UMU and UPM universities).

4.6. Discussion of tradeoffs made in solutions chosen

The next iteration of this document will include a section on tradeoffs made in solution selection based on the transition experiences. This will be a focus also of the joint transition case studies with the Euro6IX project (including the UMU and UPM universities).

This will include issues such as whether methods should bypass site administrator knowledge; i.e. how the administrators can perceive demand when tunnel methods are happening without their knowledge.

6NET has contributed a number of Internet Drafts in this general topic of analysis of tradeoffs and considerations of architectures, namely [STEP], [TRANSARCH], [TUNNEVAL], [CONSIDER].

5. Configuration Examples: Dual-Stack

This Chapter, as well as the following two, should be the main source of information for people reading this cookbook as it includes installation and configuration examples for most of the implementations for the tools and mechanisms described in Chapter 3. There are no more explanations about the theoretic functionality of the tools to be found here.

Generally, adding IPv6 functionality to existing IPv4 interfaces – if it is supported in the operating system/on the platform – is often a simple task. Nevertheless, as one basically adds a complete new IPv6 network living next to the existing IPv4 network, one has to think about internal and maybe external routing structures. These may differ from those used in IPv4, as not all of the current IGPs (like OSPFv2) are capable to route IPv6 traffic. Instead it may happen that new protocols (e.g. OSPFv3) need to be deployed.

Information on how to switch on IPv6 on any platform is included in appendix B (chapter 10) of this document. As IPv6 becomes more and more common it is likely that this information will become unnecessary. Very soon all operating systems or router platforms will have IPv6 switched on by default.

Configuration examples for routing protocols are covered within Deliverable D3.1.2 of work package 3 in the 6NET project. We'd like the reader to refer to that document to learn how to set up inter- or intra-domain-routing with either iBGP, OSPFv3, IS-IS or RIPng for most platforms.

5.1. Dual-stack VLANs

Using VLANs for IPv6 site deployment is not really a transition tool but it is a method that is becoming more commonly used for dual stack IPv6 deployment in site networks because it is so easy if the network already makes use of the VLAN technique. One just injects (new) IPv6 subnets/links into the existing IPv4 VLANs. In a smaller size network, there may be 10-15 VLANs, each just carrying traffic for one IPv4 subnet. In such situations, one can take advantage of the VLAN segregation of traffic to connect IPv6 routers such that a single IPv6 prefix (a /64 prefix) is injected into each IPv4 VLAN that requires dual-stack networking. Hosts in such subnets can then autoconfigure IPv6 addresses in addition to their IPv4 addresses or make use of DHCPv6 if the service is implemented. In this case, the IPv6 routing hierarchy may be IPv6-only, and exist in parallel to the IPv4 network infrastructure. The IPv6 link off-site may also be separate.

It is important in such an instance that IPv6 filters and access control lists are deployed to prevent IPv6 being a back door for hackers to enter the (IPv4) network.

See [VLAN-ID], an IETF I-D produced within the 6NET project, for more details.

5.1.1. Configuring an interface on a Linux host to become part of a VLAN

The following example shows how a physical interface of a linux host (i.e. eth0) is trunked to become part of a tagged VLAN. The only prerequisite is that support for VLANs needs to be switched on before compiling the kernel.

```
# ip link set <trunked interface> up;

# /usr/sbin/vconfig add <trunked interface> <VLAN-ID>

# ip link set <trunked interface>.<VLAN-ID>
```

6. Configuration Examples: IPv6 Tunneling Mechanisms

6.1. Manually Configured Tunnels

The necessary configuration to manually configure IPv6-to-IPv4 tunnels in different operating systems and router platforms is presented in the following subsections.

6.1.1. Cisco IOS platform

Manually configuring an IPv6-in-IPv4 tunnel on a Cisco IOS platform is not much different from setting up an IP-over-IP tunnel. One creates the interface by simply changing to the configuration mode by typing:

```
# configure terminal
```

There one can create the interface:

```
(config)# interface Tunnel 0
```

Note that the number of the Tunnel can be any number from 0 up to about 65000. To configure the interface with an IPv6 address on has two possibilities:

```
(config-if)# ipv6 address <full ipv6-address>/<subnet-length>
```

or

```
(config-if)# ipv6 address <prefix>/<prefix-length> eui-64
```

The first possibility will result in the interface being configured with the exact address one has specified. Note that the length of the subnet can be set as 128. Using the second possibility one specifies a prefix, which may be up to 64 bits long. The full IPv6 address the interface will then be configured with includes (for example) the MAC address of the hardware in the interface identifier as specified in the EUI-64 standard.

The tunnel source can either be specified with the name of the IPv4 source or by directly stating the IPv4 address of the local tunnel endpoint, e.g:

```
(config-if)# tunnel source 128.176.191.82
```

```
or
```

```
(config-if)# tunnel source FastEthernet0/0
```

The tunnel destination is simply set up by:

```
(config-if)# tunnel destination <IPv4 address of  remote tunnel endpoint>
```

Finally one has to set the tunnel mode to "ipv6ip" to specify the correct encapsulation and decapsulation.

```
(config-if)# tunnel mode ipv6ip
```

With the "`ipv6 route`" command one can configure routes for tunnel interfaces just like with any other interface.

6.1.2. Juniper (JunOS)

If you have Tunnel PIC installed in your Juniper router, you can configure IPv6 over IPv4 tunnels. To do this, you configure a unicast tunnel across an existing IPv4 network infrastructure.

Configuration example:

Router #1:

```
[edit]
interfaces{
      gr-1/0/0{
            unit0{
                  tunnel{
                        source 128.176.191.82;
                        destination 128.176.184.7;
                  }
                  family inet6{
                        address 3ffe:400:ffff::ffff:1/112
                  }
            }
      }
}
```

Router #2:

```
[edit]
interfaces{
```

```
            gr-1/0/0{
                  unit0{
                        tunnel{
                              source 128.176.184.7;
                              destination 128.176.191.82;
                        }
                        family inet6{
                              address 3ffe:400:ffff::ffff:2/112
                        }
                  }
            }
      }
```

Configured tunnels require IPv6 routing. For that purpose one can either create static routes r add the tunnel interface to OSPFv3 or RIPng.

6.1.3. Extreme (ExtremeWare IPv6)

You can configure a tunnel between a dual stack host and a dual stack router or between two dual stack routers.

IPv6-in-IPv4 tunnelling requires an active IPv4 interface on the switch. The address of this interface is used for the configuration of the tunnel. If that particular IPv4 interface goes down, tunnelling fails even if other IPv4 interfaces are available on the switch.

To avoid this situation, we create an IPv4 VLAN with loopback enabled. Even if all of the ports on the VLAN go down, the VLAN stays up and so does the tunnel. At the cost of an additional VLAN on each IPv6-in-IPv4 router and advertisement of the additional IPv4 address, you ensure greater tunnel stability.

Configuration of Switch A:

```
create vlan to_router_b
configure vlan to_router_b add ports 1-10
configure vlan to_router_b ipaddress 128.176.191.82/24
enable loopback-mode to_router_b
create tunnel 6in4_to_b ipv6-in-ipv4 destination 128.176.184.7 \
                                                      source 128.176.191.82
configure tunnel 6in4_to_b ipaddress ipv6 3ffe:400:ffff::ffff:1/112
enable ipforwarding ipv6 6in4_to_b
```

Configuration of Switch B:

```
create vlan to_router_a
configure vlan to_router_a add ports 1-5
configure vlan to_router_a ipaddress 128.176.184.7/24
enable loopback-mode to_router_a
create tunnel 6in4_to_a ipv6-in-ipv4 destination 128.176.191.82 \
                                                 source 128.176.184.7
configure tunnel 6in4_to_a ipaddress ipv6 3ffe:400:ffff::ffff:2/112
enable ipforwarding ipv6 6in4_to_a
```

6.1.4. 6WIND (SixOS)

While essentially being nothing more than a PC running some version of FreeBSD, 6WIND routers come with their own command-line interface (CLI). For the purpose of this example we assume one has just logged on to the rooter as user "admin".

1. First one has to go into the configuration mode for the running configuration:

```
sixwind{} edit running
```

2. Change to the "migration context":

```
sixwind{running} mig
```

3. Setup the IPv6-in-IPv4 tunnel:

```
sixwind{running-mig} 6in4 <tun-#> <local v4> <remote v4> <local v6> \
                                                          <remote v6>
```

As an actual example:

```
sixwind{running-mig} 6in4 0 128.1.2.3 68.2.3.4 3ffe:pTLA:y::xxx:2 \
                                                  3ffe:pTLA:y::xxx:1
```

With this configuration the 6WIND router will create two /128 routes for both of the IPv6 addresses of the tunnel endpoints.

4. Leave the “migration context”:

```
sixwind{running-mig} exit
```

We assume that the tunnel is going to be the only IPv6 connection for the 6WIND router (e.g., if it is the IPv6 access router for the site). Therefore the IPv6 default route needs to be set to this tunnel.

5. Change to the “routing context”:

```
sixwind{running} rtg
```

6. Configure the IPv6 default route:

```
sixwind{running-rtg} ipv6_defaultroute <local v6 IP>
```

In our example:

```
sixwind{running-rtg} ipv6_defaultroute 3ffe:pTLA:y::xxx:1
```

7. Leave the “routing context”:

```
sixwind{running-rtg} exit
```

8. Apply the configured changes to the running configuration:

```
sixwind{running} addrunning
```

9. Exit the configuration mode:

```
sixwind{running} exit
```

Note: If you have performed step 8 immediately before typing in “exit” here everything should be fine. Should you have configured other changes after step 8 before exiting the configuration context the router will ask you if you want to apply these changes to the running configuration as well. Type in “y” if you want the changes to be added to the running configuration, otherwise answer “n”, in which case all changes after the last “addrunning”-command will be lost.

10. Save changes to the startup configuration so the tunnel will still be configured after the next reboot:

```
sixwind{} copy conf running start
```

This is it.

6.1.5. Windows XP

Configuring static tunnels on Windows XP hosts can be performed by the following steps:

1. Create an IPv6-in-IPv4 tunnel, named *myTunnel*:

```
# netsh interface ipv6 add v6v4tunnel myTunnel 195.251.29.15 \
                                                195.251.29.243 enable
```

2. Add an IPv6 address to the tunnel:

```
# netsh interface ipv6 add address “myTunnel” 3ffe:2d00:1::1
```

3. Add a default route to the remote IPv6 address of the tunnel, e.g. 3ffe:2d00:1::2, so that all IPv6 traffic goes through the tunnel:

```
# netsh interface ipv6 add route ::/0 “myTunnel” 3ffe:2d00:1::2
```

6.1.6. Windows 2000

Microsoft IPv6 Technology Preview software should initially be installed to the Windows 2000 host (See Appendix B). After rebooting the system, three IPv6 enabled interfaces are created automatically. The “Tunnel Pseudo-Interface” can be use for creating static tunnels between to hosts, as follows:

1. Assign an IPv6 address to tunnel interface:

```
# C:>ipv6 adu 2/3ffe:2d00:1::1
```

2. Create a static route entry that it points to remote IPv4 address of the tunnel:

```
# C:>ipv6 rtu ::/0  2/::194.177.210.38
```

6.1.7. Linux

The easiest way to set up a tunnel on a Linux host is using the "ip"-command. Modern distributions usually include IPv6 functionality not only with this command but in general as well.

To set up an IPv6-in-IPv4 tunnel usually a "sit" interface is created:

```
# ip tunnel add sit1 remote <IPv4 address of remote tunnel endpoint> \
                                                 local <local IPv4 address>
```

Note that "sit1" is the name of the sit interface. The "local IPv4 address" is the address of the network interface to be used for the incoming IPv4 traffic which contains the encapsulated IPv6 datagrams.

After configuring the interface it has to be brought up:

```
# ip link set sit1 up
```

To equip interface sit1 with an IPv6 address other than the autoconfigured link-local address one can use the following command:

```
# ip add addr <IPv6 address>/<subnet-length> dev sit1
```

After these commands the output from "`ifconfig sit1`" should look somewhat like:

```
sit1  Link encap: IPv6-in-IPv4
      inet6 addr: 3ffe:401:1::fff0:2/112 Scope:Global
      inet6 addr: fe80::80b0:b807/128 Scope:Link
      UP POINTOPOINTRUNNING NOARP    MTU:1480    Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:0 errors: 0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
```

Creating tunnels and adding IPv6 addresses can also be achieved with the command "ifconfig", but this is an old-fashioned style and should not be used in newer Linux distributions.

6.1.8. Solaris 8

The steps to manually configure a tunnel interface on a Solaris 8 workstation are the following:

1. Create the file /etc/named/hostname6.ip.tun0, which is executed every time the workstation bootstraps. Each line of the file is used as input by the *ifconfig* command, and thus, the appropriate command syntax is required.
2. The first line in the /etc/named/hostname6.ip.tun0 file contains the IPv4 source and the destination addresses of the tunnel. For example, if the source and destination addresses are 150.140.21.45 and 194.177.63.238, respectively, the first line should be:

```
tsrc 150.140.21.45 tdst 194.177.63.238
```

3. The second line in the /etc/named/hostname6.ip.tun0 file configures the IPv6 addresses of the tunnel endpoints. The first IPv6 address belongs to the local host and the second to the peer host. For example:

```
addif 3FFE:2D00:1::1 3FFE:3D00:1::2 up
```

4. Restart the script /etc/init.d/inetinit
5. Set the IPv6 default route to the remote tunnel endpoint (if the tunnel is the only IPv6-connection of the host:

```
# add -inet6 default 3FFE:3D00:1::
```

6.1.9. FreeBSD, NetBSD and Darwin/Mac OS X

Tunnels are configured via the "`ifconfig`" command with little to no variation on all operating systems that have incorporated the KAME project's IPv6 stack, including FreeBSD, NetBSD and Darwin/Mac OS X (OpenBSD could most likely be added to this list as the same instructions would most likely work on it, too, but this has not yet been verified by the authors).

