Project Number:	**IST-1999-20393**
Project Title:	LONG *Laboratories Over Next Generation Networks*
Deliverable Type:	**P – public**

CEC Deliverable Number:	IST-1999-20393/ PTIN /WP2.1/DS/P/1/01
Contractual Date of Delivery to the CEC:	**M06 (31-May-2001)**
Actual Date of Delivery to the CEC:	**31-May-2001**
Title of Deliverable:	**Description of IPv4/IPv6 available transition strategies**
Work package contributing to the Deliverable:	**WP 2**
Nature of the Deliverable:	**R – Report**
Author(s):	Jacinto Vieira (PTIN), Francisco Fontes (PTIN), Alberto García (UC3M), Carlos Ralli (TID), Ruth Vázquez (TID), Josep M. Bafalluy (UPC), Alberto López (UPM), Mário Filipe (UE), Nis A. Clausen (TED).
Editor:	Jacinto Vieira (PTIN)

Abstract:	This document describes the Transition Mechanisms presented in the IETF organization and studies various scenarios where they can be applied. The main differences between the IPv4 and IPv6 protocols and the reason for such a transition are also discussed. The document evaluates each mechanism and describes the experiments for some implementations available**.**
Keyword List:	LONG, IPv6, Next Generation Networks, Advanced Network Services, Advanced Network Platforms.

Executive Summary

This document, the first deliverable for LONG's WP2 (Work Package 2), presents and discusses the IPv6 Transition Mechanisms (TM) and the scenarios where they are applicable. These TM are evaluated and the configurations of some implementation are described.

The study performed to elaborate this document is the basis for the continuation of the WP2, WP3 and WP4 activities such as the use of the services and applications in transition environments.

Table of contents

1. Introduction

This document, the first deliverable for LONG's WP2 (Laboratories over Next Generation Networks, Work Package 2), aims at presenting and discussing IPv6 transition mechanisms and scenarios and the reasons for such a transition.

In the LONG project, the theoretical study and tests of Transition Mechanisms are oriented to provide possible solutions from current through transition scenarios to the final and desired scenario: IPv6 only network. This transition will probably last for a few years.

It is known for a long time that IPv4 is not going to satisfy real needs, due to the huge increase of equipment and utilities that are going to be linked to the Internet in the future. The fact that IPv4 addresses are 32 bits long, and IPv6 addresses are 128 bits, makes the difference.

In chapter 2, the differences between IPv4 and IPv6 are explained and also the role of Transition Mechanisms (TM).

One advantage of IPv6 is that the new packet header will allow integration of security and other improved control features. It is also possible to support mobility without the need of re-configuration, at least for hosts. This means a great simplification in connecting, for instance, a laptop computer to the Internet network or any other device prepared to.

This document identifies the main differences between IPv4 and IPv6 and the interoperability of both in the following years. It is not an exhaustive theoretical study of each TM or their possible combination.

The mechanisms that guarantee a smooth transition from the IPv4 world into the IPv6 world are described in (Chap. 3), and their applicability is studied (applicable scenarios) in Chap. 4.

In Chap. 5, performance and management costs - direct and indirect - are evaluated and installation procedures are described for each scenario (in the cases that have been tested).

The information in this document will be continued and complemented by work developed during LONG WP2 and WP4 activities. Real transition scenarios are object of study in later deliverables, which are understood as a consequence and evolution of the information provided in this document.

2. The Case for Transition into IPv6

2.1 The Internet evolution

Internet has experienced a dramatic growth in the last years. Some studies state that public Internet is approximately doubling its traffic each year, being expected that in 2002 Internet traffic will surpass voice traffic [Coffman]. The growth is also spectacular in the number of hosts, with estimations for January 2001 of more than 109 million [ISC2001].

Until now, web is considered the main responsible for the traffic explosion, with mail exchange and data sharing as other important contributors.

In the future, e-commerce, voice and multimedia transmission will join traditional Internet services to foster bandwidth demand.

Regarding to the number of nodes, the integration of voice and data traffic, and the increase in the number of always-on devices for personal communication (mobile phones with IP capability) and home networking are expected to multiply the number of IP addresses required.

2.2 The IPv6 opportunity

IPv6 was born at the IETF as a new protocol to solve the shortage of IPv4 addresses. However, it has been shown that IPv4 has some additional limitations that needed to be solved.

Until now, these limitations had been solved by means of “patches” so that the current IP protocol could offer better address aggregation, Quality of Service, increased security (IPSec), mobility, and so on. These additional capabilities have been integrated into the new IPv6 protocol. The next section shows exactly how these new functionalities are accomplished.

Advances in fields like wireless communications, access network, and others, are making the people to demand a wider range of services which in turn, impose new requirements to the network layer.

A great increase in the number of Internet-connected devices is envisaged. The users needs are continuously evolving. The user wants to access a lot of services (web, videoconferencing, …) from his mobile device, with the proper security level from anywhere at anytime. IPv6 suits perfectly into such an environment where additional support for mobility, security, QoS and a wider address range is needed.

2.3 Main differences between IPv4 and IPv6

The main differences between IPv4 and IPv6 are explained in this section.

Addressing

IPv4 supports 32-bit field for addressing, which is no longer sufficient for the number of users on the Internet. With IPv6, the address format increases from 32 to 128 bits, allowing support for more nodes (hosts, routers or connection points), more levels of addressing hierarchy and simpler autoconfiguration.

Besides, IPv6 supports a new type of address called Anycast address. The Anycast address is an identifier assigned to more than one interface (usually into different nodes). When a packet is sent to an Anycast address, it is delivered to the nearest interface identified by that address. The concept of nearest interface is determined according to the routing protocol's measure of distance.

Also, there is a difference in the IPv6 multicast address because it introduces the concept of "scope". This new field in the multicast address was created to improve the scalability of multicast routing.

Header

To reduce the processing cost of packet handling, some IPv4 header fields have disappeared or made optional. Although the IPv6 addresses are four times longer than the IPv4 addresses, the IPv6 header is only twice the size of the IPv4 header.

The IPv4 headers can have variable length while IPv6 headers have a fixed length of 40 bytes. This allows the routers to optimize the processing of IPv6 headers. Besides, the processing is improved because of the reduced number of required header fields in IPv6.

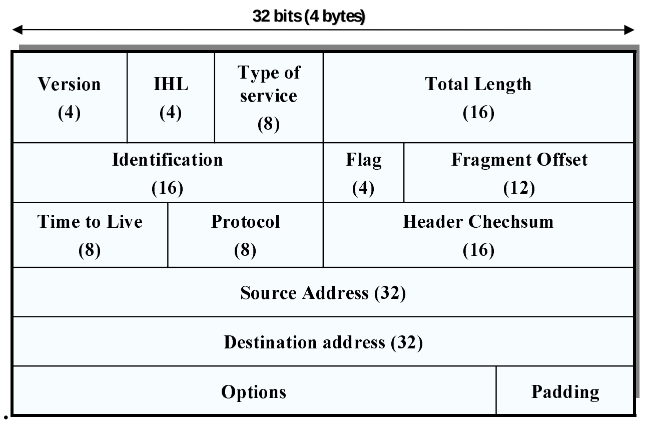

Figure 1 - IPv4 Header Format

As it has been said, the IPv6 header is simplified compared to the IPv4 header. The information not needed for routing is placed in separate extensions headers, located between the IPv6 basic header and the transport-layer header in a packet (in IPv4 has 14 fields and in IPv6 has only eight fields).

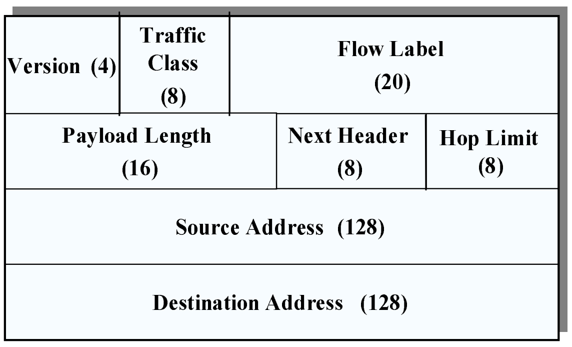

Figure 2 - IPv6 Header Format

One of the eliminated fields is exactly IHL – IP header length - due to the fixed header length of all IPv6 packets.

The other length field is maintained with 16 bits but this field does not include the length of the IPv6 header and it is now designated as "payload length field". The "payload length field" can support packets up to 64k-bytes (and even larger called jumbograms).

In addition to the IHL field, a number of basic IPv4 fields were eliminated from the IPv6 header: fragment offset, identification, flags and checksum. The fields related to the fragmentation (fragment offset, identification, flags) have been moved to extensions headers in IPv6. The "checksum" field has been eliminated because the error checking is duplicated at other levels of the protocol stack, namely, at the link and transport layers.

The TOS is replaced by "Traffic Class" and the new field was added, called "flow label" to support the QoS mechanisms.

The new name for TTL is now "hop limit". This field is used to eliminate "loops" by decrementing the value 1 for each hop of the end-to-end route. This field is set in the source node with appropriate value. When this value is decremented to zero, the packet is discarded.

In the IPv6 packet, the "Next Header" replaces the "Protocol Type" field. This field indicates the type of the next header, which can be the transport packet header or another IPv6 extension header.

An IPv6 full implementation would include the following extension headers:

Hop-by-hop Options

The Hop-by-Hop Options header is used to carry optional information that must be examined by every node between the source and the destination.

Routing Header

The Routing Header is used by the source node to "oblige" the packets to go through the intermediate nodes in the path until the destination.

Fragment Header

Source nodes that choose to fragment the IPv6 packets use the Fragment Header. This header contains fields that identify a group of fragmented packets and assigns them sequence numbers.

Destination Options Header

The Destination Options Header is used to carry optional information from the source node that needs to be examined only by destination node.

Authentication Header

Allows the certitude of integrity and authentication for IP datagrams.

IPv6 Encryption Header

This field announces the existence of the ESP (Encapsulating Security Payload) mechanism that guarantees integrity and confidentiality of IP datagrams.

Routing

The IPv6 addressing allows supporting more levels of addressing hierarchy. A hierarchical addressing system can be defined to decrease the size of the routing table. The routers can use IP address prefixes to determine the way the traffic should be routed through the network, which allows "route aggregation" at various levels of the hierarchy.

The IPv4 uses a technique called Class InterDomain Routing (CIDR) to allow the flexible use of variable-length network address. However, this technique does not guarantee an efficient and scalable hierarchy because of the legacy of IPv4 address assignments before the CIDR.

Scalability of multicast routing is improved by adding a scope field to multicast addresses.

Autoconfiguration

Autoconfiguration allows the creation of a link-local address, verification of its uniqueness and optionally autoconfiguration of other information. In IPv6, there are two possibilities: stateless and stateful autoconfiguration.

In the stateful configuration, hosts obtain their address and configuration from a DHCP server. The server maintains an updated database with addresses and interfaces and correspondent nodes.

In the stateless configuration [RFC2462], there is no need for manual configuration of hosts, just a little for routers and no need for additional servers. In this case, a host can generate its address combining local information and information advertised by routers, since routers advertise prefixes that identify the subnets associated with a link. In the absence of a router, a host can only generate link-local addresses.

Both, stateless and stateful autoconfiguration, can be used simultaneously. Routers can specify the hosts that should use stateful and stateless configuration using Router Advertisement messages [RFC2461].