The next installation steps should be followed in case of using `ifconfig`:

0. Check whether a "gif" interface exists using ifconfig. Result is either:

```
# ifconfig gif0
ifconfig: interface gif0 does not exist
```

or

```
# ifconfig gif0
  gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
```

1. Create the interface:

```
# ifconfig gif0 create
```

and audit its state:

```
# ifconfig gif0
  gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
```

This method works in FreeBSD and NetBSD, but not on Mac OS X even though it is documented in the manual pages for ifconfig on Darwin 6.8/Mac OS X 10.2.8 (aka Jaguar) as well as on Darwin 7.2.0/Mac OS X 10.3.2 (aka Panther) and in the usage information printed by the ifconfig command itself. However, Mac OS X comes standard with gif0 created so if only a single tunnel is needed this poses no problem.

2. Assign the two (remote and local) IPv4 addresses of the tunnel endpoints, e.g.:

```
# ifconfig gif0 tunnel 130.1.2.3 134.5.6.7
```

3. Assign the local IPv6 addresses of the tunnel:

```
# ifconfig gif0 inet6 alias <Local_IPv6_TUNNEL_ ADDR> \
                      prefixlen <Length of prefix for tunnel address>
```

For example:

```
# ifconfig gif0 inet6 alias 2001:DEAD:BEAF:ABBA::1 prefixlen 64
```

Note that the "`alias`" keyword is not necessary on all BSD OSes, but it never hurts to include it. The prefix-length may be omitted when it its equal to 64 since 64 is the default. When the user has only one or a few tunnels, it is recommended to use a 64 –bit prefix-length as that will typically also be used for other (less virtual) interfaces. However, longer (more specific) prefix-lengths are not a problem and may be an attractive chose for tunnel-brokers hosting many tunnels from a limited address space. It is wise to avoid prefixes longer than 126 bits though as some implementations have (had) problems with 127 and 128 bit long prefixes, even though the author is not currently aware of any such problems in recent releases of an operating system. Still, taking a prefix of 120 bit, 126 bit or even 112 bit length hardly wastes address space and my easily be worth it to avoid potential trouble, however unlikely the problem may be.

In the above commands it is also possible to specify both source and destination IPv6 address of the tunnel but it is not necessary as the routing subsystem does not need to know the remote end's global address (it is enough to be able to talk to the next hop using link-local addressing). For administrative reasons it may be attractive to assign the destination address, too, though. Some non KAME derived implementations even require the global IPv6 addresses of both tunnel endpoints to be set. This is accomplished by substituting the above command with for example:

```
            # ifconfig gif0 inet6 alias 2001:DEAD:BEAF:ABBA::1 \
                                                  2001:DEAD:BEAF:ABBA::2 prefixlen 128
```

Actually, as routing over an interface can work without assigning global addresses at all, it should not be necessary to assign any of them. Indeed, KAME derived IPv6 stacks do not require global IPv6 addresses to be assigned to a tunnel interface (as tested by the author with NetBSD and FreeBSD 3.x). In that case, one needs to explicitly add a route over the tunnel interface to the link-local address of the remote tunnel end (or run a routing protocol that will figure it out for itself using the well-known multicast address). For example, to add a default route over a tunnel interface to a remote end with link-local address fe80::2, one would add a route like this:

```
            # route add -inet6 default fe80::2%gif0
```

or

```
            # route add -inet6 default -ifp gif0
```

Such unnumbered tunnels will typically only make sense when both tunnel endpoint use a KAME derived IPv6 stack as other implantations usually don't support unnumbered tunnels.

6.1.9.1 Alternative with FreeBSD

As an alternative to the above way of creating a manual IPv6-in-IPv4 tunnel, Free BSD also offers the possibility to set up a tunnel by only modifying the contents of /etc/rc.conf file such that the tunnel is automatically created and configured at boot time. Add the following lines to the rc.conf file:

```
      gif_interfaces="gif0"
      gifconfig_gif0="130.1.2.3 134.5.6.7" \
                                               ipv6_ifconfig_gif0="2001:DEAD:BEAF:ABBA::1 \
                                                      2001:DEAD:BEAF:ABBA::2 prefixlen 128"
```

6.1.9.2 Alternative with NetBSD

With NetBSD, one can easily configure a tunnel interface by making a corresponding `/etc/ifconfig.<ifn>` file, where `<ifn>` is the name if the interface (i.e. `gif0`).

For the example above one would create a file `/etc/ifconfig.gif0` containing the following lines:

```
      create
      tunnel 130.1.2.3 134.5.6.7
      inet6 2001:DEAD:BEAF:ABBA::1 2001:DEAD:BEAF:ABBA::2 prefixlen 128
```

6.2. Tunnel

6.2.1. OpenLDAP/ssh-based Tunnel Broker

The University of Southampton has developed its own tunnel broker implementation, using FreeBSD for the router (tunnel server) platform. The broker runs on an Apache2 Linux web server and uses ssh to execute tunnel creation commands from the broker to the server. The tunnel information is held in an OpenLDAP database. All these components run over IPv6.

6.3. 6over4

6.3.1. Microsoft Implementations

6over4 is included in the MSR IPv6 stack, Windows XP and the .NET Server family but is disabled default. By the very nature of the 6over4 transitioning method, an IPv4 multicast enabled infrastructure must be available for 6over4 to be able to work.

To enable 6over4 type:

```
# netsh interface ipv6 set global 6over4=enabled
```

at the command prompt. This will create a new 6over4 tunneling pseudo-interface for each IPv4 address assigned to the host.

6.4. 6to4

6.4.1. Cisco platform (as client and relay)

A Cisco router running a version of IOS that includes 6to4 support can either become a 6to4 client, if it only has IPv4 connectivity or, if the router already has global IPv6 connectivity, become a 6to4 relay.

6.4.1.1 Client configuration

To configure a Cisco router as a 6to4 client is rather easy and can be done by setting up a tunnel as follows:

```
interface Tunnel64
  no ip address
  no ip redirects
  ipv6 unnumbered FastEthernet0
  tunnel source FastEthernet0
```

```
        tunnel mode ipv6ip 6to4
```

One also has to set up the following route:

```
      ipv6 route 2002::/16 Tunnel64
```

6.4.1.2 Cisco configuration as a 6to4 relay

If the tunnel and route is configured as described above on a router, which also has outside connectivity to the rest of the IPv6 Internet, this router automatically functions as a 6to4 relay and can thus provide outside connectivity to all hosts connected to it by 6to4.

6.4.2. Extreme (ExtremeWare IPv6)

Dual stack Extreme switches (with a globally unique IPv4 address) can be configured to communicate with other isolated IPv6 domains using the 6to4 automatic tunnelling mechanism. Supposing that the two (layer 3) switches A and B are access routers of such domains willing to make use of 6to4, they should be configured as follows below.

Like configured tunnels 6to4 tunnelling requires an active IPv4 interface on the switch. With 6to4 the address of this interface is embedded in the IPv6 address of the tunnelling interface. If that particular IPv4 interface goes down, tunnelling fails even if other IPv4 interfaces are available on the switch.

To avoid this situation, we create an IPv4 VLAN with loopback enabled. Even if all of the ports on the VLAN go down, the VLAN stays up and so does the tunnel. At the cost of an additional VLAN on each 6to4 router and advertisement of the additional IPv4 address, you ensure greater tunnel stability.

Switch A:

```
      create vlan to_router_b
      configure vlan to_router_b add ports 1-10
      configure vlan to_router_b ipaddress 128.176.191.82/24
      enable loopback-mode to_router_b
      create tunnel 6to4_to_b 6to4
      configure tunnel 6to4_to_b ipaddress 2002:80b0:b807::1/16
      enable ipforwarding ipv6 6to4_to_b
```

Switch B:

```
create vlan to_router_a
configure vlan to_router_a add ports 1-5
configure vlan to_router_a ipaddress 128.176.184.7/24
enable loopback-mode to_router_a
create tunnel 6to4_to_a 6to4
configure tunnel 6to4_to_a ipaddress 2002:80b0:bf52::1/16
enable ipforwarding ipv6 6to4_to_a
```

6.4.3. Windows XP

First IPv6 functionality should be enabled on the Windows XP host as described in Appendix B. If there is no native IPv6 on the LAN the host is connected to, the operating system configures a "6to4 Tunneling Pseudo-Interface" and adds a default route pointing to Microsoft's Research 6to4 relay router, i.e. 6to4.ipv6.microsoft.com. This allows you to initially test the newly enabled IPv6 stack and its functionality through the "tracert6" command while pointing to the above host. Unfortunately, automatically-enabled configuration will not allow you to establish a connection to any other IPv6 host or networks.

In order to establish connectivity with other IPv6 networks, such as 6NET, the 6to4 tunnel should be configured to point to a free 6to4 relay router. For example, the following command

```
# netsh interface ipv6 6to4 set relay x.y.w.z
```

points the host's 6to4 tunnel to the 6to4 relay router with the IPv4 address x.y.w.z.

In order for this 6to4 client to serve a whole subnet as IPv6 default router, the /48-prefix, the host configured itself with needs to be advertised on the normal LAN interface. Also the default route needs to be specifically set to be published:

```
# netsh interface ipv6 add route 2002:<IPv4 address in hex>::/48 \\
                          <number of normal LAN interface> publish=yes \\
                               validlifetime=<valid lifetime of route> \\
               preferredlifetime=<period for which the route is preferred>
# netsh interface ipv6 set route ::/0 \\
                           <name or number of 6to4 interface> publish=yes
```

6.4.4. Windows 2000

A Windows 2000 host can be manually configured to access IPv6 sites through the 6to4 tunneling mechanism. The "`ipv6`" utility, which is included in the "Microsoft IPv6 Technology Preview" software, is used for setting the appropriate configuration to the 6to4 tunnel interface. The required steps for acquiring connectivity with other (6to4) IPv6 hosts are the following:

1. Configure the host 6to4 address for the "Tunnel Pseudo-Interface", using the colon-hexadecimal encoding of the IPv4 address of the Ethernet interface. For example, if the IPv4 host address is 195.251.29.19, the 6to4 IPv6 address should be 2002:c3fb:1d13::c3fb:1d13 and the full command is

   ```
   # ipv6 adu 2/2002:c3fb:1d13::c3fb:1d13
   ```

   Note that the "Tunnel Pseudo-Interface" is used for statically configured tunnels, automatic tunneling and 6to4 tunnels.
2. Add the appropriate entry in the routing table through the command that points all the IPv6 packets with 6to4 address to the "Tunnel Pseudo-Interface" by executing the command:

   ```
   # ipv6 rtu 2002::/16 2
   ```

3. In order to enable communication to an IPv6 network, a tunnel should be created to a 6to4 relay router. In the following example the command adds an IPv6 default route to a 6to4 relay at 194.177.210.38 through the interface 2:

   ```
   # ipv6 rtu ::/0 2/::194.177.210.38 pub
   ```

6.4.5. Linux

6.4.5.1 Manual Configuration using the command "`ip`"

To not depend on any distribution specific setup scripts one can create a 6to4 tunneling interface by hand using the command "`ip`" and tunnel mode "sit" just like with manually configured tunnels. The only difference here is that there is no need to give any information about remote tunnel endpoints.

1. Create a new tunnel interface:

   ```
   # /sbin/ip tunnel add <tunnelname> mode sit ttl <default ttl> remote any \
                                                   local <local IPv4 address>
   ```

   Example:

   ```
   # /sbin/ip tunnel add tun6to4 mode sit ttl 0 remote any \
                                                     local 128.176.184.7
   ```

   Note: A TTL must be specified. Otherwise the TTL is inherited (value 0).

2. Bring the new interface up:

   ```
   # /sbin/ip link set dev tun6to4 up
   ```

3. Add the local 6to4 address to interface. This is the address computed from the local IPv4 address used in the creation of the tunnel interface (see section 3.2.4). Note that it is important to use the prefix length 16.

   ```
   # /sbin/ip -6 addr add <local 6to4 address>/16 dev tun6to4
   ```

   In this example the line would be:

   ```
   # /sbin/ip -6 addr add 2002:80b0:b807::1/16 dev tun6to4
   ```

4. At last only the (default) route has to be configured accordingly. In this case we want to get the host connected to the rest of the IPv6 internet and thus configure the default route to point to the “all-6to4-(relay)-routers” IPv4 anycast address:

   ```
   # /sbin/ip -6 route add 2000::/3 via ::192.88.99.1 dev tun6to4 metric 1
   ```

6.4.5.2 Using network-scripts with Redhat

Initially, IPv6 functionality should be enabled on the Redhat host (see Appendix B). Furthermore, in order to set up a 6to4 tunnel, one has to follow the next steps:

5. In the /etc/syconfig/network-scripts/ifcfg-eth0 file include

   ```
         “IPV6TO4INIT=yes”
   ```

   which forces the system to initialize a tunnel (sit1) using the configuration defined in the file “ifcfg-sit1”.

6. Modify the /etc/sysconfig/ifcfg-sit1 file as follows:

   ```
         “DEVICE=sit1”                 Defines the name of the tunnel
         “BOOTPROTO=none”              Disables the boot protocol
         “ONBOOT=yes”                  Forces the interface to start on boot
         “IPV6INIT=yes”                Initialising IPv6
         “IPV6TUNNELIPV4=x.y.z.w”      Set the remote IPv4 address of the
                                       tunnel
         “IPV6ADDR=2002:aabb:ccdd::aabb:ccdd/0” Set local IPv6 address of the
                                       tunnel
   ```

   The above settings will create a 6to4 tunnel pointing to a 6to4 relay router. In the above configuration, x.y.z.w is the IPv4 address of a 6to4 relay router and 2002:<IPv4 address as hex>::<IPv4 address as hex> is the local 6to4 IPv6 address of the host (See section 3.2.4 on how 6to4 address are created). Note that the IPv6 address, e.g. 2002:aabb:ccdd::aabb:ccdd, is followed by a “/0” suffix in order to force the creation of a default route for all the 6to4 traffic through the `sit1` tunnel.

In the file /etc/sysconfig/network include the following line:

```
   “IPV6_GATEWAYDEV=sit1”
```

This sets the 6to4 tunnel interface `sit1` as the default gateway for all IPv6 traffic.

6.4.6. FreeBSD, NetBSD and Darwin/Mac OS X 6to4 Client

FreeBSD, NetBSD and Darwin/Mac OS X (as well as probably OpenBSD9 use the stf0 (six-to-for) interface to configure 6to4. Good instructions on how to configure a stf-interface can be found at the following two weblocations:

http://www.netbsd.org/Documentation/network/ipv6/#typical2

and

http://onlamp.com/pub/a/onlamp/2001/06/01/ipv6_tutorial.html

6.4.6.1 6to4 with NetBSD's 6to4-script (Perl)

The easiest way to configure 6to4 on NetBSD (may also work on the other OSes) is to install the Perl script called "6to4", that comes with NetBSD itself. The script uses a configuration file called 6to4.conf in which all the needed configuration information is collected and properly commented. On can install the 6to4-script either manually or from NetBSD's pkgsrc system where it is found in `/usr/pkgsrc/net/6to4`.