Authentication, data integrity and encryption

Native authentication and data integrity options are part of IPv6 protocol and are required in all implementations. The encryption option (data confidentiality) is optional.

It has already been mentioned above that AH (Authentication Header) provides authentication and integrity, and, ESP (Encapsulating Security Payload) provides confidentiality and also authentication and integrity, if appropriate transforms and algorithms are used. Both applications running are a guarantee of a totally secure and controlled transmission.

Quality of Service

In IPv4, the Type of Service field (TOS) is used to define classes of services into the differentiated services architecture. A similar field is used in IPv6, called Class of Service (COS). Both fields have 8 bits.

A new field, Flow Label, has been created within the IPv6 header to distinguish traffic flows for optimized routing. This field with 20 bits may be used by traffic source to label sequences of packets for which it requests special handling by routers.

2.4 Role of transition mechanisms

During the transition phase, it is necessary to keep the IPv4 infrastructure working. In order to be compatible with IPv4 hosts and routers, there is a set of mechanisms that IPv6 hosts and routers should implement. These mechanisms are known as Transition Mechanisms (TM).

The transition mechanisms from IPv4 to IPv6 have been the object of studies in the way that they have to guarantee a smooth transition and upgrade.

Some of the prerequisites and conditions of these transition mechanisms are:

- IPv6 integration cannot disrupt the actual network;
- IPv4 functionality and connectivity must continue without changes;
- IPv4 performance cannot be affected with IPv6 installation; IPv6 will only affect its trial or installation environment;
- It is assumed that IPv6 initial performance will not match IPv4 performance in the first stage of deployment;

This process will take time and will depend on the speed of the introduction of IPv6 versions by vendors and developers, the market and the urge that network managers find to integrate those changes.

There is no need to make IP upgrades in all machines. The communication between IP connected devices (hosts, routers, other network equipment) is object of study and the reason for these transition mechanisms.

Some mechanisms may be more adequate for an early stage of transition (IPv6 islands in an IPv4 ocean) while others are more appropriate for intermediate or final transition stages. It is also believed that some mechanisms will be applied in all stages, and that the transition period will be long.

3. Transition mechanisms

3.1 Main Mechanisms

In the following sections, the standard transition mechanisms defined in different RFCs or drafts are explained.

3.1.1 Dual–Stack Nodes

The most basic mechanism for next generation IP transition is to provide a complete IPv4 stack and also a complete IPv6 stack on each host. Hosts of this kind are called dual stack nodes ant they can directly communicate with IPv4 hosts and with IPv6 hosts. This mechanism is defined in [RFC2893], “Transition Mechanisms for IPv6 Hosts and Routers”.

Because of their dual stack, this kind of nodes need at least an IPv4 address and an IPv6 address. The IPv4 address can be manually configured or it can be acquired through a DHCP server, as it is done in any traditional IPv4 network. The IPv6 address can be manually configured, but it can also be determined through IPv6 stateless/stateful configuration mechanisms.

As a consequence of its dual nature, the DNS resolver libraries of dual stack hosts must be able to handle A type records and A6/AAAA records. So, when a host needs to resolve a name into an address, the resolver may issue an A type query and/or an A6/AAAA type query. If an A type answer is received, the host will use the IPv4 stack, while if an A6/AAAA type answer is received, it will use the IPv6 stack. In the case that both answers are received, the resolver can choose which answer will be returned to the application, or again, both answers are returned to the application. In this last case, the resolver will establish an order, considering that the application will probably only take into account the first answer. This is a relevant issue to consider because it contributes to define the type of traffic transferred through the network.

As it was mentioned earlier, this is the most basic transition mechanism, so it is available in current IPv6 boxes. Just to name a few, this mechanism is available and stable in FreeBSD, Linux, Windowsand Solaris.

3.1.2 Tunneling Techniques

As can be already known, transition from IPv4 to IPv6 takes some time, it can not be done in one day. Tunneling techniques provide a way of utilizing current IPv4 infrastructure to carry IPv6, while the IPv6 infrastructure is deployed. [RFC2893] deals with the basics of tunneling techniques.

Tunneling consists of IPv6 traffic encapsulated in IPv4 and traversing IPv4 networks, i.e., if one IPv6 node wants to access other IPv6 node but there is an IPv4 ocean in between, then one dual IPv6-IPv4 node will establish a tunnel with other dual IPv6-IPv4 node. There is no need of both ends to be the source and destination of the traffic: if one wants to communicate the whole IPv6 network in one side with a whole IPv6 network at the other side of the IPv4 ocean, only one tunnel should be established at the boundary of each network, providing that the node at the border support both IPv6 and IPv4.

Nowadays tunneling techniques have in common an IPv4 infrastructure while the aim is to carry IPv6 traffic, however they usually differ in how the nodes determine the address of the node at the end of the tunnel.

In the future, tunneling techniques over IPv6 infrastructure are expected.

To establish any tunnel between two nodes, both need to have installed IPv6 and IPv4 protocols. Consequently they will have IPv4 and IPv6 addresses. IPv6-IPv4 nodes can acquire their addresses with IPv4 and IPv6 mechanisms. For example to acquire IPv4 addresses *DHCP* can be used, while to acquire IPv6 address *stateless address autoconfiguration* can be used. Since IPv6-IPv4 nodes have to interoperate with both IPv4 and IPv6 only nodes, they will need access to DNS capable of resolving IPv4 and IPv6 address.

3.1.3 Translation Techniques of IPv4 and IPv6

For translation between IPv4 and IPv6, there are three possible methods: header conversion, transport relay and application level gateway. In this section, these methods are described.

3.1.3.1 Header Conversion

Header conversion refers to converting IPv6 packet headers to IPv4 packet headers, or vice versa, according to a procedure of translation. A procedure of translation is defined in SIIT algorithm [RFC2765]. This procedure of translation is used in NAT-PT and BIS mechanisms and can be used in any other translation mechanisms based into headers conversion.

SIIT Algorithm

SIIT - Stateless IP/ICMP Translation Algorithm - is a transition mechanism algorithm, usually implemented on border routers, that translates between IPv4 and IPv6 packets headers, including ICMP headers. SIIT is a building block of a solution that allows IPv6 hosts, which do not have a permanent assigned IPv4 address, to communicate with IPv4-only hosts.

The mechanisms related to the addresses assignment or routing are not considered in this algorithm.

It is stateless because it operates independently on each packet and it does not retain any state information from one packet to another [RFC2765].

There are several situations where communication between IPv6-only hosts and IPv4-only nodes is required: e.g. a new IPv6-only network that needs to communicate with the IPv4 Internet; an existing IPv4 network where a large number of IPv6 devices are added, etc.

When an IPv4-to-IPv6 translator receives an IPv4 datagram addressed to a destination that lies outside of the attached IPv4 island, it translates the IPv4 header of that packet into an IPv6 header. Packet forwarding is based on the IPv6 destination address (in this method, although the original IPv4 header of the packet is removed and replaced by an IPv6 header, the transport layer header and data portion of the packet are not changed, except for ICMP packets). The same happens when an IPv6-to-IPv4 translator receives an IPv6 datagram addressed to an IPv4-mapped IPv6 address.

The IPv6 nodes that use the translator must have an IPv4-translated IPv6 address [RFC2765] while they are communicating with IPv4-only nodes.

SIIT algorithm uses IPv4-mapped IPv6 addresses, defined in [RFC2373] “IP version 6 Addressing Architecture”, which format is 0::FFFF::A.B.C.D. These addresses are used to refer IPv4 only nodes. In addition to mapped addresses, SIIT defines IPv4-translated IPv6 addresses, which format is 0::FFFF::A.B.C.D. These addresses are assigned to IPv6 capable nodes, so that it can be referred from an IPv4 only node, using just A.B.C.D. IPv4-translated addresses are defined to avoid the use of IPv4-compatible IPv6 addresses outside the scope of automatic tunneling.

Because of the way IPv6 protocol is specified, the TCP and UDP pseudo-header checksums are not affected by the translations. The ICMPv6 includes a pseudo-header checksum, that does not exist in ICMPv4, and for this reason, the checksum in ICMP messages needs to be modified by the translator. The ICMP error messages contains an IP header as part of the payload, thus, the translator needs to rewrite those parts of the packets to make the receiver be able to understand the included IP header [RFC2463] .

Part of the IPSec functionality is maintained by this translator. Packets with ESP (Encapsulating Security Payload) format can be translated since ESP does not depend on header fields prior to the ESP header, but the correctness of the AH (Authentication Header) is not always preserved through a translator (for example, when there is an IPv6 endpoint that computes AH on received packets that have been translated from IPv4 packets).

All ICMP messages that are to be translated require the ICMP checksum field to be updated as part of the translation; and for ICMP error messages, the included IP header also needs translation.

SIIT algorithm does not provide a translation for IPv4 header options, IPv6 routing headers, hop-by-hop extensions headers nor destinations options headers.

<u>How the mechanism works</u>:

- **IPv4 to IPv6 translation**

Fragmentation issues are a main concern for SIIT, because as MTU discovery is mandatory in IPv6, only end-to-end fragmentation is allowed in the IPv6 protocol, in contrast with IPv4, where MTU discovery is optional. So if the originating IPv4 node performed MTU discovery, which it can work fine end to end, across the translator, no problems would arise. But if the originating IPv4 node does not perform MTU discovery, the translator must ensure that the packet does not exceed the MTU of the IPv6 side of the path. This is done by fragmenting the IPv4 packet into 1280 bytes IPv6 packets, which is the minimum packet size required in IPv6.

IPv4 header translation:

Field	**IPv4**	**IPv6**
Version	4	6
Traffic Class Class Of Service (COS)	Xxxxxxxx	Xxxxxxxx (ToS and preference bits copied)
Flow label	N/A	0
Payload length	X	X-size(IPv4 header)-size(IPv4 options)
Next header (Protocol in IPv4)	X	X
Hop limit	X	X-1
Source address	A.B.C.D	::FFFF:A.B.C.D
Destination address	E.F.G.H	::FFFF:0:E.F.G.H

ICMPv4 header translation:

Type	Message	ICMPv4 type field	ICMPv6 type field	Further action
Query	Echo, Echo Reply	8, 0	128, 129	Adjust checksum
	Information request, reply	15, 16	Obsolete	Silently discard
	Timestamp, Timestamp reply	13, 14	Obsolete	Silently discard
	Address Mark Request, reply	17, 18	Obsolete	Silently discard
	ICMP Router advertising, solicitation	9, 10	Single hop message	Silently discard
	IGMP messages	Any	Single hop message	Silently discard
ICMPv4 error messages				
Destination unreachable Type 3	Net unreachable	Type 3, code 0	Code 0 – No route to host	
	Host unreachable	Type 3, code 1	Code 0 – No route to host	
	Protocol unreachable	Type 3, code 2	Type 4, code 1 – ICMPv6 Parameter problem	Pointer to Ipv6 next header field
	Port unreachable	Type 3, code 3	Code 4 – Port Unreachable	
	Fragmentation needed and DF set	Type 3, code 4	Type 2, code 0 – ICMPv6 Packet too big	Adjust MTU with headers difference
	Source route failed	Type 3, code 5	Code 0 – No route to destination	
		Type 3, code 6, 7, 8, 11 and 12	Code 0 – No route to destination	
	Communication with destination host administratively prohibited	Type 3, code9 and 10	Code 1 – Communication with destination host administratively prohibited	
Error	Redirect	Type 5	Single hop message	Silently discard
Error	Source Quench	Type 4	Obsolete	Silently discard
Error	Time Exceed	Type 11, code n	Type 3, code n	
Error	Parameter problem	Type 12	Type 4	

- **IPv6 to IPv4 translation**

Because the minimum MTU required for IPv6 is bigger than the one required for IPv4, end to end MTU discovery is not appropriate because the result may result in MTUs lower than acceptable for the IPv6 starting point. In the case that IPv4 path MTU is smaller than the minimum required for IPv6, the translator must then fragment these packets to fit in the IPv4 MTU.