6.4.7. BSD with Zebra

In this example the router is configured to advertise the 6to4 anycast prefix 192.88.99.0/24 using OSPFv2 [RFC3068], and to advertise the 6to4 prefix 2002::/16 [RFC3056] via BGP.

With FreeBSD, the file "/etc/rc.conf" is use for system-level configuration. Here one should add:

```
ifconfig_xl0_alias0="inet 192.88.99.1 netmask 0xffffff00"
stf_interface_ipv4addr="192.88.99.1"
stf_interface_ipv6_ifid="::"
ipv6_gateway_enable="YES"
```

The first listed command configures the IPv4 anycast prefix as an alias for the x10 interface. Note that in real deployments, x10 will have to be replaced by an address corresponding to the used physical interface with IPv4 connectivity. The second command specifies the address with which the IPv6 6to4 address for the relay router is created (the bits 16-47 of the IPv6 address). The third command sets all the site- and interface-id bits to zero. The last command finally enables IPv6 forwarding on the router.

However, the file "rc.conf" can't be used to configure everything. For that reason some manual tuning of the file "rc.local" is also needed; the script is run after the processing of rc.conf has finished.

```
ifconfig stf0 inet6 2002:c058:6301:: prefixlen 16 anycast

/usr/local/sbin/zebra -d
```

```
/usr/local/sbin/bgpd -d
/usr/local/sbin/ospfd -d

/usr/local/bin/vtysh -b
```

The first command adds an "anycast" flag to the IPv6 6to4 address, so that it is not used as a source address in outgoing packets. The next three commands start zebra routing processes in the background. Finally the last command leads to "/usr/local/etc/Zebra.conf" being read and fed to the respective routing process as a configuration file.

The relevant parts of the Zebra configuration file /usr/local/etc/Zebra.conf" are:

```
router bgp 1741
 no bgp default ipv4-unicast
 neighbor 2001:708::2 remote-as 1741
 neighbor 2001:708::2 description 6net-rtr
 neighbor 2001:708:0:1::625 remote-as 1741
 neighbor 2001:708:0:1::625 description v6-rtr
!
 address-family ipv6
 network 2002::/16
 neighbor 2001:708::2 activate
 neighbor 2001:708::2 soft-reconfiguration inbound
 neighbor 2001:708:0:1::625 activate
 neighbor 2001:708:0:1::625 soft-reconfiguration inbound
 exit-address-family
!
router ospf
 ospf router-id 128.214.231.106
 network 128.214.231.104/29 area 3248883125
 network 192.88.99.0/24 area 3248883125
!
```

Here, the BGP sessions have been configured using AS1741 for two routers in the same autonomous system. The configuration syntax is similar to one used with Cisco IOS. The main difference is the Statement "`network 2002::/16`" under IPv6 address-family, which will originate a BGP advertisement for the 6to4 prefix.

The Prefix 192.88.99.0/24 is advertised using OSPFv2 under a stub area. The simplest way is to use area 0 though. IPv4 BGP has been configured to advertise 192.88.99.0/24, but this will not work unless there exists an IGP route to it. Using static routes would be an option, but in this case the traffic would be black-holed should the relay router go down. For this purpose, OSPF was chosen so the BGP advertisements would cease immediately if the OSPF process on the relay router halts for any reason.

6.5. ISATAP

To use ISATAP within a site, one will need one or more IPv6 hosts supporting ISATAP, and also a dual stack router supporting it. The following paragraphs on configuration examples cover both host and router setup on multiple platforms. Different implementations of the mechanism should interoperate so that host and router platforms can be mixed as needed.

For general security consideration when setting up ISATAP within a site please refer to section 3.2.6.1 in chapter 3. Implementation specific security issues are being addressed in this chapter where needed.

6.5.1. Cisco IOS Platform (as Router/Server)

The setup of a Cisco router as ISATAP server is pretty much straightforward. All one needs to do is to define a tunnel interface as follows:

```
interface Tunnel0
no ip address
no ip redirects
ipv6 address 2003:abc:def:123::/64 eui-64
no ipv6 nd suppress-ra
tunnel source FastEthernet0
tunnel mode ipv6ip isatap
```

If an ISATAP supporting client is set up to know about this Cisco router it will be able to automatically configure its interface with the prefix above. The last 64 bits will be based on the EUI-64 address and include the IPv4 address of the client as specified in the ISATAP draft [ISATAP]. According to present experience with ISATAP it does not matter which interface is specified as the tunnel source as long as it has an IPv4 address reachable by the clients. One can also specify an address directly.

6.5.2. 6WIND (as router/server)

The following configuration example was tested on a 6WINDgate 6231 running version 6.3.0b12 of SixOS.

Configuring a 6WIND router to become an ISATAP router is really rather easy and consists mainly of only two commands, one specifiying the IPv4 address of the tunneling interface the other defining the prefix to use for the global IPv6 addresses in the ISATAP subnet. This configuration enables the router to give itself a valid global (ISATAP-style) IPv6 address as well as to send out router advertisements to possible clients.

For the following example we assume that one has logged on to the router as user “admin” or as a user with similar permissions.

1. The first step in the configuration is of course to switch to the configuration context for the configuration file one wants to edit which in the case of this example is the running configuration. Within the configuration context one has to enter the migration context:

```
sixwind{} edit running
```

```
sixwind{running} mig
```

2. No follows the actual ISATAP configuration. First one has to configure an "ISATAP router", that is a separate routing process for the outgoing IPv4 interface one uses for the tunneling. It is possible to define several of these processes each with a different number and possibly different IPv4 addresses.

   The theoretic command for this configuration is:

   ```
   sixwind{running-mig} isatap_router 'number' 'address_v4' ['address_v4']
   ```

   As `'number'` one specifies the number of the ISATAP process, `'address_v4'` is the IPv4 address of the interface the automatic tunnels end on. The optional `'address_v4'` feature is given as either "up" or "down" and defines configured process is switched on or off. If no state is given the default is "down".

   Later on the state of an ISATAP routing process can also be changed by the command

   ```
   sixwind{running-mig} isatap_router 'number' 'state'
   ```

   In our example however we immediately specified the state as "up":

   ```
   sixwind{running-mig} isatap_router 1 128.176.191.74 up
   ```

3. Next a prefix has to be defined which is announced into the ISATAP subnet. This prefix should have length 64 and it should be taken care that there is not yet a route defined for this prefix, e.g. by having configured another interface of the router with an address within this prefix.

   The theoretic command for this configuration step is:

   ```
   sixwind{running-mig} isatap_prefix 'number' 'address_v6/len'
   ```

   specifies the ISATAP routing process this prefix corresponds to and thus should be the same as in the command above. In our example the prefix was configured as follows:

   ```
   sixwind{running-mig} isatap_prefix 1 3ffe:400:10:110::/64
   ```

4. At last after exiting from the migration context the new configuration has to be applied to the running configuration and one can also add it to the startup configuration to not loose it during a reboot:

```
sixwind{running-mig} exit
sixwind{running} addrunning
sixwind{running} exit
sixwind{} copy running start
```

6.5.3. Windows XP host (as client)

6.5.3.1 Manual Configuration as ISATAP Client

Since Service Pack 1 ISATAP, like all IPv6 functionality, is configured using the "`netsh`" command in the command shell. Only the IPv4 address of an ISATAP server is needed to configure a Windows XP host as an ISATAP client. (Of course it is required that IPv6 be installed on the host before using "`ipv6 install`"). The full command to set up ISATAP after that is:

```
c:\ netsh interface ipv6 isatap set router <IPv4 address of ISATAP router>
```

This is it. With the command "`ipconfig /all`" one can verify that the host has indeed received router advertisements from the server and configured its interface accordingly with an IPv6 ISATAP address. Using the command "`tracert`" or typing

```
c:\ netsh interface ipv6 show route
```

further shows, that the default route is now also configured for the ISATAP interface (number 4), even though 6to4 probably still also is configured.

6.5.3.2 Automatic Configuration as ISATAP Client

When the IPv6 protocol is started (e.g. at boot or installation) and realizes that there is no native IPv6 connectivity on the link, it tries to resolve the hostname "ISATAP" (on Windows XP without SP1 "_ISATAP"). If this hostname resolves into an IPv4 address the host will configure itself as an ISATAP client for this server and set the default route accordingly. Please note that the host will also configure 6to4 but just as a backup to the ISATAP connection or to communicate with other 6to4 sites.

6.5.4. .NET/Windows 2003 Server (as Client and Router/Server)

Windows 2003 Server can be configured as both ISATAP client and server.

6.5.4.1 Windows 2003 Server as ISATAP Client

Configuring a .NET/2003-server as an ISATAP client works just like with Windows XP either manually (see section 6.5.3.1) with the command *netsh* or automatically (see section 6.5.3.2) by resolving the name "ISATAP". The only difference is that once ISATAP is installed the 6to4 configuration is deleted.

6.5.4.2 Windows 2003 Server as ISATAP Server

For a host running Windows 2003-server to become an ISATAP router it is of course necessary for the advertised ISATAP prefix to be routed to the server. Additionally the default route of the host needs to be configured to be published. Otherwise Windows clients will not automatically set their default route to their ISATAP interface.

To configure the default route to be published the following command is used:

```
c:\ netsh interface ipv6 set route ::/0 \\
                             "<name or number of default interface>" publish=yes
```

If the Windows server is a dual stack host integrated in a native IPv6 subnet, the default interface will most likely be the normal LAN interface (4). Otherwise it might be a configured tunnel or 6to4 interface.

```
c:\ netsh interface ipv6 set interface \\
                                 "<interface name or number>" forwarding=enabled
```

Now the ISATAP interface has to be configured. In order to do so however it first needs to be enabled. This is achieved by configuring the Windows server as an ISATAP client for itself:

```
c:\ netsh interface ipv6 isatap set router <IPv4 address>
```

The IPv4 address refers to the physical interface used for the tunneling.

The ISATAP interface now also has to be set to forward packets. Additionally it has to be configured do send out router advertisements:

```
c:\ netsh interface ipv6 set interface 2 forwarding=enabled \\
                                                          advertise=enabled
```

At last the route and thus the prefix to be advertised on the ISATAP interface is configured. Like the default route this route has to be explicitly configured to be published.

```
c:\ netsh interface ipv6 add route <ISATAP prefix/64> 2 \\
            publish=yes validlifetime=<valid lifetime of route> \\
               preferredlifetime=<period for which this route is preferred>
```

The time periods can be specified in seconds (s), minutes (m), hours (h) or days (d) (e.g 1d2h3m4s). The default for `validlifetime` is eternity. If no `preferredlifetime` is given, the default is the value for `validlifetime`.

6.5.5. Linux (as Client and Router/Server)

ISATAP has been included in the Linux kernel by the USAGI project [USAGI]. It can thus be patched into any present kernel sources and compiled as is described further down. One also needs the iproute package from USAGI.

6.5.5.1 Patching the Kernel

First one has to download the newest USAGI patch for the kernel one wants to patch, which are available on USAGI's pages at

ftp://ftp.linux.ipv6.org/pub/usagi/

Of course it is best to use stable versions.

Once downloaded and unpacked one can patch the existing kernel sources in /usr/src/linux by typing:

```
# patch -p1 <usagi-linux2<4,5 or 6>-stable-<snapshot>-<kernel-version>.diff
```

After the sources have been patched the kernel needs to be configured by typing

```
# make xconfig
```

The section 'networking options' contains the new features. Here 'IPv6: ISATAP interface support (EXPERIMENTAL)' needs to be chosen.[1]

6.5.5.2 Building USAGI `iproute` Packages

The USAGI project has its own iproute package. This is included in usagi-tools-* which can also be found at the above URLs. If not all tools are wanted one can also install iproute individually as an rpm package from

http://v6web.litech.org/isatap/dist/

The version to be found there is not up to date though.

[1] Please note that one has to switch on 'prompt for development and/or incomplete code/drivers' in section 'code maturity level options' to be able to switch on any experimental features at all

6.5.5.3 Configuring a Linux Host as an ISATAP Gateway

If a Linux host is used as an ISATAP router this router becomes a kind of access router for an ISATAP subnet. This subnet has to be provided with a prefix just like any other subnet and this prefix has to be routed.

During a test setup, the host lemy.uni-muenster.de has been configured as an ISATAP router. At the same time this host was already a normal access router to another IPv6 subnet with the prefix 2001:638:500:200::/64 and connected to the rest of the IPv6 internet by an IPv6-in-IPv4 tunnel.

For simplicity the /64 prefix was shortened to /63, so 2001:638:500:201::/64 could be used for the ISATAP subnet.

After compiling the kernel and rebooting the host the first thing to do is to configure an ISATAP interface (is0). In the test setup the IPv4 address of the interface where the packets come through (eth0 on host lemy.uni-muenster.de) was 128.176.184.113. So the interface was configured by typing:

```
# /sbin/ip tunnel add is0 mode isatap local 128.176.184.113 ttl 64
```

To enable the interface:

```
# /sbin/ip link set is0 up
```

Now the interface has to be configured with an IPv6 ISATAP address. An ISATAP address includes the IPv4 address of the host and has the format:

<prefix/64>:0:5efe:<IPv4 address in hex format>

In this case host lemy.uni-muenster.de has the IPv4 IP 128.176.184.113 and is configured with an address for the prefix 2001:638:500:201::/64. So the ISATAP address for lemy is:

2001:638:500:201:0:5efe:80b0:b871

To add this address to the interface configuration type:

```
# /sbin/ip addr add 2001:638:500:201:0:5efe:80b0:bf71/64 dev is0
```

Now the host has to actually be configured to become a router and send out router advertisements. Usually ‘radvd’ is used for router advertisements. It can be found at

http://v6web.litech.org/radvd/

In this case the configuration file for radvd (usually /etc/radvd.conf) has the following entries:

```
interface is0
{
      AdvSendAdvert on;
      UnicastOnly on;
      AdvHomeAgentFlag off;
      prefix 2001:638:500:201::0/64
      {
            AdvOnLink on;
            AdvAutonomous on;
            AdvRouterAddr off;
      };
};
```

Now the router configuration is done. To have this setup come up at boot one can best add the "/sbin/ip"-lines from above to /etc/rc.local.