IPv6 header translation:

Field	IPv6	IPv4
Version	6	4
IP header length	N/A	5 (no options)
ToS and precedence	Xxxxxxxx	Xxxxxxxx
Total length	X	X+size (IPv4 header)
Identification	N/A	0
Flags	N/A	MF=0, DF=1
Fragment offset	N/A	0
TTL	X	X-1
Protocol	Next header = X	X
Header checksum	-	Generated
Source address	0::FFFF:0:A.B.C.D	A.B.C.D
Destination Address	0::FFFF:0:E.F.G.H	E.F.G.H

ICMPv6 header translation:

Type	Message	ICMPv6 type field	ICMPv4 type field	Further action
Informational	Echo request, echo reply	128, 129	0,8	Adjust cheksum
	MLD multicast listener query, report, done	130, 131,132	Single hop message	Silently discard
	Neighbour discovery	133, 134,135, 136, 137	Single hop message	Silently discard
Error				
Destination Unreachable Type 1	No route to destination	Type 1 – Code 0	Code 1 – Host unreachable	
	Communication with destination host administratively prohibited	Type 1 – Code 1	Code 10 - Communication with destination host administratively prohibited	
	Beyond Scope of source address	Type 1 – Code 2	Code 1 – Host unreachable	
	Address unreachable	Type 1 – Code 3	Code 1 – Host unreachable	
	Port unreachable	Type 1 – Code 4	Code 3 – Port unreachable	
Error	Packet too big	Type 2	Code 4 – Destination unreachable	
Error	Time exceed	Type 3	Type 11	
Error	Parameter problem	Type 4 – Code 1	Type 3 – Code 2 Protocol Unreachable	
Error	Parameter problem	Type 4 – Code not 1	Type 12 – Code 0	

3.1.3.2 Transport Relay

In this method, the translation is done at transport level. This mechanism acts as an intermediate between a TCP/IPv6 (or UPD/IPv6) connection and a TCP/IPv4 (or UPD/IPv4). No modification is necessary for IPv6 and IPv4 hosts.

Transport relay does not present fragmentation or ICMP conversion problems, since each session is closed in IPv4 and IPv6 worlds, respectively, but it does have problems like the translation of network layer addresses embedded in application layer protocols.

3.1.3.3 Application level gateway (ALG)

Application Level Gateway (ALG) is an application specific mechanism that allows an IPv6 node to communicate with an IPv4 node and vice-versa. The ALG's is used when translation of header packets is not enough. For example, the FTP protocol carries IP address and TCP port information within its payload. An FTP-ALG is required to translate this address. Normally, the ALG is used in conjunction with NAT-PT to provide support for this type of application.

3.1.4 Tunneling Mechanisms

3.1.4.1 Configured Tunneling

Tunneling provides a way to utilize and existing IPv4 routing infrastructure to carry IPv6 datagrams. To do this, IPv6 datagrams are encapsulated within IPv4 datagrams, so that they are IPv6 datagrams traveling through IPv4 networks.

The term "configured tunneling" describes the type where the tunnel endpoint is explicitly configured. This usually occurs when the tunnel endpoint is a router. The packet goes from an IPv6/IPv4 node (either host or router) to a router, where the IPv6 datagram is extracted from the IPv4 datagram, as shown below.

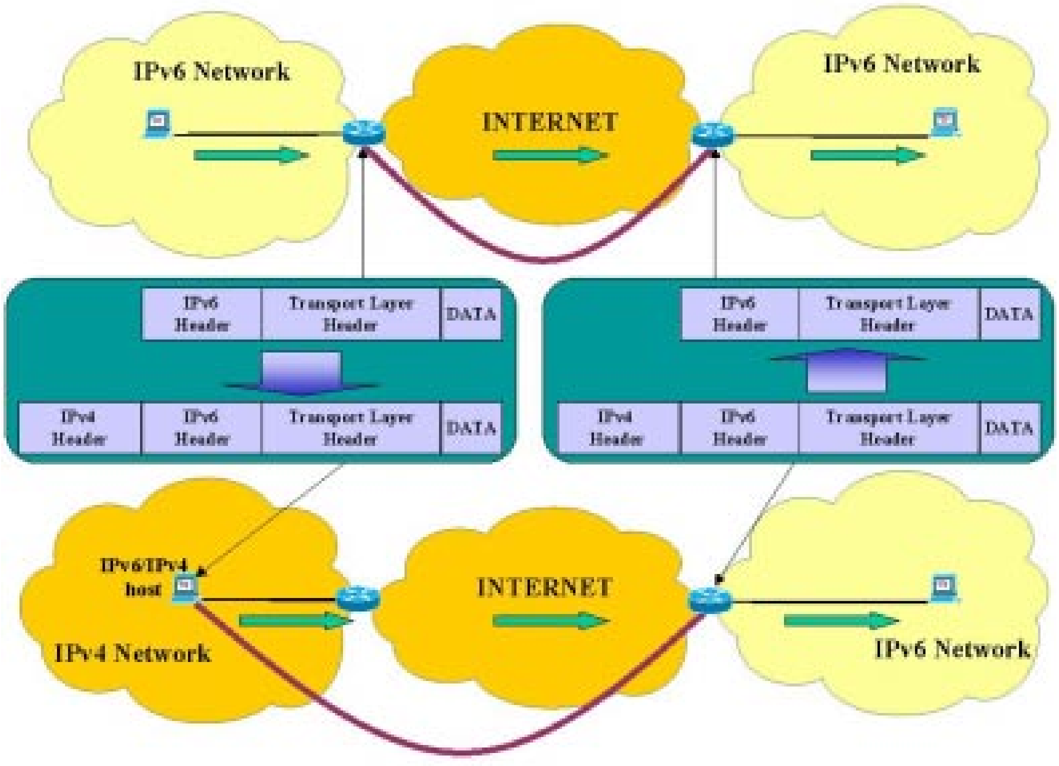

Figure 3 – Configured tunneling

The main feature of configured tunnels is the way that the tunnel endpoint is determined. They appear in environments where the two end hosts are separated by an IPv4 ocean and one or both of them, are within an IPv6 network. As far as the IPv4 datagram does not reach the destination host directly, the IPv6 address and the IPv4 address within that datagram does not refer to the same node (the IPv4 address refers to the router and the IPv6 address refers to the destination host). The previous fact makes impossible to know the IPv4 address for a given IPv6 address and this is the reason why the tunnel endpoint address must be determined from configuration information on the node performing the tunneling.

How the mechanism works:

The encapsulating node (it may be or not the source node) creates an IPv4 datagram encapsulating the IPv6 datagram (IPv6 datagram + IPv4 header) and it transmits the IPv4 datagram. For each tunnel the encapsulating node must store the tunnel endpoint address.

The decapsulating router (not the destination host) receives the IPv4 datagram, decapsulates it by removing the IPv4 header updates the IPv6 header and processes the received IPv6 packet. Although the router was not the destination it is used like it, for the encapsulating IPv4 header.

3.1.4.2 Automatic Tunneling

Tunneling provides a way to utilize and existing IPv4 routing infrastructure to carry IPv6 datagrams. To do this, IPv6 datagrams are encapsulated within IPv4 datagrams, so that they are IPv6 datagrams traveling through IPv4 networks.

The term “automatic tunneling” describes the type where the tunnel endpoints are automatically configured. This usually occurs when the tunnel final destination is a host. The packet goes from an IPv6/IPv4 node (either host or router) to a host, where the IPv6 datagram is extracted from the IPv4 datagram.

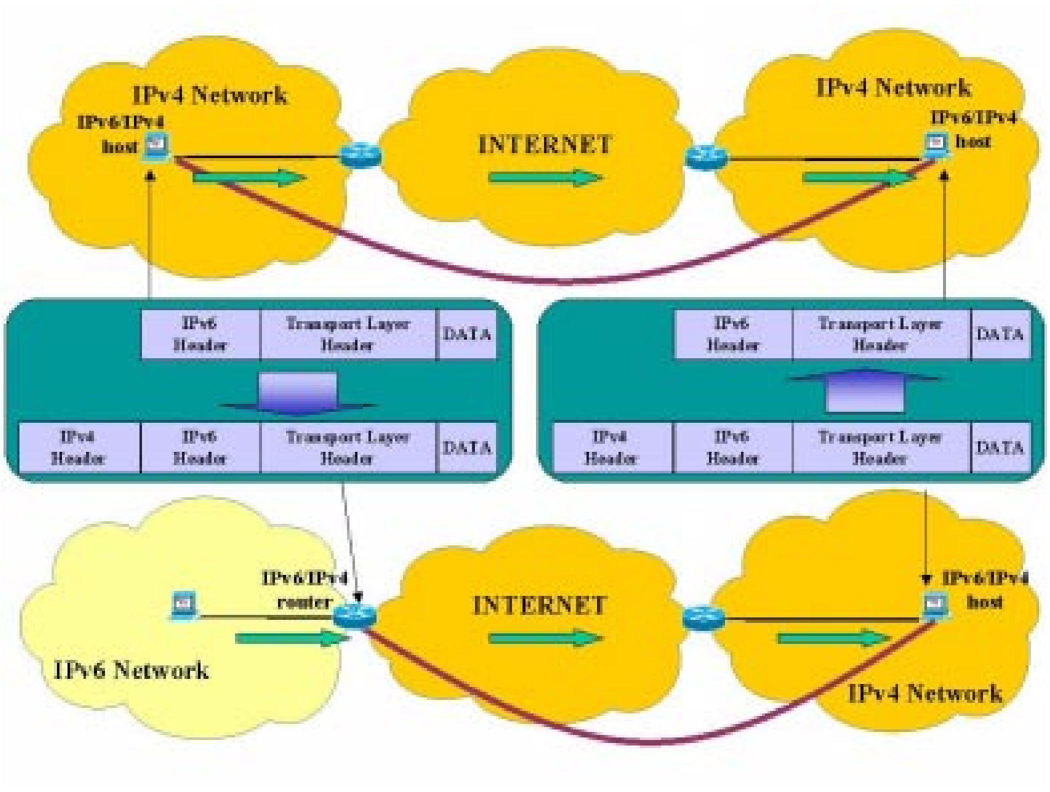

Figure 4 Automatic Tunneling

The main feature of automatic tunnels is the way that the tunnel endpoint address is determined. This kind of tunnels appears in environments where the two end points are separated by an IPv4 ocean and both are IPv6/IPv4 nodes. As far as the destination is the same node for both IPv4 and IPv6 addresses, that is both addresses refer to the same node. This fact can be exploited by encoding information in the IPv6 destination address (which is called IPv4-compatible IPv6 address) that will allow the encapsulation node to determine the tunnel endpoint IPv4 address automatically.