6.5.5.4 Configuring a Linux Host as an ISATAP client

Configuring a Linux host as an ISATAP client is rather easy. One really only has to know the IPv4 address of the host acting as the ISATAP gateway (in this case lemy.uni-muenster.de = 128.176.184.113) and the IPv4 address of the client in question (here 128.176.245.58). With this information configuration is performed by two commands:

```
# /sbin/ip tunnel add is0 mode isatap local 128.176.245.58 v4any \
128.176.184.113 ttl 64
# /sbin/ip link set is0 up
```

Now the client should be connected to the rest of the IPv6 world, as long as the ISATAP server has global IPv6 connectivity.

6.6. DSTM

6.6.1. A DSTM Experiment with FreeBSD 4.5

6.6.1.1 The test network topology

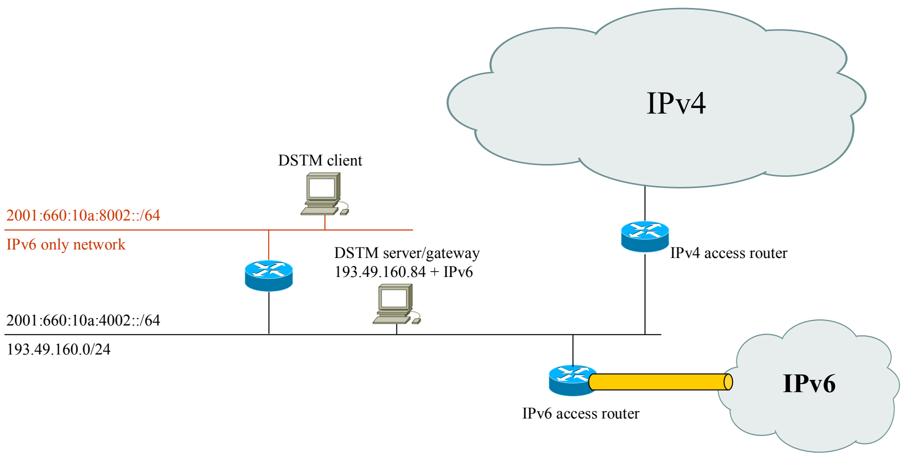


Figure 6-1: Test Network Infrastructure

The DSTM server/TEP implementation by ENST [ENST] is available on the ENST's web site:

http://www.ipv6.rennes.enst-bretagne.fr/dstm/

The implementation requires that the server and gateway (TEP) be installed on the same host if the allocations made by the server should be coherent with the tunnels set up on the TEP. An external TEP (For example a 6Wind router with DSTM code) can also be used (But, this scenario has not been tested). In that case the TEP manages the tunnels directly and does not interact with the server, which then only is responsible for distributing the leases of IPv4 addresses. It must only be insured that the leases expire/renew durations are coherent with respect to the TEP configuration. In this configuration example however the TEP and the server are the same program. Thus the tunnel extremities are on the machine running the server and there is no way to use two different Linux/FreeBSD nodes separating the server from the TEP.

When installing DSTM one has to be aware of the fact that the TEP has to be the router declared as having access to the IPv4 address pool from the IPv4 world, and that the DSTM server program does not manage route advertisements. Therefore, on the server site, one must provide by some way routes and route advertisements for the IPv4 addresses of DSTM clients toward the TEP node. For example on the site entry router, one can set a route for the remote IPv4 address pool allocated to DSTM clients toward the TEP IPv4 address, and make sure that this IPv4 route is announced or aggregated. Also there are routing loops between the TEP and the site IPv4 default router thus, on the TEP node, one should set a static discard route to suppress all traffic to down clients.

The DSTM implementation uses RPCv6, TSP and TSP–SSL (For VPN scenario) for the communication between DSTM Server/TEP and DSTM clients. Until now, the current implementation has no support for DHCPv6

6.6.1.2 DSTM Installation

To Install DSTM, one needs to perform the system installation which includes the application of an RPC patch from the ENST website and then commencing with the installation of DSTM system modules. To install DSTM system modules the source code should be unpacked and after going to dstmd directory, the '`make system`' and '`make systeminstall`' commands should be executed to compile the system modules.

Note that the website mentioned above explains the installation procedure in more detail.

After system installation one needs to configure the DSTM server/TEP and the DSTM clients as follows:

6.6.1.3 Configuration of the DSTM server/TEP

In this example the following lines where the output from the command "`ifconfig`" before starting on the configuration of the DSTM server:

```
xl0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet 193.49.160.84 netmask 0xffffff00 broadcast 255.255.255.0
        inet6 fe80::260:8ff:fe59:6623%xl0 prefixlen 64 scopeid 0x1
        inet6 2001:660:10a:4002:260:8ff:fe59:6623 prefixlen 64 autoconf
        ether 00:60:08:59:66:23
        media: Ethernet 10baseT/UTP (10baseT/UTP <half-duplex>)
gif0: flags=8050<POINTOPOINT,RUNNING,MULTICAST> mtu 1280
        inet6 fe80::260:8ff:fe59:6623%gif0 prefixlen 64 scopeid 0x7
```

The following lines should be added to /etc/rc.local:

```
      ifconfig gif0 create
      ifconfig gif1 create                      }
      ifconfig gif2 create                      }  creates 4 tunnels
      ifconfig gif3 create
sysctl -w net.inet.ip.forwarding=1              # enables IPv4 routing


route add -net 195.98.237.144/28 193.49.160.126 –reject
```

The following lines should be added to /etc/rc.conf:

```
      hostname="DSTM-server.renater.fr"               # Name of the workstation
      ifconfig_xl0="inet 193.49.160.84 255.255.255.0"          # IP configuration
```

```
ipv6_enable="YES"                                         # Enables IPv6
portmap_enable="YES"                     # Enables portmap (used by DSTM)
portmap_program="/usr/sbin/rpcbind"                   # Portmap program
defaultrouter="193.49.160.126"                        # Default gateway
```

In /usr/local/etc/rpcdstmd.conf one should add:

```
subnet 195.98.237.144 netmask 255.255.255.224 {    # IPv4 address pool used
   default-lease-time 1200;
   tep6 2001:660:10a:4002:260:8ff:fe59:6623;          # IPv6 address of the
                                                      # TEP (itself here)
   tep4 193.49.160.84;                # IPv4 address of the TEP (itself here)
   range 195.98.237.146 195.98.237.158;# Part of the pool used for allocations
}
```

To launch the server one has to execute:

```
# touch /var/db/rpcdstmd.lease
# /usr/src/sbin/dstmd/server/rpcdstmd –notsp –rpcport 6000
```

6.6.1.4 Configuration of the DSTM client

Output from command "`ifconfig`":

```
xl0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=3<rxcsum,txcsum>
        inet6 fe80::2c0:4fff:fe83:22c4%xl0 prefixlen 64 scopeid 0x1
        inet6 2001:660:10a:8002:2c0:4fff:fe83:22c4 prefixlen 64 autoconf
        ether 00:c0:4f:83:22:c4
        media:  Ethernet  autoselect  (100baseTX  <full-duplex>)  status:
active
```

The following lines should be added to the file /etc/rc.config on the client host.

```
hostname="DSTM-client.renater.fr"
ipv6_enable="YES"
```

Lines to add to /etc/rc.local:

```
      ifconfig gif1 create                          # creates one tunnel interface
sysctl -w net.inet.ip.dti_magic=10 # sets up the DTI timer (time the kernel sleeps
                                    # waiting dstmd daemon to do something)
```

To launch the daemon type:

```
# dstmd -rpcserver <DSTM-server> –port 6000
```

It is necessary to specify the name or the address of the DSTM server and sometimes the port.

6.6.1.5 Tests results and Issues

When installation and configuration is accomplished, it is possible for every DSTM client to communicate with IPv4 hosts using IPv4 enabled applications.

The performance testing of DSTM was done over a Very High Speed Network (1 Gbps) and comparison was made between normal IPv4 communication and DSTM IPv4 communication over the network. When DSTM performance was measured after measuring IPv4 performance, then it was found that RTT (Round Trip Time) decreased just about 2 percent and throughput decreased just around 7 percent if the payload size was kept below 1212 bytes (limit after which fragmentation starts with DSTM configurations). If the payload size was bigger, throughput decreased a bit more due to the increase of load with the DSTM mechanism.

There are also some other issues:

- Some times in older kernel versions the make script in the dstmd/bsd folder works only after a slight and minor modification.
- One has to rebuild the full kernel after making the RPCv6 patch and if one uses the patch, then it is also very sensitive to system version.
- RPCv6 mechanism cannot cross firewall and cannot be used for VPN scenario.
- With KAME, the correct number of gif (interfaces) must be put in the system configuration file (no dynamic gif).
- All IPv4 communication must be initiated by the DSTM client, because the DSTM client is the one that requests the tunnel to be set up (Although there exists an implementation that was not tested yet making it possible for remote IPv4-only hosts to initiate a communication with a DSTM client in an IPv6-only network).

6.6.2. DSTM usint TSP-SSL (in a VPN scenario) on FreeBSD

6.6.2.1 Installation and Setup

Please refer to the previous section 0 about how to install DSTM on a FreeBSD host.

In order to use SSL with TSP, some certificates are needed:

1) Certificates with authority to sign other certificates (CA).
2) Certificates (cert) and corresponding private keys, signed by some known CA.

For both server and client the following files and certificates are needed:

1) *local cert:* A file containing a certificate for the local machine and the corresponding private key. In the following example configuration this file will be /etc/dstmd/cert.pem. The certificate is signed by some CA. This CA must be known by any correspondent (the other side of a TSP connection) in its CA file (see below).

Both the certificate and key are in PEM format, the certificate is the first data in the file.

The file contains:

```
        Comments
                -----BEGIN CERTIFICATE-----
                certificate
                -----END CERTIFICATE-----
                Comments
                -----BEGIN RSA PRIVATE KEY-----
                Private key
                -----END RSA PRIVATE KEY-----
```

The certificate's key may be protected (crypted) with a password/pass phrase. If the certificate's key is not encrypted, some measures should be taken to protect the file /etc/dstmd/cert.pem as a whole, i.e:

```
# chown root /etc/dstmd/cert.pem; chmod 400 /etc/dstmd/cert.pem
```

2) *password file:* If the certificate's key is encrypted, put the pass phrase needed to decode it in some other file (in clear text), e.g. /etc/dstmd/pass and protect it:

```
# chown root /etc/dstmd/pass; chmod 400 /etc/dstmd/pass
```

3) *CA:* A file containing all CAs used for signing certificates of correspondents, and also the CA signing the CA of this file (if there is a chain of certification), concatenated in some file, e.g.

```
   cat .../*CA*pem > /etc/dstmd/cacert.pem
```

Each CA is in PEM format, the file contains:

```
        Comments
                -----BEGIN CERTIFICATE-----
                First CA
                -----END CERTIFICATE-----
                Comments
                -----BEGIN CERTIFICATE-----
                second CA
                -----END CERTIFICATE-----
        ...
```

4) *Accepted cert list:* If you want to restrict access to only some certificates.

One *local cert* is needed for the server and one for each client, but they may be the same - if the security risk of sharing is accepted. Note: *cert key* may protected by a password; if you use a password, it must be passed (clear text) to dstmd/rpcdstmd in a file (protected! use chmod 400). If you do not use a password, the *key/cert* file should be protected (chmod 400).

To obtain a certificate, either use a certificate issued by some authority (it seems that standard certificates are sufficient, no special "role" is needed), or use the easiest way and create one using the program openssl as described in the 'How to create CA and certs, using openssl' section of the README-SSL file provided with the SSL version of DSTM available on ENST's website.

6.6.2.2 Configuration of the DSTM Server/TEP

The DSTM server/TEP is configured exactly as described above (when using RPCv6). The difference when using TSP only arises when starting the server, for which one needs to execute the following command:

```
#rpcdstmd –tspport 7000
```

For using SSL with TSP start rpcdstmd with the following additional options:

```
-key /etc/dstmd/cert.pem -ca /etc/dstmd/cacert.pem
```

If the files exist and should be used also specify the following options:

```
-pass /etc/dstmd/
-cert /etc/dstmd/accepted.pem
```

6.6.2.3 Configuration of the DSTM Client (with TSP)

Again the only difference to using DSTM with RPC is the command to start the daemon:

```
#dstmd -tspserver 2001:688:1f9b:1003:207:e9ff:fe11:bfb8 -port 7000
```

When also using SSL of course also the necessary certificate options need to be specified:

```
-key /etc/dstmd/cert.pem -ca /etc/dstmd/cacert.pem
```

```
-pass /etc/dstmd/pass      (if /etc/dstmd/pass exists)
-cert /etc/dstmd/accepted.pem  (if /etc/dstmd/accepted.pem exists)
```

6.6.2.4 Testing Results and Issues

Installation was easy as an RPC patch was not needed and moreover DSTM is now a module so there was no need to recompile the kernel. The use of SSL greatly increases security but will also lead to a rather significant performance loss due to the added load of certificate verification when setting up new connections.

Please refer to the last section (6.6.1.5) of the configuration example on using DSTM with RPC on a FreeBSD systems for more implementation results.

6.6.3. Linux (RedHat 7.3, 8.0, 9.0)

6.6.3.1 Availability

DSTM (currently version 1.4) for Linux is available from ENST and is virtually identical to the FreeBSD version available from the same source. The Server and TEP components must be co-located to work under this implementation and the client must be installed on any host that requires DSTM, the components communicate via RPCv6. The mechanism is available from the link below.

http://www.ipv6.rennes.enst-bretagne.fr/dstm/

6.6.3.2 Initial Installation and Configuration

To install DSTM the files must first be unzipped/untared into an appropriate directory and the system modules compiled and installed. Regardless of the module required, the same basic configuration steps must be taken. First the system module must be compiled using the '`make system`' and '`make installsystem`' commands. Mostly there is no need of a kernel rebuild but the modules work only on a 2.4.* kernel where ipv6 is a module. Otherwise for new kernel versions, see 'linux/00README' and the module ipv6f and sometimes kernel rebuild will be needed after applying the given kernel patch. Also for the new kernels the module stuff (insmod etc.) has changed and hence must be updated.

The RPCv6 patch is not required even when using RPC. When using TSP + SSL one needs certificates as described in the FreeBSD section 6.6.2. When only using TSP certificates are not needed.

6.6.3.3 Server/TEP Installation and Configuration

Installing the Server/TEP module (rpcdstmd), is done by moving to the '/dstmd/server' directory and running the 'make' and 'make install' commands.

The server is configured via the rpcdstmd.conf file that takes the following format:

```
subnet 194.80.38.225 netmask 255.255.255.240 {  # IPv4 address pool used
```

```
                                                # default-lease-time 1200;
        tep6 2001:630:80:7100::1;               # IPv6 address of the TEP
        tep4 194.80.38.38;                      # IPv4 address of the TEP
        range 194.80.38.227 194.80.33.230;      # Part of the pool used for
                                                # allocations
    }
```

There is some minor additional setup to be done at this point before the server can be started. First, IPv4 forwarding must be enabled:

```
# sysctl -w net.ipv4.ip_forward=1.
```

Also, a lease file must be created:

```
# touch /var/db/rpcdstmd.lease
```

The needed modules are now automatically loaded and tunnels re created if needed, so there is no need for "-load" or "-create" options when running the server.

When running the server with RPC support one can execute:

```
# /usr/src/sbin/dstmd/server/rpcdstmd –notsp –rpcport 6000
```

To start the server with TSP support the command is the following:

```
# /usr/src/sbin/dstmd/server/rpcdstmd –tspport 6000
```

When using both SSL and TSP the program rpcdstmd has to be started with the following additional options:

```
-key /etc/dstmd/cert.pem –ca /etc/dstmd/cacert.pem
```

Of course the filenames "`cert.pem`" and "`cacert.pem`" need to be substituted with the real certificates on the system.

Also add the options

`-pass /etc/dstmd/pass` and

`-cert /etc/dstmd/accepted.pem`

if either file exists and should be used.

Sometimes when the modules like dti.o are required, they do not load while launching the sever then the moduls should be loaded by using "`insmod dti.o`" or the "`modprobe`" after changing to the dstmd/linux/dtmod directory.

6.6.3.4 Client Installation and Configuration

To install the DSTM client module (dstmd) the '`make`', '`make install`' commands should be run from the '/dstmd' directory but no further configuration is necessary.

To run dstmd, a number of command line options must be included. One has to specify the server name (or IPv6 address) and port number that identifies the server and also an option to load the rpc module is necessary under Linux.

```
# dstmd -rpcserver penguin.trans.ipv6-uk.net -port 6000
```

When using TSP one can use the following command:

```
# dstmd -port 3545 –tspserver 2001:688:1fa1:2::100
```

When also using SSL with TSP one has to start the dstmd with the additional options:

```
-key /etc/dstmd/cert.pem –ca /etc/dstmd/cacert.pem
```

Also add the following options when either file is present and should be used:

```
-pass /etc/dstmd/pass
-cert /etc/dstmd/accepted.pem
```

In order to use IPv6 transport for DNS queries one should put IPv6 address(es) in the nameserver lines in the file /etc/resolv.conf on a client host. Otherwise one should at least not specify a hostname for the rpcserver option of the dstmd client if one uses IPv4 name resolution.

When the module dstm.o is required, it sometimes does not load automatically when launching the client. I those cases the module can be loaded by hand through the use of either the command “`modprobe`” or “i`nsmod`” (after changing to dstmd/linux/dstmmod).

6.6.3.5 Operational and Installation Issues

When installing DSTM under Linux (RedHat 7.3 or 8.0) there were a number of issues encountered. Primarily, the kernel source (exact same version) is required in order to install properly and under RedHat (and perhaps other Linux derivatives) this is not in the default location. This can be resolved by providing a command line redirect. Also, when compiling the system module, the ‘`make system`’ and ‘make `installsystem`’ commands should be run, not ‘`make systeminstall`’ as listed.

Installation of DSTM on a Linux host with kernel 2.5.70, things where a little more complicated as IPv6 was not a module which made it necessary to patch the kernel itself and rebuild it completely. Even in this case though the modules dti.o or dstm.o sometimes fail to load automatically.

When installing the DSTM client (dstmd), the make install command fails due to dstmd.8 not being in /usr/local/man/man8. Under RedHat there is no /usr/local/man/man8 directory so this must be created and the dstmd.8 file copied there manually.

Also, when installing the server (rpcdstmd) the instructions specify to use ‘`make depend`’, ‘`make`’ and ‘`make install`’, we found the ‘`make depend`’ command did nothing and so is unnecessary.

When installation and configuration is accomplished it should be possible for every DSTM client to communicate with IPv4 hosts using IPv4 applications. Moreover if the DSTM server is used with SSL options then the IPv4 address allocation takes place only after verifying the certificates. DSTM clients without a valid certificate are denied for address allocation. This greatly increases security but also slows down the process of address allocation and hence initialisation of IPv4 communication. It also makes it a little more complicated for the user end as one needs to come by the SSL certificates beforehand.

As with the FreeBSD implementation IPv4 communication can only by initiated by a DSTM client because the DSTM client is the one that requests the tunnel to be set up. There exists however a DSTM implementation that was not tested yet which permits communication to be initiated by outside (IPv4-only) hosts.

6.6.3.6 Conclusions

While the Linux version of DSTM is essentially the same as in FreeBSD, the installation and configuration is less complicated than the FreeBSD version and should be given preference. For example, the RPC patch is not required and various configuration options are simplified under Linux.

In addition to the version evaluated here from ENST, there is a gateway/TEP mechanism available in the 6wind routers and there is now a DSTM Client for Windows XP from 6talk.net that should work with this implementation. However, while we can confirm that the 6wind gateway works with this to some extent, neither has been evaluated here.

7. Configuration Examples: IPv6 translation methods

7.1. NAT-PT

7.1.1. NAT-PT (RedHat 7.3)

7.1.1.1 Availability

This implementation of NAT-PT was developed by ETRI using the Linux kernel 2.4.0-test9 and should work on any kernel 2.4.x or above (it was tested successfully on RedHat7.3, 8.0 and various USAGI kernels). It also includes both DNS and FTP ALGs functionality and is available from the following link:

http://www.ipv6.or.kr/english/natpt-overview.htm

7.1.1.2 Installation and Configuration

To install NAT-PT the files must first be unzipped/untared into an appropriate directory and some minor configuration must be done.

Device Installation and Configuration

Primarily the IPv4_addresses.list file must be populated with the IPv4 address pool to be used and a static binding made for the DNS. This file takes the form of a basic list shown below:

```
194.80.38.226
194.80.38.227
```

194.80.38.228

194.80.38.229

DNS 194.80.38.225 2001:630:80:7100::10

Additionally, minor changes may be made to nat-pt_global.h and nat-pt.c to configure the network prefix and the IPv4 and IPv6 network interfaces respectively though in our case this was unnecessary.

The application is compiled using ‘make’ or ‘make clean’ and run using nat-pt. DNS must also be properly configured in order for NAT-PT to create dynamic address bindings.

Host Configuration

No major changes are needed for host configuration however the DNS may need setting up. The resolvers (DNS clients) on the IPv6 hosts must be configured to send DNS requests to an IPv6 address. This can either be the 'real' address of a native IPv6 DNS server, or the mapped address of an IPv4 DNS server.

7.1.1.3 Operational Issues

A number of issues were encountered when compiling the NAT-PT device, primarily due to a missing ‘included’ file, bpf.h. This file appears in 5 files; nat-pt_global.h, nat-pt.c, alg_manager.c, utils.c and dns_alg.c and each reference must be updated in order to make the compile successful. Additionally, in nat-pt.c a constant from the ‘clock’ function CLK_TCK is used erroneously and must be replaced by ‘time()’ which returns CLK_TCK.

7.1.1.4 Conclusions

Operation experience shows the mechanism to work both under normal conditions and when using the ALG mechanisms but suggests the system is slightly unstable.

7.1.2. Ultima (FreeBSD 4.5)

7.1.2.1 Availability

Ultima is a NAT-PT implementation from BT labs, it supplies a NAT-PT device with DNS, FTP and SIP ALG functionality. In this case, it was installed on FreeBSD 4.5 but should run on any FreeBSD above version 4.2. It cannot however be directly downloaded from the Internet as is the case above but is available on a case by case basis, below is a link the BTExact highlighting the Ultima transitioning toolkit.

http://www.ipv6.btexact.com/activities/projects.html#ultima

7.1.2.2 Installation and Configuration

Installation of Ultima is two-fold in that there is the basic device configuration but also the http interface setup, which is rather less well documented. However, installation of the device itself is straight forward, simply unzip the file into an appropriate directory and run the ultima_install program.

Device Installation and Configuration

Setup is through a text-based interface which is both comprehensive and simple to use and can also be run at any time to reconfigure the device. It follows these basic steps.

IPv4 interface	Specify IPv4 interface of device
IPv6 interface	Specify IPv6 Interface of device
IPv6 prefix	Specify IPv6 prefix of Ultima
Address Pool configuration	Configure IPv4 address pool, selecting this option allows the addition or removal of IPv4 addresses in the address pool and the addition or removal of static address mappings. At least one static mapping is required for the DNS
Lan/wan and DNS options	Allows selection of lan or wan operating modes (basically turns on or off rtadv. Configures DNS ALG to allow IPv4 or IPv6 queries to get forwarded or blocked on either interface
http interface password below)	Configures http interface password (see

The http interface setup is rather more complicated. This requires both the apache and openSSH ports to be installed onto the machine in addition to numerous other PERL modules. Afterwards, the interface package must be complied and installed. Once installed, the device can be operated remotely via a browser.

Host Configuration

No major changes are needed for host configuration however the DNS options may need to be configured.

7.1.2.3 Operatioonal Issues

The installation procedure threw up several complications in that the supplied ‘ultima_NAT-PT.zip’ package cannot be easily unzipped in FreeBSD with the usual methods, the package seems to be windows-based with MSWord documents supplied as opposed to the usual README text files. To unzip the files, PKZip for UNIX was used though they could be unzipped in Windows and transferred to UNIX.

Additionally, the file permissions had to be changed as they were not set to be executable by any user by default (ultima_install, ultima, get_passwd, check_prefix, check_IPv4_address) so the device could not be configured or run.

Finally, the http interface while very nice does not come with installation instructions beyond the requirements for the various ports and PERL modules.

7.2. ALG

7.2.1. WWWoffle

This configuration example assumes that wwwoffle is configured to listen on all IPv4- and IPv6-sockets of the server it is installed on. The sample configuration that comes with wwwoffle source packages may be used as a reference.

Note that this configuration does not cover all cache-specific options. The following parts of the wwwoffle configuration file are excerpts that cover only the IPv6-specific parts of the configuration.

7.2.1.1 StartUp-Section

This section sets options that are parsed by wwwoffle on startup.

```
StartUp
{
 # Bind to all available IPv4-addresses.
 bind-ipv4         = 0.0.0.0

 # Bind to all available IPv6-addresses.
 bind-ipv6         = ::

 # Let proxy listen on port 8080.
 http-port         = 8080

 # Let HTTP-server for wwwoffle-control run on port 8081.
 wwwoffle-port     = 8081

 # Spool-directory for cache.
 spool-dir         = /var/spool/wwwoffle

 # Do syslogging.
 use-syslog        = yes

 # Set password for control pages to "secrect".
 password          = secret

 # Max.-number of server-threads.
 max-servers       = 8

 # Max.-number of servers when fetching pages.
 max-fetch-servers = 4
}
```

Please note that the performance-specific options need to be adjusted to fit individual needs. Most options that do not directly concern IPv6 are left to their default values.

7.2.1.2 LocalHost-Section

This section contains aliases and IPs under which the wwwoffle server is known. Requests to any of those aliases or IPs are not cached. Configuring this section is trivial. Valid entries are names and IPs without wildcards. The first entry will be used as the proxy's name which is relevant for some features of the wwwoffle configuration page.

```
LocalHost
{
 proxy.mynetwork.org
 proxy.ipv6.mynetwork.org
 localhost
 127.0.0.1
 ::ffff:127.0.0.1
 ip6-localhost
 ::1
 128.176.184.159
 2001:638:500:200::ff00
 3ffe:2a00:100:7efa::2
 3ffe:666:3ffe::22
}
```

7.2.1.3 Other Sections

In general, all other sections are configured as they would be for an IPv4-only wwwoffle. Additionally, for options which hold an IPv4 address, IPv6 addresses may be used as well in the same manner as in the previous section.

7.2.1.4 Client-Side Configuration

All clients in the IPv6-only subnet should be configured to use the wwwoffle server as a proxy. This is done by configuring all web- and ftp-clients and similar applications accordingly.

7.2.2. WWW6to4 HTTP Proxy

The www6to4-1.5 HTTP proxy is a small program that is meant to act as a dual-stack front end to an IPv4 only browser. It has a few other features but those are entirely optional and not discussed here. This proxy is meant to run on a client machine and not to server a large number of clients and keep a cache for them. For the latter one is much better off with a full-fledged proxy like squid or wwwoffle.

The program is primarily distributed in source form, but binary packages for NetBSD and Debian Linux can also be found. Compilation from source normally involves no more than typing 'make' in the source directory. To run it, one needs a configuration file www6to4.conf including at least the following two lines:

```
listen-to              127.0.0.1,::1
listen-port            8000
```

If the www6to4.conf configuration file is located in the /etc-directory, www6to4 can simply be started with:

If the configuration file is located somewhere else its location needs to be given at the command line:

```
www6to4 –c <path to configuration file>/www6to4.conf
```

Note that there is no reason why www6to4 runs as root, it can run as an ordinary user or even as user 'nobody'.

Once www6to4 is running, all that is left to do for the user is to configure his or her web browser to use it as its the proxy (localhost:8000).

www6to4 is not an ftp proxy, it will proxy for http and https only. However, www6to4 is capable of 'forwarding' ftp URL's to another proxy that does understand ftp. In that case a so-called 'forward-file' needs to be configured by adding the following line to the www6to4.conf file:

```
forwardfile            <path_to>/www6to4_forward.conf
```

The www6to4_forward.conf should then contain a line like the following example:

```
ftp://*        proxy.mydomain.tld:8000
```

The following functionality was recently requested and added by the 6NET project to www6to4. I will be officially integrated in the next release (version 1.6). The program www6to4 has now a new option called "`-forwardingonly`". This is useful in IPv6-only networks, where one wants to connect directly to IPv6 servers over IPv6, but where requests to IPv4 servers need to be forwarded to another (per site) dual-stack proxy. The new option provides exactly this functionality; it looks up the DNS IPv6 address recors (AAAA) for the requested HTTP server and only if no such records are found wil it forward the request according to the rules in the forward file (typically named /etc/www6to4_forward.conf). The forward file then needs to contain the following lines:

```
ftp://*          proxy.mydomain.tld:8000
http://*         proxy.mydomain.tld:8000
```

These lines tell the www6to4 program to forward any ftp or http request to proxy.mydomain.tld at port 8000.