How the mechanism works:

The encapsulating node creates an IPv4 datagram encapsulating the IPv6 datagram (IPv6 datagram + IPv4 header) and it transmits the IPv4 datagram. To determine the tunnel endpoint address, the IPv4 compatible destination address of the IPv6 packet being tunneled is used. The IPv4-compatible IPv6 address is built by adding to the 96 most significant bits (set to zero) the IPv4 address, as shown below:

96-bits	32-bits
0:0:0:0:0:0:	IPv4 Address

Figure 5 - IPv4-compatible IPv6 address

IPv4-compatible addresses are assigned only to nodes that support automatic tunneling. These addresses are unique as far as the IPv4 addresses do not come from the private IPv4 address space.

The decapsulating host (the destination host) receives the IPv4 datagram, decapsulates it by removing the IPv4 header, updates the IPv6 header and finally, processes the received IPv6 packet.

3.1.4.3 6to4

The 6to4 method allows the connection of IPv6 hosts or sites which are isolated within an IPv4 network (with no native IPv6 support) to other IPv6 sites (or individual hosts) in the same circumstances.

Usually, the 6to4 mechanism is implemented in border routers, without specific host modifications. These routers must have globally routable IPv4 addresses and it is necessary to proceed with some router configuration.

The big advantage of this method is that none of the end-user nodes requires an IPv4-compatible IPv6 address, or configured tunnels like in other transition mechanisms, reducing the need for the end-user nodes manual configuration.

Another advantage of 6to4 is that the mechanism is not affected by the presence of firewalls in the IPv4 network as long as the 6to4 end-user nodes (routers or hosts) have globally routable IPv4 addresses.

There is a disadvantage of applying this method to individual hosts, because it limits the number of sites served by a given relay router.

This method is supposed to be used during the period of co-existence of IPv4 and IPv6 and is not intended as a permanent solution because it treats the wide area IPv4 network as a unicast point-to-point link layer. It will probably be used in the early phase of transition from IPv4 to IPv6 networks (large IPv4 ocean with small IPv6 islands).

- **6to4 prefix**

  The IANA has permanently assigned one 13-bit IPv6 Top Level Aggregator (TLA) identifier under the IPv6 Format Prefix 001 for the 6to4 scheme. Its numeric value is 0x0002, i.e., it is 2002::/16 when expressed as an IPv6 address prefix. This prefix has exactly the same format as normal /48 prefixes and it can be abbreviated as 2002:V4ADDR::/48.

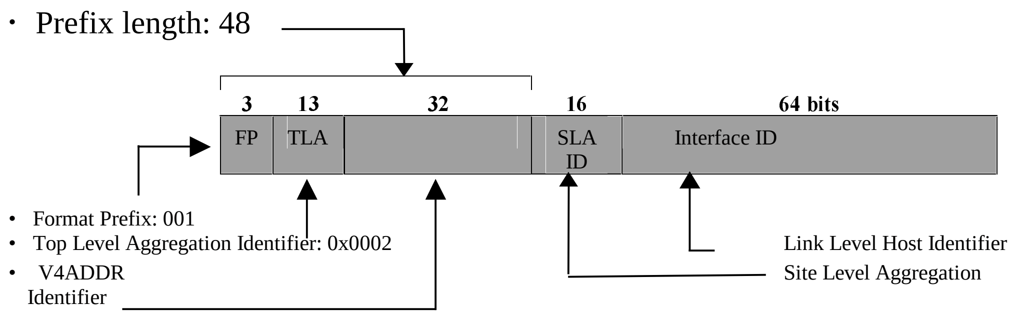

Figure 6 – 6to4 Address

- **Encapsulation in IPv4**

  IPv6 packets from a 6to4 site are encapsulated in IPv4 packets when they leave the site via its external IPv4 connection. Note that the IPv4 interface that is carrying the 6to4 traffic is notionally equivalent to an IPv6 interface, and is referred to below as a pseudo-interface. V4ADDR **must** be configured on the IPv4 interface.

  IPv6 packets are transmitted in IPv4 packets. The IPv4 header does not contain the final Destination and Source IPv4 addresses. One or both of these will be identical to the V4ADDR field of an IPv6 prefix formed as specified above. The IPv4 packet body contains the IPv6 header and payload.

<u>How the mechanism works:</u>

Each site identifies a router to run dual stack and 6to4 tunneling, ensuring that the router has a valid (routable) IPv4 address.

When one IPv6 host at a 6to4 site tries to access, by domain name, another IPv6 host at another 6to4 site, the DNS will return both IPv4 and IPv6 address for that host.

The requesting host selects the IPv6 address, which will have a 6to4 prefix and sends a packet to its site 6to4 router. This router realizes that it must send a packet to another site and that the next hop destination prefix contains the special 6to4 TLA value. Then it encapsulates the IPv6 packet in an IPv4 packet.

When the destination site's 6to4 router receives the IPv4 packet, it removes the IPv4 header and leaves the original IPv6 packet for local forwarding.

In this type of mechanism, IPv6 packets are tunneled (by this it is meant that the IPv4 packet body contains the IPv6 header and payload) in IPv4 between two routers.

The border router must decapsulate the IPv6 packets and forward them to their final destination.

It is necessary to understand that when tunneling to a router, the endpoint of the tunnel is different from the destination of the packet being tunneled.

- The entry node of the tunnel (the encapsulating node) creates an encapsulating IPv4 header and transmits the encapsulated packet;
- The exit node of the tunnel (the decapsulating node) receives the encapsulated packet, reassembles the packet if needed (i.e, if fragmentation happened in the IPv4 way), removes the IPv4 header, updates the IPv6 header, and processes the received IPv6 packet.

3.1.4.4 Tunnel Broker

Nowadays IPv6 global Internet uses a lot of tunnels over the IPv4 infrastructure, that is why a tool like Tunnel Broker makes sense, in order to reduce the configuration and maintainability of those tunnels.

This approach is based on the idea of provision a set of dedicate servers, called Tunnel Broker, to manage the configuration and maintainability of tunnels.

Tunnel Brokers can be seen as virtual IPv6 ISPs for those users who are already connected to the IPv4 Internet. It fits especially well for small isolated IPv6 hosts (also sites), that want to easily connect to an existing IPv6 networks.

<u>How the mechanism works:</u>

The client of the Tunnel Broker service is a dual-stack IPv6 node (host or router) connected to the IPv4 Internet. Before setting up the tunnel, there must be a change of information between the Tunnel Broker and the client, such as authentication, authorization and if necessary accounting information.

A quiet simple structure of tunnel broker can be composed by: the tunnel broker server, a DNS and several tunnel servers, as shown below.

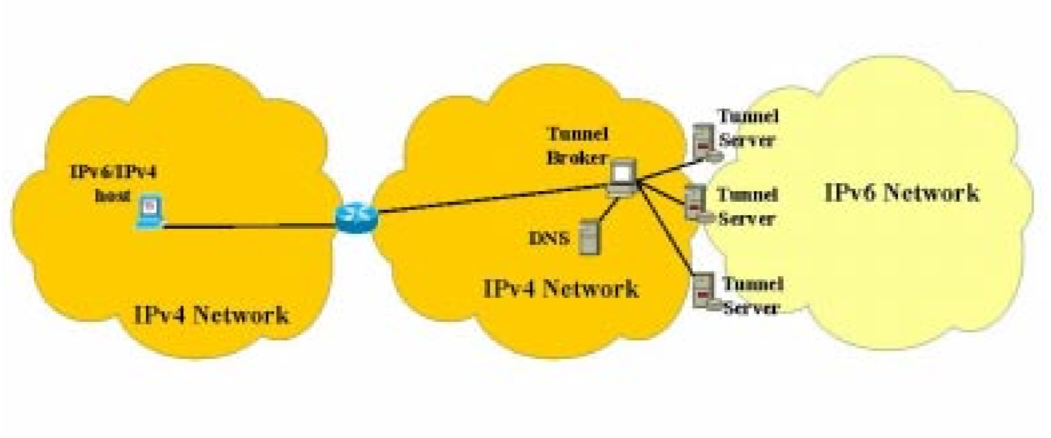

Figure 7 – Tunnel Broker

- The users connect to the tunnel broker to register and activate tunnels. The tunnel broker can share the load of network side tunnel end-points among several tunnel servers. It may also register the user IPv6 address and name in the DNS. The client has to provide at least the IPv4 address of his side of the tunnel, a name to be used for the registration in the IPv6 DNS assigned to his side of the tunnel and if it is a standalone host or a router.
- The tunnel broker designates a Tunnel Server to be used as the actual end-point at the network side. It chooses the prefix to be allocated to the client (between 0 and 128), fixing the lifetime of the tunnel.
- The Tunnel Broker registers the IPv6 addresses assigned to the tunnel end-points in the DNS.
- The Tunnel Broker configures the server side of the tunnel and notifies the relevant information to the client.

After these steps, the tunnel broker user is allowed to get access to any IPv6 network such as the 6Bone or another the Tunnel Server is connected to.

3.1.4.5 6over4

This mechanism [RFC2529] allows isolated IPv6 hosts located on a physical link where there is no IPv6-enabled router, to become fully IPv6 enabled by using a multicast-enabled IPv4 network as a virtual local link. That is why sometimes this mechanism is also known as "virtual Ethernet".

In order to support *Neighbor Discovery* and *Stateless Address Configuration*, some administratively scoped addresses [RFC2365] are used. These multicast groups simulate the virtual link.

The main advantage of this method is that IPv6 hosts do not require IPv4-compatible addresses or configured tunnels. The hosts themselves perform the tunneling. The basic architecture consists of a router with a native IPv6 connection and supporting the 6over4 mechanism, a multicast enabled network connecting the hosts and the router, and 6over4 hosts. In such an environment, 6over4 hosts can connect to native IPv6 hosts.

This mechanism appears to have essentially the same scaling properties as native IPv6 over most media except for the slight reduction in MTU size of the datagrams that will reduce bulk throughput.

During transition, routers will need to advertise at least two IPv6 prefixes, one for the native LAN (e.g. Ethernet) and one for the "6over4" domain. In addition, the prefix-length for link-local addresses must be 128 during transition when a router is handling both native LAN and "6over4" on the same physical interface. This is done to distinguish between these two cases instead of using the prefix FE80::/64 in both of them.

<u>How does the mechanism works:</u>

The basic idea in this mechanism is to simulate an Ethernet link layer using IPv4 multicast scoped addresses. So, the *Network Discovery mechanism* is used between 6over4 hosts and the IPv6-enabled in the same way it is used in a usual Ethernet link layer. This approach is similar to native IPv6 over a simulated LAN. The only difference is that in this case, 6over4 hosts are connected to the same IPv4 multicast scoped domain instead of being connected to the same shared-media network.