7.2.3. Postfix Configuration

7.2.3.1 Server side configuration

In most cases, postfix' configuration files are located in /etc/postfix. Basically, only the file main.cf needs to be modified. When modifying main.cf, it is important to be aware of potential security risks that may occur if these modifications are not done carefully. One should under all circumstances avoid creating open relays that allow spam- and bulk-mailers to abuse the ALG as a relay-server.

To be able to relay mail from the IPv6-only subnet, this subnet has to be added to the trusted subnets (the class of subnets that are allowed to relay mail via this server). By default, postfix trusts all machines that are in the same subnet as the postfix-server. This behaviour is configured by setting

```
mynetworks_style = subnet
```

in main.cf. We assume that the IPv6-only subnet has the prefix 3ffe:400:10:110::/64. To allow hosts in this subnet to relay via this postfix-server, we add the prefix to “mynetworks” in main.cf.

```
mynetworks = [2001:638:500:200::]/64, [3ffe:400:10:110::]/64, \
                                                [::1]/128, 127.0.0.0/8
```

Note that in this case, 2001:638:500:200::/64 is the subnet that the postfix-server is located in.

Using the two options above allows relaying from the IPv6-only subnet for which the postfix-server acts as an ALG.

7.2.3.2 Client Side Configuration

The postfix server above may be used as a normal SMTP-server by any e-mail clients in the IPv6-only subnet. This has to be configured for each application separately.

7.2.3.3 Mail Transfer Applications (MTAs)

MTAs such as postfix and sendmail can be configured to use an SMTP-smarthost. In postfix’ case, this is done by adding the following option to main.cf:

```
      relayhost = [2001:638:500:200:0:0:0:ff00:25]
```

This lets postfix relay all outgoing mail via 2001:638:500:200::ff00:25. 2001:638:500:200::/64 is the subnet that the postfix server is located in.

Using the two options above allows relaying from the IPv6-only subnet for which the postfix server acts as an ALG. This is a save configuration in terms of relay protection as long as only the IPv6-only subnet is allowed to use the postfix server as a relaying smarthost.

7.2.4. SMTP Relaying with Sendmail

7.2.4.1 From IPv6 to IPv4

One can set up a single dual-stack mail server to act as a so called ‘smart host’ (smart mail relay) for IPv6-only hosts. A sendmail configuration file ‘sendmail.cf’ is typically generated from an m4 macro file. One should add the following lines to this m4 macro file:

```
DAEMON_OPTIONS(`Family=inet, address=0.0.0.0, Name=MTA')dnl
DAEMON_OPTIONS(`Family=inet6, address=::, Name=MTA6, Modifiers=O')dnl
```

IPv6 is marked optional in the above setting so that the ‘sendmail.cf’ generated from it can be used on IPv4-only kernels as well.

Once the dual-stack mail server has been set up, IPv6-only hosts can configure it as their ‘smart host’. For sendmail a smarthost can be configured by adding the following line to its existing ‘sendmail.cf’ configuration file:

```
DSmail64.mydomain.tld
```

Here ‘`mail64.mydomain.tld`’ is the domain name of the dual-stack mail server that has been set up.

7.2.4.2 From IPv4 to IPv6

Two methods can be used to relay email through a dual-stack sendmail server to an IPv6-only mail server. The first method is by defining IPv6-only relays in the configuration file of the dual-stack sendmail. An example of this is to set the so-called ‘`LUSER_RELAY`’ in the dual-stack sendmail to the IPv6-only mail server in such a way that all email destined for email accounts not known at the dual-stack server are relayed to the IPv6-only mail server. This configuration for the dual-stack sendmail can be generated by adding a line lithe the following to the m4 macro file it is generated from:

```
define(`LUSER_RELAY', `relay:mail6.mydomain.tld')
```

The second method to relay email through a dual-stack sendmail server to an IPv6-only mail server (from IPv4-only hosts) is by setting up the MX records for the domain in question appropriately. Let us take as example the domain ‘mydomain.tld’ and assume that an (optional) IPv4-only mail server ‘mail4.mydomain.tld’, a dual-stack mailserver ‘mail64.mydomain.tld’ and an IPv6-only mail server ‘mail6.mydomain.tld’ have been set up for this domain.

An appropriate DNS configuration should then list MX records in the following order of priority:

```
mydomain.tld.      IN MX  0 mail6.mydomain.tld.
mydomain.tld.      IN MX  10 mail64.mydomain.tld.
mydomain.tld.      IN MX  20 mail4.mydomain.tld.
```

In other words, the IPv6-only mail server(s) should have lowest priority code (meaning highest priority mail server), followed by the dual-stack mail server(s) and finally the IPv4-only mail server(s), if any, should be given the highest code(s) (and thus the lowest priority). Of course, mail6.mydomain.tld should have an AAAA DNS record, mail64.mydomain.tld should have both an AAAA and an A DNS record, while mail4.mydomain.tld would only have an A record in DNS.

IPv4 clients will then send email to (one of) the IPv4 capable mail servers, which will relay it to (on of the) IPv6-only server(s). In the example above, an IPv4 client will send its mail to the first mail server on the list that has an IPv4 address: the dual-stack server mail64.mydomain.tld. Only if mail64.mydomain.tld is unreachable or down will it (try to) send the email to mail4.mydomain.tld. After mail64.mydomain.tld has received the email it will relay it to the higher priority mail server mail6.mydomain.tld, which is IPv6-only.

IPv6 capable clients will try to send their email for domain mydomain.tld directly to the IPv6-only mail server mail6.mydomain.tld. If this is (temporarily) unreachable those clients will instead use the dual-stack server mail64.mydomain.tld.

Note that most sites will want to have a backup mail server for both IPv6 and IPv4. This means that at least two of the mail servers need to be IPv6 capable and two of them need to be IPv4 capable as in the example above.

7.2.5. The totd DNS-Proxy (Linux/Unix)

The compilation of totd is quite straightforward. After unpacking the sources in any directory on the totd server, it can be configured and compiled by issuing the following commands. Preferably, compilation is done as an unprivileged user.

```
      # ./configure --prefix=/usr/local
      # make depend
      # make
```

Then, as root, totd may be installed with the command:

```
# make install
```

The configuration file /usr/local/etc/totd.conf has to be edited to include the following lines:

```
forwarder 3ffe:400:10:100:201:2ff:feb5:3806
prefix fec0:0:0:ffff::
totuser totuser
```

Explanation of the lines above:

- `forwarder`: IP address of parent DNS to query. One may also append a port number by adding "`port <portnumber>`" to use a specific port. It is possible to use multiple forwarders which will be queried in descending order if the previous DNS server failed to reply.
- `prefix`: prefix that totd needs to prepend to converted IPv4 addresses. If multiple prefixes are given, totd will assign them to converted addresses in a round-robin manner.
- `totuser`: userid which totd uses after dropping root-privileges. For security reasons it is strongly suggested to use this option. One can add a user called "totuser" by issuing the following command as root:

  ```
  # useradd -c "totd User" -s /bin/false totuser
  ```

Note: It is prudent to additionally change the password of totuser if the Linux server which will be running totd does not disable accounts that did not get an initial password. This can be achieved by using the following command

```
# passwd totuser
```

There are various other options which may be set in totd.conf but do not play an important role in this example. One can refer to the totd.conf.sample in totd's source tree for information on these additional options.

One may want to add an init-script to /etc/init.d to start totd automatically at system boot. The Linux distribution's documentation should elaborate on how to do this. Most distributions provide sample init-scripts or init-script skeletons which are very useful for building this init-script.

The daemon totd is started by calling:

```
# /usr/local/sbin/totd
```

With "ps -af | grep totd" one can verify that totd is actually running.

7.3. TRT

7.3.1. pTRTd and totd on a Linux router

7.3.1.1 Prerequisites:

For this example it is assumed that there is a Linux router which has two interfaces (eth0 and eth1) and which serves as an edge router to an IPv6-only subnet on eth1 which has the following prefix:

```
3ffe:400:10:110::/64
```

The Linux router itself has the IPv6 address 3ffe:400:10:100:250:4ff:feec:b3b3. To provide connectivity to IPv4-hosts for the subnet mentioned above, it will be shown how a TRT setup using pTRTd and totd is implemented. It is also assumed that there is a DNS server available under the IP 3ffe:400:10:100:201:2ff:feb5:3806. This server will be used to resolve DNS queries done by clients with IPv6-only connectivity and forwarded by totd.

7.3.1.2 Overview of installation steps

The following steps summarize the configuration work that has to be performed to provide TRT functionality on the Linux router for the 3ffe:400:10:110::/64 subnet:

- Installation of totd on a dedicated host. Not that this host does not necessarily have to be a dual-stack host nor does it have to be the same host that pTRTd will run on.
- Configuration of totd to use existing DNS and to prepend a specific prefix to converted IPv4 addresses.
- Starting totd.
- Check that all prerequisites for the installation of pTRTd on the Linux router are met.
- Installation of pTRTd and starting it.

7.3.1.3 Installation of totd

Please refer to section 7.2.5 on how to install and configure totd.

Totd does not necessarily need to be installed on a dual-stack host or on the (Linux/Unix flavor) router that will be running pTRTd. However, the server hosting totd has to have IPv6-connectivity because it has to be reachable from an IPv6-only subnet. If there is an IPv6-capable DNS available to the totd sever, there is no need for it to have IPv4-connectivity. Otherwise, IPv4-connectivity is needed for contacting a DNS with only an IPv4 stack.

7.3.1.4 Installation of pTRTd

There are a few requirements that have to be fulfilled before pTRTd can be used:

- tun/tap driver support needs to be present in the Linux kernel (Version 2.2 or 2.4) either as a module or compiled into the kernel itself.
- A working totd server has to be present to provide translated IPv4 addresses. Totd has to be configured to prepend the prefix fec0:0:ffff:: to converted IPv4 addresses. Note that fec0:0:ffff:: is a site-local prefix. It does not make sense to use global prefixes.
- /sbin/ip has to be present to allow pTRTd to set up an interface and a route.

Compilation and installation of pTRTd is trivial. After unpacking the sources on the Linux router, configuration and compilation of pTRTd is done by issuing the following commands:

```
# ./configure --prefix=/usr/local
# make
```

As root, pTRTd may be installed to /usr/local by running:

```
# make install
```

The usage of pTRTd is as follows:

```
# ptrtd [-i [<driver>:]<interface>] [-p <prefix>] [-l <prefix length>]
```

`<prefix>` defaults to fec0:0:0:ffff::/64 which means that there is no need to give the "-p" option if totd was configured to use this prefix. In general, there should be no need to give any options when starting pTRTd if TRT is set up according to this description. PTRTd is run by simply starting the daemon:

```
# /usr/local/sbin/ptrtd
```

If possible, one should perform a few checks to see whether or not pTRTd came up properly:

- `"ps -ef | grep ptrtd"` shows whether pTRTd is indeed up and running.
- `"ip link show"` verifies that a tap0 interface came up after starting pTRTd.
- `"ip route show"` checks whether or not there is a route which routes fec0:0:0:ffff::/64 via the tat0 interface mentioned above.

7.3.1.5 Configuring clients in the IPv6-only subnet

It is necessary to configure all clients in the IPv6-only subnet to use the totd-server as DNS or otherwise no proper translation of IPv4 addresses is done.

To test the configuration, one can use an IPv6 enabled application and try to contact a server that is normally only reachable via IPv4, e.g. start an IPv6-capable browser like Mozilla and point it to a web server that only has an IPv4 address. Ideally, the browser is configured not to use any proxies for testing purposes. If it still can display the pages served by the IPv4-only server, the TRT-installation was successful.

7.3.2. NTPD Time Server as a Proxy

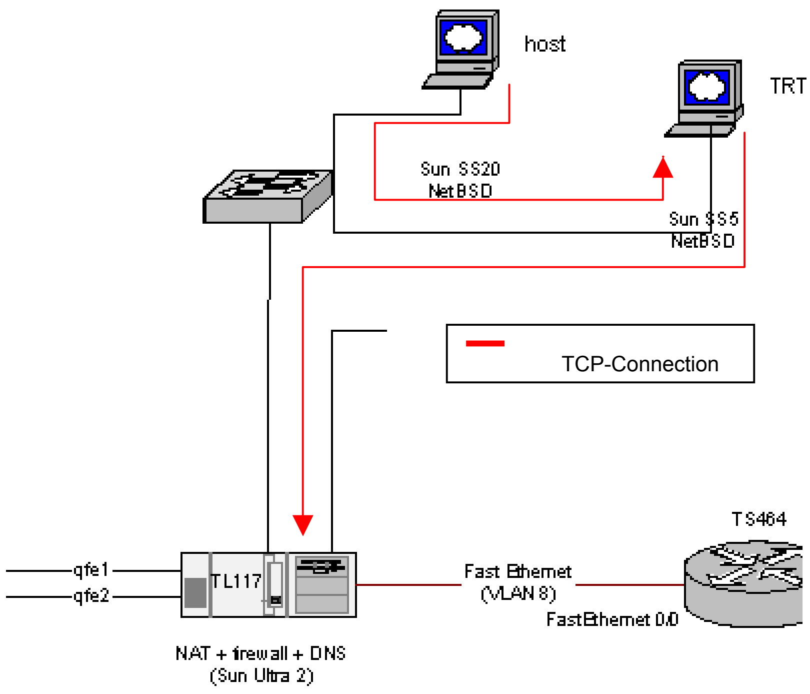
Currently the ntpd time server does not support IPv6 in its latest release. However, the next release will support IPv6 and it can already be tested by fetching the latest ntpd development sources. Building, installing and configuring ntpd with IPv6 support requires nothing special; it will simply listen for requests over both IPv4 and IPv6 and it can talk to either IPv4 or IPv6 ntpd servers.

7.3.3. The Faith TRT for FreeBSD and NetBSD

The transport relay translator faithd [RFC3142] is an IPv6-to-IPv4 TCP relay. Faithd relays TCP (not UDP, RTP or other transport protocols) from IPv6 to IPv4 (*not* the other way around). The faith daemon needs to be run on a dual-stack router between you local IPv6 site and outside IPv4 network. The daemon needs to be invoked per TCP service (TCP port number).

7.3.3.1 Example Setup

This example setup consists of two hosts on the same link running NetBSD software which are connected to a gateway. One of the hosts will be the TCP translator of the connections that the other host tries to establish. Once the TCP translation has been performed, the TCP connection will follow the normal way through the gateway.



7.3.3.2 Installation

The installation of the TRT consists in running a daemon called *faithd* included with the NetBSD software. This daemon must be run for every service we want to provide. To be able to run the daemon, previously is necessary to enable an interface called *faith*.

When the *faithd*[2] daemon receives TCPv6 traffic from the *faith* interface it will perform the translation to TCPv4. The destination of the TCPv4 connection will be determined by the last 4 octets of the original Ipv6 destination. It is necessary to reserve a prefix to perform this service. For example, if the prefix 3ffe:400:10:110:: is reserved and the Ipv4 destination is 10.1.1.1, the original destination should be 3ffe:400:10:110::0a01:0101. The address translation can be performed by hand but of course you are better off using a DNS proxy such as totd (see section 7.2.5).

The *faith*[3] interface captures IPv6 traffic for implementing IPv6 to IPv4 TCP translations. To be able to use this interface is necessary to recompile the kernel, enabling the pseudo-interface *faith*.

The following example shows how to use faithd to provide a TCP relay for the ssh service using 3ffe:400:10:110::/64 as faith prefix.

Perform the following steps on the router that will run the faith relay service:

1. If there already is an IPv6 TCP server for the "ssh" service, i.e. sshd, disable this daemon.
2. Execute the following as root to enable faith support:

```
# sysctl –w net.inet6. ip6.accept_rtadv=0
# sysctl –w net.inet6.ip6.forwarding=1
# sysctl –w net.inet6.ip6.keepfaith=1
```

Route packets with destination address within the faith prefix to the "faith0":

```
# ifconfig faith0 up
# route add –inet6 3ffe:400:10:110:: -prefixlen 64 ::1
# route change –inet6 3ffe:400:10:110:: -prefixlen 64 –ifp faith0
```

3. Start "`faithd`" as root as follows:

```
# faithd ssh /usr/sbin/sshd sshd -1
```

More examples that where successfully tested by the authors:

```
# faithd ftpd /usr/libexec/ftpd ftpd –l
# faithd sshd
# faithd telnet
# faithd telnet /usr/libexec/telnetd telnetd
# faithd smtp
# faithd www
# faithd https
# faithd irc
```

[2] More info about *faithd* in the manual pages.

[3] More info about *faith* in the manual pages.

```
# faithd icqs
# faithd pop3
# faithd nntp
```

If inetd(8) on your platform has support for faithd, it is possible to setup faithd service (almost) like any other service started from inetd and configure it in /etc/inetd.conf. At least recent FreeBSD releases have included support for this.

On NetBSD one can make the above example setup permanent with automatic configuration at boot time. One simply creates a configuration file /etc/ifconfig.faith0 including the following lines:

```
# pseudo interface for IPv6/IPv4 transport relay
create
inet6 3ffe:400:10:110:: prefixlen 64
```

Also add the flowing line(s) to /etc/sysctl.conf:

```
net.inet6.ip6.keepfaith=1
net.inet6.ip6.forwarding=1
# in case you don't want to do regular IPv4 forwarding at all:
net.inet.ip.forwarding=0
```

In addition it is strongly recommended to limit access to the translator. One way to do so is (at least on NetBSD) by creating a /etc/faithd.conf file restricting allowed connections. In the following example we assume that 3ffe:400:10::/48 is the address space in use at the site:

```
# permit anyone from our site to use the translator, to connect to
# the following IPv4 destinations:
# any location except 10.0.0.0/8 and 127.0.0.0/8
# Permit no other connections.
#
3ffe:400:10::/48 deny 10.0.0.0/8
3ffe:400:10::/48 deny 129.168.0.0/16
3ffe:400:10::/48 deny 127.0.0.0/8
3ffe:400:10::/48 permit 0.0.0.0/0
```

7.3.3.3 Problems

The main problem in this platform is the scalability. The address resolution used in this case is made by the *hosts* table. That means that every server we want to access should be one entry of the *hosts* table. We can try to solve this problem using a special DNS server called totd, which has not been used in this platform.

One important problem with the TRT to be able to provide HTTP is the absolute links in the web pages, because the TRT can not parse them. One solution to that problem could be an ALG (Application Layer Gateway) which will manage with the absolute links properly. That problem makes TRT not recommended in HTTP services.

8. Appendix A -- Availability of tools and mechanisms

This Chapter will include a list of implementations (where to get them) of the mechanisms and tools described above and maybe those that could not be covered in the configuration examples. Also platforms that support IPv6 will be mentioned. This list is not yet complete, but it will be updated in the next version of the cookbook.

8.1. Configured Tunnel

There is no known platform that has IPv6 support and doesn't also offer the feature of setting up IPv6-in-IPv4 tunnels.

8.2. Tunnel Broker

8.2.1. OpenLDAP/ssh-based tunnel broker at University of Southampton.

8.2.2. IPv6tb (Tunnelbroker by CSELT/Telecom Lab Italia)

The software is available for download at:

http://carmen.ipv6.tilab.com/ipv6/download.html

8.3. Automatic Tunnels

To make a list for all those implementations, which support this transition mechanism is rather fruitless. There is no known platform that implements IPv6, that does not have this feature, at least none, the workpackage participants know of. Also the use of this mechanism is no longer recommended. It has been replaced by much more sophisticated mechanisms.

8.4. 6to4

8.4.1. Cisco IOS

The feature "IPv6 Tunnelling: Automatic 6to4 Tunnels" is currently available in the following releases of Cisco IOS:

GD Release: 12.3(2)T4

LD Release: 12.3(5)

ED Releases: 12.3(4)XD, 12.3(4)T2, 12.3(3)B, 12.3(2)XE, 12.3(2)XC, 12.3(2)XB1, 12.3(2)XA2, 12.3(2)T3, 12.3(1a)BW, 12.3(1a)B, 12.2(20)S, 12.2(17a)SX1, 12.2(15)ZL1, 12.2(15)ZJ3, 12.2(15)T9, 12.2(14)ZA5, 12.2(14)SX1a, 12.2(14)S6, 12.2(13)ZF2, 12.2(13)ZE, 12.2(13)ZD, 12.2(13)T9, 12.2(11)YV1, 12.2(11)YU, 12.2(11)YT2, 12.2(11)YQ, 12.2(11)T9, 12.2(8)TB8, 12.2(8)YN, 12.2(8)T9

8.4.2. ExtemeOS

Extreme provides a special IPv6 Software for its Switches to support IPv6. This Add-on “ExtremeWare IPv6” contained the 6to4 feature from the beginning.

8.4.3. WindowsXP

Though newer service packs have changed handling and behaviour of the mechanism, WindowsXP contains 6to4 support out of the box.

8.4.4. Windows2000

Windows 2000 contains 6to4 support when the “Microsoft IPv6 Technology Preview” is installed.

8.4.5. Linux

The USAGI kernel patches contained 6to4 support since snapshot 2001/02/19. The first official vanilla kernel sources included 6to4 tunnelling with versions 2.2.19 and 2.4.6 respectively.

8.4.6. *BSD/Mac OS X

The KAME stack first included 6to4 March 2000. Official *BSD releases have KAME code built in since FreeBSD 4.0, NetBSD 1.5, OpenBSD 2.7 and BSD/OS 4.2. MacOS contains IPv6 functionality and with that Support for the stf interface since Version 10 (aka Version X Jaguar).

8.5. 6over4

Other than an implementation by Microsoft and an early inclusion of this mechanism into EFTs of the Cisco IOS (based on 11.3T, but which was later removed due to no customers making use of it), we know of no current implementations of this transition mechanism.

8.6. ISATAP

8.6.1. Cisco IOS

ISATAP support is currently available in the following versions of Cisco IOS:

GD Release: 12.2(15)T10

LD Release: 12.3(5)

ED Release: 12.3(4)XD, 12.3(4)T, 12.3(3)B, 12.3(2)XB, 12.3(2)XA2, 12.3(2)T2, 12.3(1a)BW, 12.3(1a)B, 12,2(20)S, 12.2(17a)SX1, 12.2(15)ZJ3, 12.2(15)T9, 12.2(14)SZ4, 12.2(14)SX2, 12.2(14)S6

Linux

ISATAP support has been included in the USAGI package from snapshot 20011112 on. The first stable release to include ISATAP was based on kernel 2.4.18 (2.2.20) and became available on April 1st, 2002.

There is to date no Linux distribution that includes ISATAP support in its preconfigured kernel.

8.6.2. Windows

Windows XP can be configured as ISATAP client since Service Pack 1. .NET/Windows 2003-Server can be configured as both client and server for ISATAP.

8.6.3. BSD

Due to license requirement owing to its IPR (draft-ietf-ngtrans-isatap-13.txt) KAME is no longer working on ISATAP in their stack and have removed it after the draft was released in March 2003. Older Versions of the KAME stack included ISATAP support since January 2003, when KAME began to work on implementing draft-ietf-ngtrans-isatap-08.txt.

8.7. Teredo

Windows 2003 Server includes a Teredo implementation.

Teredo is included in the Advanced Networking Pack for Windows XP SP1.

It was used by the Microsoft ThreeDegrees peer-to-peer trial.

8.8. Tunnel Setup Protocol (TSP)

8.9. DSTM

The ENST team has an implementation: http://www.ipv6.rennes.enst-bretagne.fr/dstm.

The 6TALK initiative from ETRI has released a DSTM implementation, including a Windows client, see: http://www.6talk.net/dstm/

8.10. TRT

Currently only a few known TRT implementations exist. One is FAITH [FAITH]. The program comes as part of the KAME project and has been written by Yoshinobu Inoue and Jun-ichiro itojun. For more information on FAITH please refer to:

http://www.kame.net/newsletter/19981001

Another TRT implementation is The Portable Transport Relay Translator Daemon (pTRTd) which is also a part of the KAME project. It has been implemented by Nathan Lutchansky and can be found via

http://v6web.litech.org/ptrtd/

9. Appendix B: Enabling IPv6

9.1. MS Windows XP

Windows XP embeds IPv6 functionality by default, as the TCP/IP protocols suite includes both IPv4 Internet Layer and IPv6 Internet Layer. However, Windows XP contains a separate implementation of TCP and UDP for IPv6. Figure 9-1 shows the differences between a typical Dual IP Stack and MS Windows XP implementation.

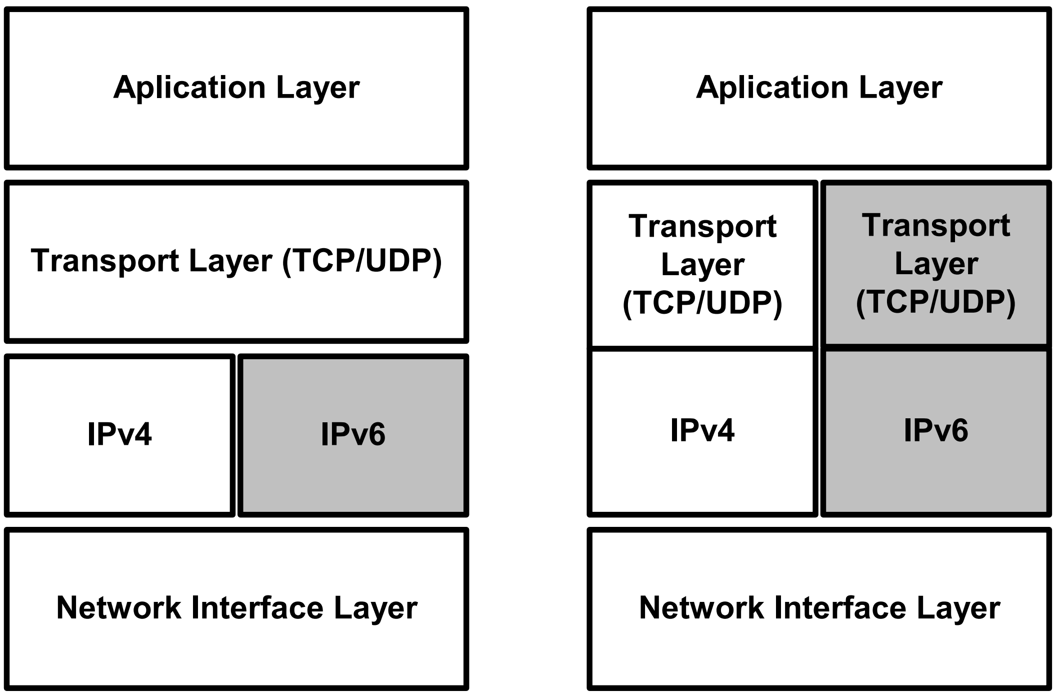


Figure 9-1:Left: typical IP dual stack layers, right: Windows XP IP layer stack implementation

By typing at the command prompt

```
# ipv6 install
```

it will enable and initialize IPv6 functionality on your host. By default, Windows XP automatically configures the link-local address for each interface that corresponds to installed Ethernet adapters. Link-local addresses have the prefix FE80::/64. The IPv6 address for each Ethernet interfaces derives from the FE80::/64 prefix and a 64-bit suffix that derives from the IPv4 plus MAC addresses of the network adapter. For example, a host with IPv4 address 195.225.29.15 and MAC address 00-00-39-3f-7b-90 is assigned the link-local address fe80::200:39ff:fe3f:7b90. The host is now ready to communicate with other host in the same Ethernet segment.