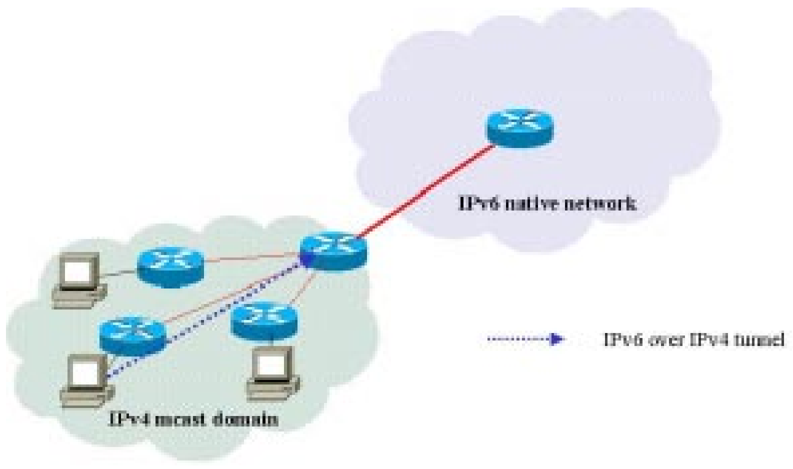

Figure 8 – 6over4

The mapping of IPv6 addresses to link layer addresses is done as usual using the ND protocol. In this case, the Source/Target Link-layer Address option uses IPv4 as link-layer. So, the whole IPv4 network is used as a shared media link-layer by means of following IPv4 multicast scoped addresses:

- All-nodes multicast address (239.X.0.1). The administratively scoped multicast address used to reach every node in the IPv4 domain supporting this mechanism.
- All-routers multicast address (239.X.0.2). The administratively scoped multicast address used to reach every router in the IPv4 domain supporting this mechanism.
- Solicited-node multicast address (239.X.C.D). An administratively scoped multicast address computed as a function of the solicited-target's address. (Being C and D the two low order bits of the IPv4 address)

In all this addresses, X represents the Local Scope Identifier (usually should be 192 [RFC2365]).

As can be noted, using IPv4 as link layer eliminates any physical-layer restriction from the migration plan, as host being migrated can be widely spread along the domain and even located at several hops from the IPv6 enabled router.

So the mechanism can be summarized as follows, the "6over4" hosts obtain their configuration (link-local address and prefix, IPv4 address of the IPv6-enabled router, and so on) using the ND protocol over IPv4 multicast. Then the IPv6 datagrams are sent encapsulated in IPv4 datagrams with protocol type 41. So, the hosts are responsible for doing that tunnelling.

3.1.4.6 DSTM

Dual Stack Transition Mechanism (DSTM) allows the communication between nodes with dual-stack (IPv6/IPv4), which belong to an IPv6 network, and IPv4-only remote nodes. It is not a solution for IPv6-only nodes.

DSTM assigns a global temporary IPv4 address to the IPv6 node and it uses IPv4-in-IPv6 tunnels so that IPv6 nodes could send IPv4 traffic through the IPv6 network.

It is bi-directional, i.e., the communication may be initiated either by the IPv6 node or the IPv4-only node.

The main advantages of this mechanism are:

- It is transparent to the network, as only IPv6 routing needs to be maintained inside the IPv6 network. This characteristic simplifies the network management.
- It is transparent to the application, as it allows IPv4-only applications to work in a dual IPv6/IPv4 node without modifications.
- It tackles the lack of IPv4 addresses through the use of a DHCPv6 server.

How the mechanism works:

DSTM is a mechanism implemented at nodes inside the IPv6 network and the border router communicating the IPv4 and IPv6 domains. It also makes use of DHCPv6. Therefore, DSTM requires a DHCPv6 server and a client at each IPv6 node.

The function of each of these components is as follows:

- DHCPv6 Server: It assigns a global temporary IPv4 address to the IPv6 node willing to communicate with the IPv4 remote node. It also maintains the mapping between IPv4 and IPv6 addresses. For DSTM support, DHCPv6 must support a new option to allow IPv6 nodes to obtain its temporary IPv4 address and to tell the client the IPv6 address of the end of the tunnel.
- DSTM daemon: It makes use of the DHCPv6 client in the node to request an IPv4 global address when initiating a communication.
- Dynamic Tunneling Interface (DTI): It is a virtual IPv4 interface inside the dual-stack node allowing the IPv4 application to send IPv4 traffic through an IPv6 network in a transparent way. The traffic delivered to the DTI is encapsulated into IPv6 packets and sent through the IPv6 interface to the border router.
- Border router: It is a dual-stack router connecting the IPv6 and IPv4 domains. This is the place where the IPv4-in-IPv6 tunnel ends in a communication between IPv6 and IPv4 remote nodes. This router also caches the mapping between IPv6 and temporary IPv4 addresses of dual-stack nodes.

<u>DSTM operation: Communication initiated by the IPv6 node</u>

Figure 9 shows the operation of DSTM when an IPv4 application on a dual-stack node (X) establishes a session with an IPv4 application on an IPv4-only node (Z) through a border router (Y).

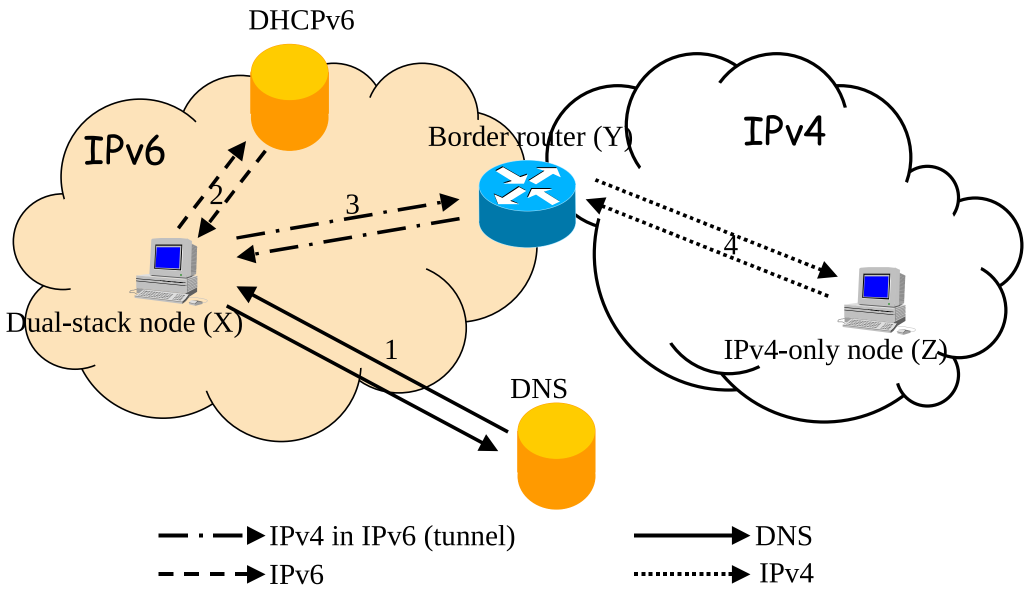

Figure 9 - DSTM operation

Step 1. When X wants to communicate with Z, it makes a DNS query for destination Z. The answer is the IPv4 address of Z. The actual path the DNS query follows is not relevant for DSTM operation.

Step 2. Once the destination address is known, the application on node X sends its first IPv4 packet to the DTI of node X. But node X itself needs an IPv4 address. That is the reason why it sends a query to the DHCPv6 server. This server also returns the Tunneling End Point (TEP) to the originating DHCPv6 client. The TEP is in charge of decapsulating the IPv4-in- IPv6 packets. In the example, the TEP is identified through the IPv6 address of the border router. It is important to remark that just IPv6 routing needs to be maintained in the routers of the IPv6 network.

Step 3. Node X encapsulates the IPv4 packet arriving to the DTI into IPv6 packets and sends them to the TEP. It is important to remark that the presence of an IPv6 network is transparent to the application.

Step 4. The TEP (border router) decapsulates the packets coming through the tunnel and sends them towards node Z through the IPv4 network. Traffic going back to X is sent from Z to Y using IPv4. Finally, Node Y, which maintains a cache of IPv4-IPv6 mappings, tunnels the packet back to X.

DSTM operation: Communication initiated by the IPv4-only node

In this case, a new component is introduced. It is called the AIIH Server (for Assignment of IPv4 global addresses to IPv6 Hosts), which combines the functionality of a DNS server and a DHCPv6 server.

The communication is established as follows. Node Z asks for the IPv4 address of X. This request arrives to the AIIH server. If node X does not have already a temporary IPv4 address assigned, the AIIH server allocates one and registers it in the DNS. AIIH returns the IPv4 address to node Z. Then, Z sends an IPv4 packet to the border router (Y). Node Y asks the AIIH server for the IPv6 address of X. Once Y knows this address, it tunnels the IPv4 packet in IPv6 and sends it to X.

Up to now, the only implementation of DSTM has been carried out using the FreeBSD OS.

3.1.5 Translation Mechanisms

3.1.5.1 NAT-PT

NAT-PT allows native IPv6 hosts and applications to communicate with native IPv4 hosts and applications, and vice-versa. The translation may be uni or bi-directional, and, is totally transparent to the end user because it requires no changes to end nodes.

A NAT-PT device resides at the boundary between an IPv6 and an IPv4 network.

In a traditional-NAT-PT (Basic-NAT-PT and NAPT-PT), sessions are uni-directional, outbound from the V6 network to gain access to the V4 network.

In Bi-directional-NAT-PT, there is connection in both, inbound and outbound, directions and can be originated in any of the networks.

The biggest strength of NAT-PT is being the right internetworking mechanism for an IPv6 only ISP as it allows interconnectivity to the IPv4 world for most applications.

The great disadvantage of NAT-PT is that problems appear when end-to-end security mechanisms are broken for IPv6 to IPv4 communication. End to end network layer security is not possible due to the intervention of NAT-PT at the header level. Also application layer security may be broken where an ALG is used to translate embedded addresses used at the application layer (this is a pertinent problem when we think about mobility). NAT-PT should only be used when there is no way of communicating by tunnel - and when security is crucial. Anyway, as a transition method this is a very powerful one.

How the mechanism works:

NAT-PT consists in a combination of Network Address Translation and, Protocol Translation, which means: address translation and also translation of an IPv4 packet into a semantically equivalent IPv6 packet and vice versa.

The address translation is needed in order to make hosts of one network able to identify hosts of the other network using the native address format of the first network.

During a session, NAT-PT handles the mapping of the IPv4 pool addresses to the IPv6 host.

NAT-PT adds the prefix as a packet passed from the IPv4 network to the IPv6 network and removes it from packets passing in the opposite direction.

NAPT-PT also translates the transport identifier (e.g., TCP and UDP port numbers, ICMP query identifiers).

NAT-PT may include a selection of Application Level Gateways (ALG). ALGs are included when in an IP application there are embedded addresses within the IP packet payload. Examples of this are FTP and DNS.

With Basic-NAT-PT, a block of V4 addresses are set aside (the pool) for translating addresses of V6 hosts as they originate sessions to the V4 hosts in external domain. For packets outbound from the V6 domain, the source IP address and related fields such as IP, TCP, UDP and ICMP header checksums are translated. For inbound packets, the destination IP address and the checksums as listed above are translated.

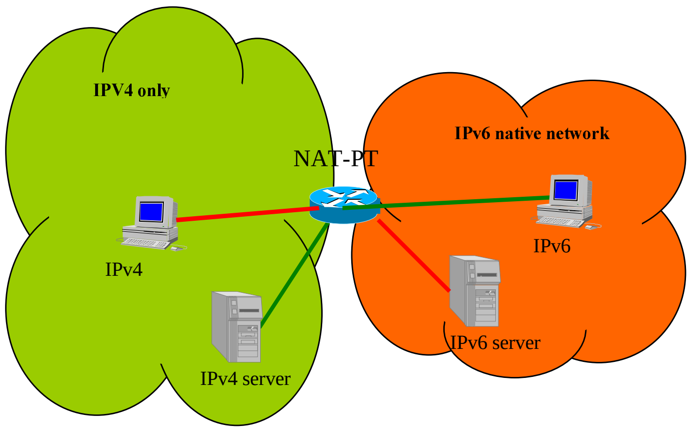

Figure 10 - NAT-PT

3.1.5.2 SOCKS64

SOCKS 64 provides a way to connect an IPvx network to other external IPvy one. Today, since the largest IP network is the current Internet (IPv4), the main case will be small IPv6 networks connecting to IPv4 world although the opposite scenario will probably apply in the future.

SOCKS [RFC792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC1928] was initially designed to enable some applications (telnet, ftp, http, etc.) to traverse firewalls in a transparent and secure manner. Later extensions (SOCKv5) have been defined allowing a SOCKS server to access both IPv4 and IPv6 networks. A dual stack

SOCKSv5 server running an appropriate translator application is known as a SOCKS 64 server and has the capability to translate IP packets from IPv4 to IPv6 and viceversa.

SOCKS 64 mechanism divides its functionality in two parts:

A SOCKSv5 Library must be installed on each client negotiating incompatible network boundaries. This library acts as a shim layer between the application and the transport layers. No changes to the application are needed since this shim layer translates the socket calls invoked by the application to modified socket calls towards the SOCKS 64 server. This is known as 'socksifying' a host.

A SOCKS 64 Server acts as a gateway for hosts enabling the possibility of accessing the neighboring network. The DNS resolution is delegated to this server so that it knows if protocol translation has to be applied.

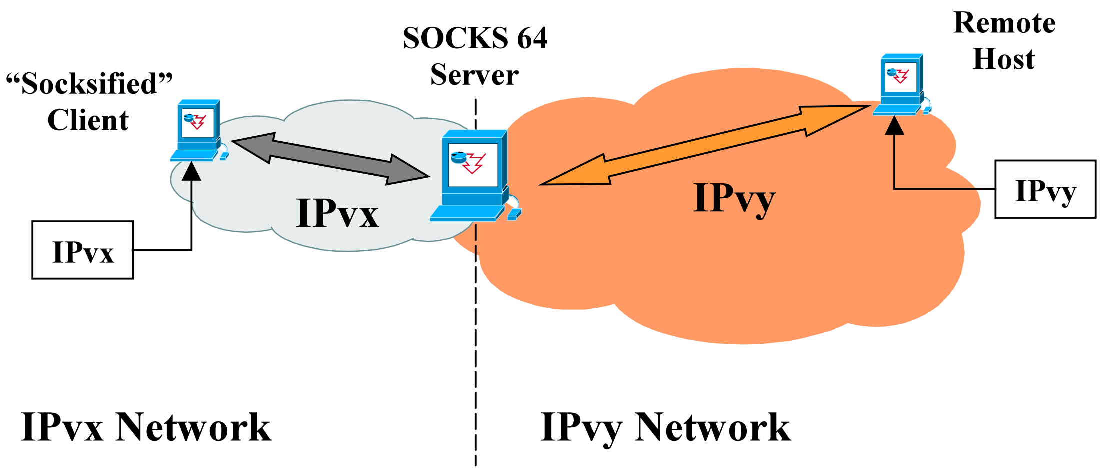

Figure 11 – Sock64

It is important to remark that SOCKS 64 mechanism enables hosts of one IP network to access nodes in another network but the connection must always be initiated by a 'socksified' host. A 'non-socksified' client has no possibility of initiating a connection across incompatible network boundaries.

<u>How the mechanism works:</u>

When a client contacts a host residing in an incompatible network, the following steps happen:

- A client application in a 'socksified' IPvx host starting a TCP or UDP connection to a IPvy server results in the library function "gethostbyname" being called. The application identifies the remote host using the Full Qualified Domain Name (FQDN).
- The SOCKS library in the client intercepts the "gethostbyname" library call and starts a TCP IPvx connection to the 1080 port of the SOCKS server.
- After the SOCKS server runs some authentication processes (if required), it resolves the FQDN provided by the local host. After the DNS resolution is done, the IPvy remote address is obtained.

- Then, the SOCKS server provides a fake IPvx address to the 'socksified' host identifying the remote IPvy host.
- The SOCKS server establishes an IPvy TCP/UDP connection to the IPvy remote node on behalf the 'socksified' host.
- All the packets sent from the 'socksified' to the remote host and viceversa are sent in these two connections:

  - IPvx: local_host-SOCKS_Server (TCP, usually port 1080).

  - IPvy: SOCKS_Server-remote_host (TCP/UDP, to the service well-known port).

The translation between both protocols is made in the SOCKS server.

3.1.5.3 BIS

When considering IPv4-IPv6 transition, the availability of IPv6 enabled applications is a major concern, basically because there is no point in considering a new and improved protocol stack if none of the existent applications are able to use the new features provided. Besides, in the early stages of transition, it is expected that several applications will not be IPv6 capable. So, transition mechanisms that enable the use of IPv4 applications in IPv6 environments are an interesting option to allow the introduction of IPv6 infrastructure in the IPv4 network.

These issues are the main target aimed by the Bump-In-The-Stack technique described in [RFC2767], which intents to provide IPv6 connectivity to IPv4-only applications. For example, an IPv4-only web browser that wants to retrieve information from an IPv6 http server, or two IPv4 applications residing on IPv6 stack machines that communicate through an IPv6 network.

The main characteristic of BIS is that it is physically placed in the machine where the IPv4 application executes; the application interacts with BIS as if it was the protocol stack, responding to DNS queries in which fake IPv4 address substitute real IPv6 address. We can consider BIS as a NAT-PT that resides in the same machine as the IPv4 application.

The rest of this document contains the following items: a section of BIS components, a description of BIS mechanism and finally a description of BIS implementations found.

All the components are located in the machine where the IPv4 application is located.

- **Translator**

The main function of this module is the network level translation of IPv4 packets into IPv6 packets and vice versa using the guidelines defined in [RFC2765].

- **Extensions Name Resolver (ENR)**

We can say that the ENR is a DNS application level gateway.

When an IPv4 application sends an A type query to the name server, the Extension Name Resolver (ENR) snoops the request an sends an extra AAAA type query for the same name. If the A query is resolved, it is sent to the application as it is. If only the AAAA query is resolved, the ENR requests the address mapper a fake IPv4 address associated to this IPv6 address, an returns the IPv4 address to the application in an A record.

- **Address Mapper.**

As it was mentioned earlier, when the application needs to communicate with an IPv6 only host, an internal IPv4 address is assigned to that IPv6 address, so that the application can establish the communication. The administration of IPv4-IPv6 associations is performed by the address mapper.

<u>How the mechanism works:</u>

The basic BIS mechanism is outlined in Figure 12 and briefly described in this section. When an IPv4 application needs to communicate, it launches a DNS A type query for the target host, host6. The ENR snoops the query and makes an additional AAAA query for the same host. If there is an answer for that A query, it means that the target host is IPv4 enabled so the application can directly communicate with it. If only the AAAA query is answered, it means that host6 is an IPv6 only host so the BIS mechanism is needed. In this case, the ENR requests the address mapper for an IPv4 address to be assigned to host6, so it can send an A record back to the application. Please note that the IPv4 address assigned is meaningful in the local environment only and it can not be used outside the local host. Once the application obtains an IPv4 address for host6, it starts sending packets to this address. When this packets are received by the tanslator, it requests the corresponding IPv6 address to the address mapper, in order to be able to translate the packet using [RFC2765] and send it to host6. When packets are received from host6, the inverse mechanism is used to make the translation.

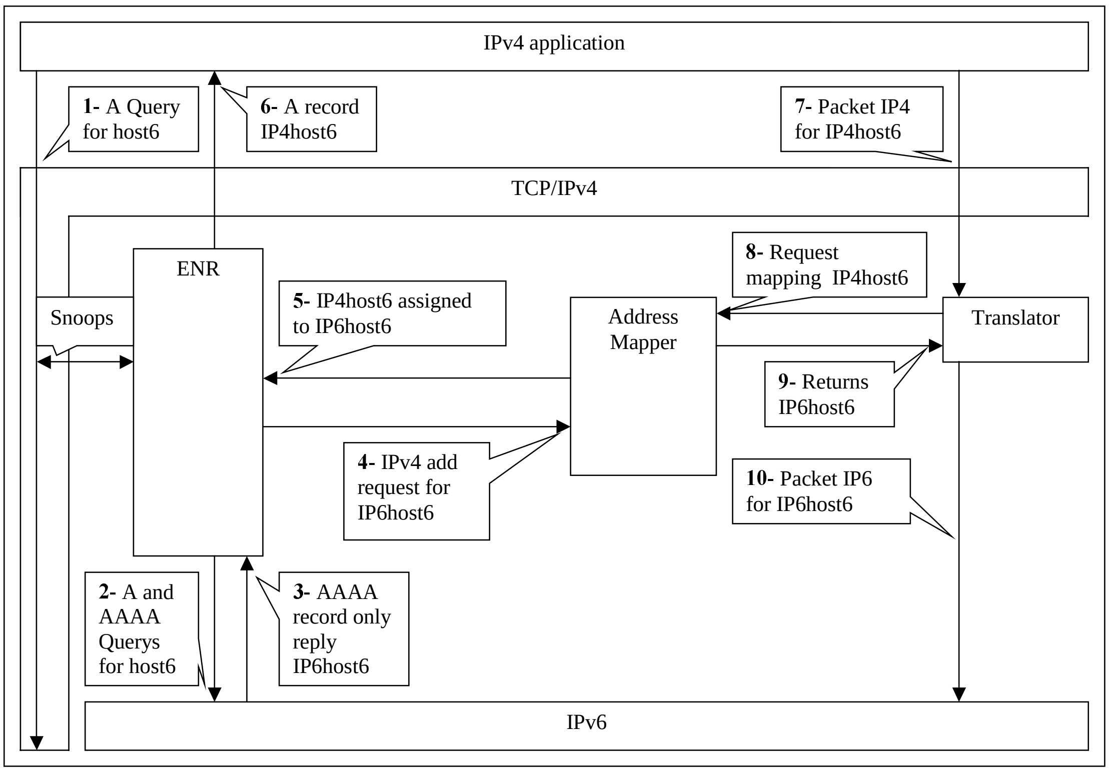
IPv4 application
1- A Query for host6
6- A record IP4host6
7- Packet IP4 for IP4host6
TCP/IPv4
ENR
8- Request mapping IP4host6
Snoops
5- IP4host6 assigned to IP6host6
Address Mapper
Translator
9- Returns IP6host6
4- IPv4 add request for IP6host6
10- Packet IP6 for IP6host6
2- A and AAAA Querys for host6
3- AAAA record only reply IP6host6
IPv6

Figure 12 - How the BIS mechanism works

Implementations.

A free BIS implementation for Windows 95/98 and NT, called Toolnet 6, can be found in [HITACHI].

3.1.5.4 TRT

Information about this mechanism can be retrieved from the Internet drafts: <draft-ietf-ngtrans-tcpudp-relay-01.txt> to <draft-ietf-ngtrans-tcpudp-relay-04.txt>, being the last one, the current one.

This mechanism is similar to NAT: just like NAT, it needs a stateful TRT system between the communicating peers, but unlike NAT the translation is made in transport layer instead of IP layer.