Windows XP supports stateless address autoconfiguration mode, with which network site-local addresses, route entries and other configuration parameters are automatically configured based on the router message advertisements. However, Windows XP hosts may manually be configured through the network configuration shell "netsh" and following set of commands:

```
# netsh interface ipv6 {add, set, delete, …} <parameters>
```

The netsh shell allows the configuration of the IPv6 addresses, the manipulation of the route entries and many other tuning and showing configuration commands. For example, the following command sets the site local address 2001:648:220::1 to the interface "Local Area Connection":

```
# netsh interface IPv6 add address "Local Area Connection" 2001:648:220::1
```

The complete syntax of the netsh command shell is the following:

```
# netsh interface IPv6 add address InterfaceNameOrIndex \
     IPv6Address [[type=] unicast|anycast] \
     [[validlifetime=]Minutes|infinite] [[preferredlifetime=] \
     Minutes|infinite][[store=]active|persistent]
```

9.2. MS Windows 2000

A pre-production version of IPv6 stack can be enabled at a host running Microsoft Windows 2000 (Service Pack 1 or greater) by installing the "Microsoft IPv6 Technology Preview" software, which is freely distributed at: http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp. Initially, the software should be saved locally and extracted. Through the "Control Panel Network & Dialup Connection", the IPv6 software is installed as new network protocol to an Ethernet interface. Automatically, the IPv6 stack is enabled to all Ethernet interfaces of the host and the new files are copied into the appropriate operating directories. Also, some network programs such as "*telnet.exe*" are updated in order to support IPv6 functionality. Note that special care should be given when installing IPv6 stack to Windows 2000 hosts with Service Pack 2 or 3 installed. In such cases, a fixing program should be executed before the installation of the "Microsoft IPv6 Technology Preview" software, as it is detailed documented in the relevant FAQ file.

The "Microsoft IPv6 Technology Preview" software includes various command line utilities that are used for tuning and confirming the state of the IPv6 configuration in the Windows 2000 host. A short description of some of the installed utilities follows:

- **net.exe:** Utility that stops or starts the IPv6 protocol and unloads/loads it from/to the memory. The relevant commands are "net stop tcpipv6" and "net start tcpipv6".
- **ipv6.exe:** Basic utility that configures network interfaces and updates the routing table. It also retrieves and displays information about the IPv6 protocol.
- **6to4cfg.exe:** Utility that sets up and configures a 6to4 tunnels.

- **ping6.exe, tracert6.exe:** The IPv6 version of the well-known utilities.
- **ttcp.exe:** Utility that sends TCP or UDP data between two networks nodes.
- **ipsec.exe:** Utility that configures policies and security associations for IPv6 IPSec traffic.

The IPv6 stack implementation in Windows 2000 supports stateless address autoconfiguration. Therefore, if the site-local IPv6 router is configured to advertise the network prefixes, the Windows 2000 host can automatically configure its IPv6 address and routing table. Usually, this is sufficient for the host to be connected to the IPv6 network and to be able to communicate with other hosts using IPv6 protocols. In stateful mode, however, additional configuration of the IPv6 stack is required through the command line utilities, especially the "*ipv6.exe*" tool. Some of the basic configuration commands are presented in the next paragraphs:

```
ipv6 if [if#]:
```

It provides a view of the existing IPv6 enabled interfaces .

```
ipv6 adu <int no>/<ipv6 addr>[lifetime VL[/PL]] [anycast] [unicast]:
```

It adds an <ipv6 addr> address to the interface <int no>. The lifetime parameter specifies how long the address will be valid. If this parameter is not specified, lifetime is infinite. If this parameter is set to zero, the IPv6 address is removed. In the following example, IPv6 address 2002:968c:152d::968c:152d is assigned to the interface 2 (tunnel pseudo-interface):

```
        # ipv6 adu 2/2002:968c:152d:: 968c:152d
```

```
ipv6 ifc if# [forwards] [advertises] [-forwards] [-advertises] [mtu #bytes]
[site site-identifier]:
```

It configures special attributes of an IPv6 interface. It enables an interface to forward all packets whose destination address is not assigned to the interface or to send/receive router advertisements. Also, it sets the MTU size and site-identifier of an interface.

```
ipv6 rtu prefix if#[/nexthop] [lifetime L] [preference P] [publish] [age] [spl
site-prefix-length]:
```

It adds a route entry to the routing table. Routing entry time-to-live and preference are also set. For example in the following example, a default static route (::/0) is added to through interface 2 and the next hop address is set to ::131.107.152.32 (IPv4 tunnel destination IP address of a manually configured tunnel):

```
      # ipv6 rtu ::/0 2/::131.107.152.32
```

```
ipv6 rt:
```

It displays the contents of the routing table.

```
ipv6 nc [if# [address]]:
```

It displays the total neighbor cache of a host or an interface.

It should be noted that IPv6 configuration is not saved permanently. Therefore, each time the Windows 2000 hosts reboots, it requires reconfiguration for the IPv6 protocols. Therefore, it is advised the entire configuration to be kept in an executable start-up batch file.

9.3. Sun Workstation with Solaris 8

Solaris 8 and its later versions fully support IPv6 protocols. The IPv6 implementation incorporates all basic services and functionality of the protocol and it supports tunneled interfaces, such as IPv6 over IPv4. In previous versions of the Solaris operating system, IPv6 is supported only by installing specific protocol patches.

IPv6 is enabled in Solaris 8 during the installation procedure. Otherwise, the following installation steps should be followed:

1. Create an empty file named /etc/hostname6<interface>, where <interface> is the interface name in which IPv6 will be enabled.
2. Reboot the system
3. Execute the following command to all the IPv6-enabled interfaces:

   ```
   # ifconfig <interface> inet6 plumb up
   ```

4. Execute the scripts located at /etc/init.d/inetinit.

After following the above procedure, the Solaris 8 host automatically start the Network Discovery daemon, which will probe the local router for an IPv6 address and other configuration parameters.

For Solaris 8 and 9, you also need to edit /etc/nsswitch.conf to enable DNS resolution for the ipnode entry.

9.4. FreeBSD

FreeBSD natively supports IPv6 functionality in its kernel. A FreeBSD host enters to the IPv6 stateless mode by just adding the option ipv6_enable="YES" in the /etc/rc.conf configuration file.

This forces the system to listen for router advertisement in order to set up the IPv6 interfaces and other configuration parameters. In the following output logs, the FreeBSD host picked up two different addresses as there were multiple router advertisements on the same broadcast domain.

```
# ifconfig -a
     fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
       inet 147.102.220.1 netmask 0xffffff00 broadcast 147.102.220.255
       inet6 fe80::203:47ff:fece:3cee%fxp0 prefixlen 64 scopeid 0x1
       inet6 3ffe:2d00:2:220:203:47ff:fece:3cee prefixlen 64 autoconf
       inet6 2001:648:2:220:203:47ff:fece:3cee prefixlen 64 autoconf
        ether 00:03:47:ce:3c:ee
       media: Ethernet autoselect (100baseTX <full-duplex>)
       status: active
```

In the stateful mode, an IPv6 interface is configured through the options in the /etc/rc.conf file. For example, assuming that the IPv6 interface is fxp0 and the IPv6 address is 2001:648:2:220*,* the configuration file should like as follows:

```
ipv6_enable="YES"
ipv6_network_interfaces="fxp0"
ipv6_prefix_fxp0="2001:648:2:220"
ipv6_ifconfig_fxp0="2001:648:220::1 prefixlen 64"
ipv6_defaultrouter="2001:648:220::0"
ipv6_prefix_fxp0="3ffe:2d00:2:220"
ipv6_ifconfig_fxp0="3ffe:2d00:220::1 prefixlen 64"
ipv6_default_interface="fxp0"
```

9.5. Redhat

Enabling IPv6 to Redhat kernel requires either to recompile the kernel with IPv6 support or load an appropriate module (without recompiling the kernel). Redhat provides such a module on its Linux distributions with versions greater than 7.3. The IPv6 module is loaded by using the *"modprobe"* command, e.g. modprobe IPv6, or by adding the following line to /etc/sysconfig/network configuration file.

```
"NETWORKING_IPV6=yes"
```

A Redhat system enters to the stateless mode by adding the following option in the /etc/sysconfig/network-scripts/ifcfg-eth0 configuration file[4]:

```
    "IPV6INIT=yes"
```

This option forces the Redhat host to listen for router advertisement in order to set up the IPv6 interfaces.

In the stateful mode, the following modifications is required to the /etc/sysconfig/network-scripts/ifcfg-eth0 configuration file:

```
    "IPV6ADD =  3FFE:2D00:1::1 /10"
```

[4] eth0 should be replaced with the current interface that one uses to connect to the Internet e.g. ppp0 if you use dial-up

10. References

[6TO4SEC]	"Security Considerations for 6to4", P. Savola, Internet Draft, draft-ietf-v6ops-6to4-security-02.txt; March 2004.
[BIA]	"Dual Stack Hosts Using 'Bump in the API' (BIA)", S. Lee, M. Shin, Y. Kim, E. Nordmark, A. Durand, RFC 3338; October 2002.
[CONSIDER]	"Considerations for IPv6 Tunneling Solutions in Small End Sites", T. Chown, Internet Draft, draft-chown-v6ops-unmanaged-connectivity-00; October 2003.
[D6.2.2]	6NET Deliverable 6.2.2: "Operational procedures for secured management with transition mechanisms"; February 2003.
[DHCPv6]	"Dynamic Host Configuration Protocol for IPv6", Bound, Carney, Perkins, Volz, Lemon, Droms (ed.), RFC 3315; July 2003.
[DSTM]	"Dual Stack Transition Mechanism (DSTM)", Bund, Toutain, Medina et al., Internet Draft; draft-ietf-ngtrans-dstm-08.txt; July 2002.
[DSTM_TSP]	"DSTM Tunnel Setup using TSP", Blanchet, Medina, Parent, Internet Draft, draft-blanchet-ngtrans-tsp-dstm-profile-01; July 2002.
[DSTM_DHCPv6]	"DSTM Ports Option for DHCPv6", Myung-Ki Shin, Internet Draft, draft-ietf-dhc-dhcpv6-opt-dstm-ports-00.txt; June 2002.
[DSTM_VPN]	"DSTM in a VPN Scenario", Richier, Medina, Toutain, Internet Draft, draft-richier-dstm-vpn-00.txt; February 2002.
[ENST]	ENST DSTM web site: http://www.ipv6.rennes.enst-bretagne.fr/dstm/.
[ENT-SCEN]	"IPv6 Enterprise Network Scenarios", J. Bound, Internet Draft, draft-ietf-v6ops-ent-scenarios-01; November 2003.
[FAITH]	"Translating IPv4 and IPv6 connections", Yoshinobu Inoue, Jun-ichiro Itojun Itoh, http://www.kame.net/newsletter/19981001/; January 17th 2003.
[IEEE-V6]	"A scenario-based review of IPv6 Transition Tools", M. Mackay, C. Edwards, M. Dunmore, T. Chown, G. Carvalho, IEEE Internet Computing; May/June 2003.
[ISATAP]	"Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)", F. Templin, T. Gleeson, M. Talwar, D. Thaler, Internet Draft, draft-ietf-ngtrans-isatap-20; February 2004.
[PARIS-IPV6]	"Review of IPv6 Transition Scenarios for European Academic Networks", Conference Paper for "IPv6 Conference in Paris", Tim Chown, Ming Feng, Mike Saywell; October 2002.
[RENUMBER]	"Procedures for Renumbering an IPv6 Network without a Flag Day", F. Baker, E. Lear, R. Droms, Internet Draft, draft-ietf-v6ops-renumbering-procedure-00; February 2004.
[RFC1631]	"The IP network Address Translator (NAT)", K. Egevang, P. Francis, RFC 1631; May 1994.

[RFC1928]	"SOCKS Protocol Version 5", M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones, RFC 1928; March 1996.
[RFC2365]	"Administratively Scoped IP Multicast", D. Meyer, RFC2365/BCP23; July 1998.
[RFC2529]	"Transmission of IPv6 over IPv4 Domains without Explicit Tunnels", B. Carpenter, C. Jung, RFC 2529; March 1999.
[RFC2765]	"Stateless IP/ICMP Translation Algorithm (SIIT)", E. Nordmark, RFC 2765; February 2000.
[RFC2766]	"Network Address Translation - Protocol Translation (NAT-PT)"; Tsirtsis, Srisuresh; RFC 2766; February 2000.
[RFC2767]	"Dual Stack Hosts Using the 'Bump-in-the-Stack' Technique", K. Tsuchiya, H. Higuchi, Y. Atarashi, RFC 2767; February 2000.
[RFC3056]	"Connection of IPv6 Domains via IPv4 Clouds", B. Carpenter, K. Moore, IETF RFC 3056; February 2001.
[RFC3068]	"An Anycast Prefix for 6to4 Relay Routers", C. Huitema, RFC 3068; June 2001.
[RFC3089]	"A SOCKS-based IPv6/IPv4 Gateway Mechanism". H. Kitamura, RFC 3089; April 2001.
[RFC3142]	"An IPv6-to-IPv4 Transport Relay Translator", J. Hagino, K. Yamamoto, RFC 3142; June 2001.
[TRANSSEC]	"IPv6 Transition/Co-existence Security Considerations", P. Savola, Internet Draft, draft-savola-v6ops-security-overview-02; February 2004.
[STEP]	"Simple IPv6-in-IPv4 Tunnel Establishment Procedure (STEP)", P. Savola, Internet Draft, draft-savola-v6ops-conftun-setup-02; January 2004
[TEREDO]	"Teredo: Tunneling IPv6 over UDP through NATs", draft-huitema-v6ops-teredo-01.txt, C. Huitema; February 2004.
[TRANSARCH]	"A View on IPv6 Transition Architecture", P. Savola, Internet Draft, draft-savola-v6ops-transarch-03; January 2004.
[TUNNEVAL]	"Evaluation of v6ops Tunneling Scenarios and Mechanisms", P. Savola, Internet Draft, draft-savola-v6ops-tunneling-00; March 2004.
[USAGI]	"USAGI (UniverSAl playGround for IPv6) Project – Linux IPv6 Development Project", http://www.linux-ipv6.org/.
[V6OPS]	IETF IPv6 Operations WG, http://www.ietf.org/html.charters/v6ops-charter.html
[VLAN-ID]	"Use of VLANs for IPv4-IPv6 Coexistence in Enterprise Networks", T. Chown, Internet Draft, draft-chown-v6ops-vlan-usage-00; October 2003

Some of these Internet Drafts did not reach RFC status and expired as an I-D. You may find some of them in an archive, e.g. http://www.join.uni-muenster.de/drafts/ or http://www.watersprings.org/.