The aim of a TRT relay system is to provide a way to gain access to IPv4-only network resources outside an IPv6-only network. It is based on the transport relay technique, already used in firewall-related products.

How the mechanism works as IPv4-IPv4 transport relay translator:

It is here explained for TCP, but it works similarly for UDP when recognizing UDP inbound and outbound traffic pair in some way.

- The host A wants to initiate the communication to the host B: it tries to make a TCP connection to the destination host B, TCP packets are routed toward the TCP relay system based on routing decision.
- The TCP relay system pretends to be B and having its IP, establishing the communication between the initiating host and itself.
- Meanwhile the TCP relay system makes another TCP connection to host B and relays traffic from A to B and the other way around.

How the mechanism works as IPv6-IPv4 transport relay translator:

As before, explained only for TCP and only for IPv6 to IPv4, but it works the same the other way around.

- The host A (IPv6-only) wants to initiate the communication to the host B (IPv4-only). It tries to make a TCP connection to the destination host B, using as B address, an IPv6 address composed of a 64 bits prefix plus zeros plus the IPv4 address, where the prefix should be part of IPv6 unicast address space.
- The packet is routed to the TRT system and captures it. TCP relay system pretends to be B and having its IP, establishing the communication between the initiating host and itself.
- Meanwhile the TCP relay system makes another TCP connection to host B, which address gets by taking the lowermost 32 bits of the destination address. Afterwards relays traffic from A to B and the other way around.

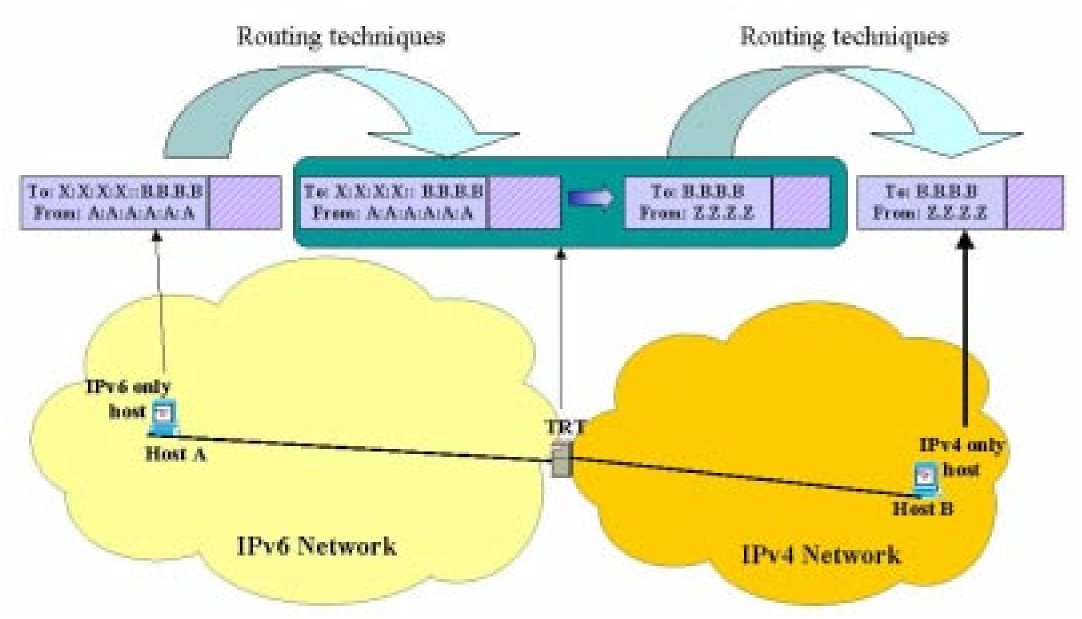

Figure 13 – IPv6-IPv4 transport relay translator

4. Transition Scenarios

The aim of this chapter is to define a set of theoretical scenarios to allow the understanding of where and how the mechanisms could be applicable. The present scenarios try to cover all possible real situations during a transition from IPv4 to IPv6. Also, they are applicable in all situations, without referring to ISP scenarios, corporate network, etc.

In further documents more practical and real situations will be studied although they will fit in one or more of these scenarios.

In this context, a scenario maybe interpreted as a snapshot of the network at some stage of its evolution from a full-IPv4 to a full-IPv6 network. This way, a transition strategy is considered as a succession of several transition scenarios.

Some concepts are necessary to define:

- Transition Mechanisms: specification and/or implementation to solve a specific problem. The transition mechanism should be implemented on the nodes (routers or hosts) with IPv6 support.
- Transition Strategies: a set of steps that is necessary to migrate an IPv4 network to IPv6 network.
- Transition Scenarios: a real situation that is necessary to consider in order to keep the interoperability between the IPv4 and IPv6 worlds.

So, in the following paragraphs the applicability of main Transition Mechanisms is evaluated. Maybe some of them listed as "not applicable" are really applicable doing some "hacking" but the goal of this document is to show the most suitable.

There are some mechanism that make no sense in many scenarios, for instance, BIS is designed to run IPv4 native applications in IPv6-only hosts so that it is no worth to be mentioned in any of the following sections.

In all scenarios we consider that IPv4 networks include IPv4-only nodes but IPv6 networks may contain both IPv6-only and dual stack nodes. This is because nowadays all IPv6 nodes are really dual stack nodes while current ones are IPv4-only, so these two situations are the most expected (IPv4-only, IPv6=Dual Stack).

4.1 Connection of IPv4 and IPv6 networks - Scenario A

This scenario represents the situation of hosts connected in an IPv4 network that need to communicate with hosts connected in an IPv6 network and vice-versa. The network A is IPv4 native, that is, only IPv4 routing is used. In the same way, only IPv6 routing is used in network B.

For the communication between network A's hosts and network B's hosts, a transition mechanism must be used.

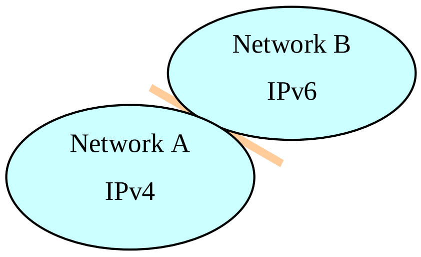

Figure 14 – Scenario A: IPv4 site - IPv6 site

Configured Tunnels and Tunnel Broker

Not applicable.

Configured tunnels are not applicable because both endpoints should be dual hosts and we assume that all hosts in Network A are IPv4 only. In scenario C.2 there are examples of another configuration with a dual host in an IPv4 network accessing an IPv6 site.

Since Tunnel Broker is a tool for managing and setting up configured tunnels, it makes no sense in this scenario.

Automatic Tunnels

Not applicable.

Using the same arguments described before for the configured tunnels, automatic tunneling is not applicable in this scenario.

6to4

Not applicable.

The same reasons that above must be applied.

6over4

Not applicable.

This scenario must not be mistaken for one allowing dual hosts in the IPv4 network, this last situation matches better with scenario C.2.

DSTM

Applicable.

Hosts in the IPv6-only network wanting to communicate with the IPv4 network need to implement a DSTM client and be dual-stack. There must also be a border router supporting DSTM to be used as tunneling end-point (TEP). Besides, there must be an AIIH server.

NAT – PT

Applicable.

However, there are some limitations when the NAT-PT is used. All IP sessions must take place via the same NAT-PT device. One way to guarantee this is to have the NAT-PT based on a border router unique (stub network). In this context, NAT-PT provides connectivity between an IPv6 stub network and the IPv4 world. The prefix PREFIX::/96 is advertised in the IPv6 stub network by the NAT-PT box and packets addressed to this prefix are routed to the NAT-PT.

No host configuration is needed and all NAT-PT translation is transparent to the end users.

SOCKS 64

Applicable.

SOCKS 64 can be used in this scenario in one of the networks or even in both at the same time. For instance, a SOCKS 64 server installed in the edge of network A will be used for hosts in A to access application servers in B but, with this configuration, B hosts cannot access application servers in A since SOCKS 64 only allows connections initiated from the internal side. Another possible solution could be to install two SOCKS 64 servers, each one for each network or site. This is possible since the two networks are directly connected, that is, there is not an IPvx Interconnection network.

TRT

Applicable.

TRT has been designed to be used in scenarios like this and its aim is to provide connectivity to the IPv4 only site from the IPv6 hosts. When an IPv6 host wants to access an IPv4 host addressed with A.B.C.D, it sends the packets to X:X:X:X:0:0:AB:CD IPv6 address (being X:X:X:X a 64 bits prefix) and the TRT server extracts the IPv4 destination.

The main problem using TRT is that it can be used by B IPv6 hosts to access A application servers but IPv4 hosts cannot access IPv6 servers in B using TRT.

4.2 Interconnection of networks

There will be real situations in which, it will be necessary to interconnect end-user networks. For example, an organization has branch offices in different places. Nowadays, each branch office network is connected through an IPv4 infrastructure. If this organization wants to migrate to IPv6, probably, it has to be done in several stages. Below, the possible scenarios during this transition phase are presented.

The main issue here is that the interconnection network is not managed by any of the end sites, so that each one can only use those mechanisms that imply changes or new equipments in their own network.

4.2.1 Current scenario - Scenario B.0

In the current scenario, we have IPv4 networks connected through an interconnection network, which is also IPv4. Figure 15 presents this scenario, where the networks A and B can be ISP service centers or end-user networks, such as customer networks, organization networks or home networks.

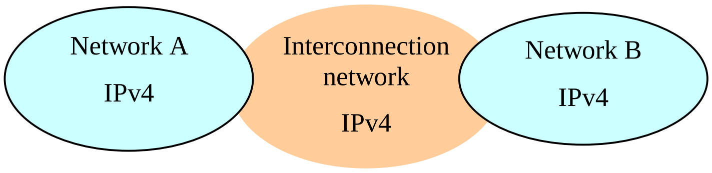

Figure 15 – Scenario B.0: Actual Scenario

4.2.2 Intermediate scenarios – Scenarios B1–B4

The following section presents various transition scenarios to interconnect end-user networks or connecting one end-user network to an ISP.

4.2.2.1 IPv6 End-User networks connected using IPv4 network infrastructure - Scenario B.1

This scenario is presented to show the connection of two IPv6 end-user sites through an IPv4 network infrastructure. Another situation is connecting a residential IPv6 end-user to an ISP.

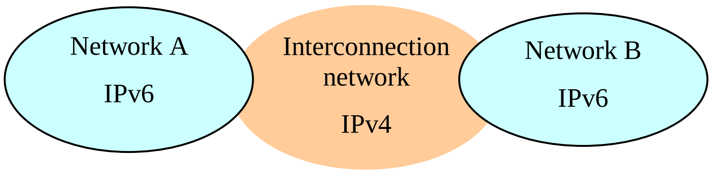

Figure 16 – Scenario B.1: IPv6 site -IPv4- IPv6 site

Configured Tunnels and Tunnel Broker

Applicable.

Configured tunneling is applicable in this scenario, which is the most suitable one for it. All A hosts can access B hosts and vice-versa by only setting up a tunnel from border router A to border router B and configuring routes in those routers (manually or maybe using a routing protocol).

Supposing B network is an IPv6 ISP which gives access to many A-like networks, the tunnels management tasks increase and it is advisable to install a Tunnel Broker in B, so the process of soliciting and setting up a tunnel becomes easier and faster.

Automatic Tunneling

Applicable.

This mechanism can be used between both (A and B) border routers since they own IPv4 public addresses and each one can reach the other using IPv4.

Also, if hosts in A and B networks are dual stack and own public IPv4 addresses, automatic tunnels can be used to establish connections from one network to another. For this mode of operation IPv4 routing has to be configured in A and B networks so that A hosts can access B hosts and vice-versa. Using this kind of tunnels there is no need to assign global IPv6 addresses to the dual hosts since IPv4-compatible addresses are used instead of them.

6to4

Applicable.

This is the “natural” scenario for 6to4. Network A will have a 6to4 router on its border with the IPv4 network and so will Network B. Both border routers will have an IPv4 address that will be used to build the IPv6 6to4 addresses. This address can be fixed or can be obtained dynamically.

Both border routers will encapsulate/de-encapsulate IPv6 packets in IPv4 packets. IPv4 packets routing will be obviously performed by the IPv4 network.

6to4 is defined by its authors as a temporary mechanism that should be used in the first steps of transition to allow IPv6 islands to interoperate each one with all other, without setting up hundreds of tunnels.

6over4

Applicable, see some restrictions below.

This mechanism is specially designed to connect IPv6 isolated nodes among them and to IPv6 native networks. It would be applicable if IPv4 multicast was configured in the interconnection network. Obviously, if the interconnecting network is the Internet, this mechanism makes no sense because an Internet with multicast is not expected by now.

Finally, this method is not the most suitable for such kind of scenario because it does not offer any advantage over the tunneling mechanisms and it is much more difficult to configure.

DSTM

Not applicable.

DSTM is not applicable due to the definition of this mechanism, which is designed to allow a dual host in an IPv6 network to communicate with an IPv4 host in a remote IPv4 network.

NAT-PT

Applicable, but not recommended.

NAT-PT is applicable being implemented on both border routers.

IPv6 hosts in network A have to be able to "see" IPv6 hosts in network B as if they were IPv4 hosts, and vice-versa. The configuration for this scenario is very complex namely when a DNS is used. If only static mapping translation is used, it will work and, in this case, the IPv6 host must have an IPv4 address assigned in the NAT-PT configuration.

SOCKS 64

Not applicable.

This is because two SOCKS 64 servers should be used and, if an A host would initiate a connection, it would be from the internal side of A SOCKS server but external for the B SOCKS server and then, impossible to establish.

TRT

Not applicable.

TRT mechanism cannot be applied in B.1 scenario because of its own definition and mode of operation which consists in allowing IPv6 hosts to access the IPv4 world encoding IPv4 addresses in IPv6 ones.

4.2.2.2 IPv4 and IPv6 End-User networks connected using IPv4 network infrastructure - Scenario B.2

The main point that makes this scenario different from A (IPv4-IPv6) is that both networks have no direct interface with each other.

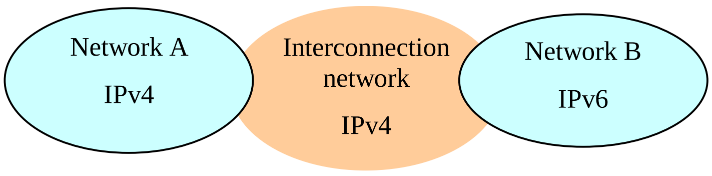

Figure 17 – Scenario B.2: IPv4 site-IPv4-IPv6 site

Configured Tunnels and Tunnel Broker

Not applicable.

This mechanism is not applicable here because the two end-points should be dual nodes. If dual nodes in network A are assumed, tunnels could be used from these dual nodes to the border router B (but this matches scenario C.2, not this one).

Automatic Tunnels

Not applicable.

Automatic tunneling is not applicable since there are no dual nodes in Network A. Again, if we assume there are dual nodes in A, the situation changes (see scenario C.2 again).

6to4

Not applicable.

The same reasons that above can be applied.

6over4

Not applicable.

Even if we would allow dual hosts in A (scenario C.2), the problem of configuring a common IPv4 multicast domain with a common administrative-scoped multicast address range could become a very difficult task.

DSTM

Applicable.

DSTM is applicable in this scenario by running the DSTM client in every dual-stack node in the IPv6 network that want to communicate with the remote IPv4-only nodes. There must also be a border router to be used as TEP. Besides, there must be an AIIH server.

Therefore, from DSTM operation point of view, it is the same as scenario A, since all changes and new boxes are introduced in the same network willing the access.

NAT-PT

Applicable.

NAT-PT must be implemented on the border router belonging to the IPv6 network. All requests and responses that belong to the same IP session, to and from the IPv6 network, have to be routed through the same NAT-PT router.

Of course, formally speaking this method is only practical for the IPv6 site willing to access the IPv4 one, or maybe an IPv6 ISP willing to offer IPv6 access to its IPv4 clients, because the NAT-PT box is configured and maintained by site B.

IPv4 site could also ask for this service to the IPv6 site, but NAT-PT must be installed in site B premises.

SOCKS 64

Applicable.

SOCKS 64 is applicable here by using it in the border router B so that, B hosts could access application servers in A but IPv4 hosts could not access application servers in B since the mode of operation of SOCKS mechanism (connections can only be initiated by internal hosts).

In this case there is not the possibility of setting up two SOCKS 64 servers as in scenario A, because the IPv4 ocean separates them.

Also, the same "practical" issues that in NAT-PT can be applied here.

TRT

Applicable.

TRT can be used here to provide access from B hosts to A application servers. The other way is not possible since the definition and mode of operation of TRT.

4.2.2.3 IPv4 and IPv6 End-User networks connected using IPv6 network infrastructure - Scenario B.3

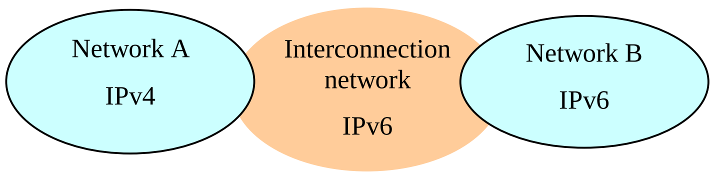

Figure 18 – Scenario B.3: IPv4 site -IPv6-IPv6 site

Configured Tunnels and Tunnel Broker

Not applicable.

Configured tunnels are not applicable here due to its own definition, then Tunnel Broker makes no sense either.

Automatic Tunnels

Not applicable (same reasons that for configured ones).

6to4

Not applicable (as the other tunneling techniques).

6over4

Not applicable.

However, allowing dual hosts in A, 6over4 would fit perfectly in such a scenario (similar to C.2) provided that A is a multicast-enabled network supporting multicast scoped addresses.

The key concept here is that if Network A administrator would decide to use 6over4; he would not depend on some other party to do that. He would only have to enable IP multicast in its network, use a router supporting 6over4 to connect to the interconnection network and then configure the proper administrative scopes into his domain.

DSTM

Applicable, but see the following restriction:

using DSTM in such a scenario implies to configure most of DSTM elements (DHCPv6 server, DNS server and DSTM clients in dual nodes) in B site, but the TEP should be installed in border router A, so that, an agreement between the two sites is needed, and maintenance becomes more difficult.

NAT-PT

Applicable.

NAT-PT must be implemented on a border router belonging to the IPv4 network. All requests and answers that belong to the same IP session, to and from the IPv6 network, have to be routed through the same NAT-PT router. To connect an IPv6 to an IPv4 host, the IPv4 host must be identified on the IPv6 network by an IPv6 address with format PREFIX:: X.Y.Z.W, where X.Y.Z.W is its IPv4 address on the IPv4 network.

This method is practical for the IPv4 site A willing to access the IPv6 world (and so the IPv6 remote site B). If the connectivity is demanded by site B an agreement is needed and NAT-PT would be maintained by site A administrator.

SOCKS 64

Applicable.

A SOCKS 64 server can be installed in the border router A so that IPv4 hosts in A can access IPv6 application servers in B.

TRT

Applicable but not expected in real cases.

The only situation that would make TRT suitable here is that A network is an IPv4 ISP and wants to provide IPv4 access to its IPv6 clients (B networks). In this particular case, A network can install a TRT server in its border router so that the IPv6 hosts in B networks can access application servers in A (or maybe other IPv4 networks beyond A).

This solution is not recommended due to its complex structure and some security problems found in TRT.

4.2.2.4 IPv4 End-User networks connected using IPv6 network infrastructure - Scenario B.4

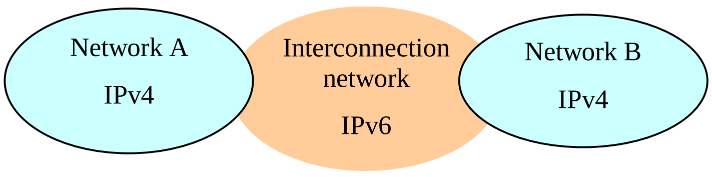

Figure 19 – Scenario B.4: IPv4 site-IPv6-IPv4 site

Configured Tunnels and Tunnel Broker

Not applicable.

Maybe here, it must be necessary to use IPv4-in-IPv6 configured tunnels, but there are no implementations of these nowadays. If this kind of configured tunnels is someday implemented, it would be a good idea to have a tool like "Tunnel Broker".

Automatic Tunnels

Not applicable.

6to4

Not applicable.

6over4

Not applicable.

6over4 is used when connecting IPv4 clouds to IPv6 domains.

DSTM

Not applicable.

NAT-PT

Applicable, but not recommended.

NAT-PT can be used in this scenario being implemented on the border routers belonging to the IPv4 network. However, it is necessary to study some details namely when the DNS is used. The IPv4 hosts on network A use directly the IPv4 addresses of IPv4 hosts of the network B to carry out the connections, and vice versa. The two NAT-PT translate these addresses into IPv6 addresses to route the packets through the IPv6 network.

SOCKS 64

Not applicable (same reasons that in B.1)

TRT

Not applicable.

TRT is defined to give access from IPv6-only hosts to application servers in IPv4-only networks.

4.2.3 Final scenario - Scenario B.5

In this scenario, we have IPv6 networks connected to the interconnection network also IPv6. No transition mechanism is needed.

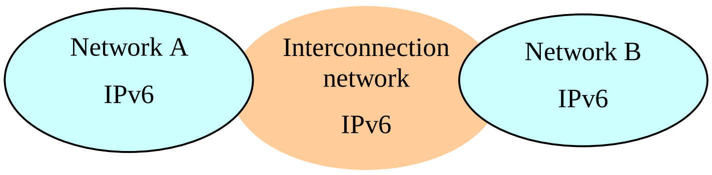

Figure 20 – Scenario B.5: All-IPv6

4.3 Connection from an isolated-host

The following section describes the situation of one user/host in an IPv4 network who intends to access IPv6 services or one user in an IPv6 network who intends to access IPv4 services. The point is that for users in IPv4 networks (C.1 and C.2) we distinguish between a user that does not want to install the IPv6 stack (C.1) and other who does it (C.2). In C.3, the user/host is supposed to be dual stack.

4.3.1 Isolated-only IPv6 host connection to IPv4 network - Scenario C.1

This scenario represents the situation of an IPv4-only user that is in an IPv4 network and intends to access an IPv6 network. A transition mechanism is needed to use between the user host and the IPv6 network. This scenario stands for a current user who wants to access to IPv6 sites but does not want to install the IPv6 stack in its host.

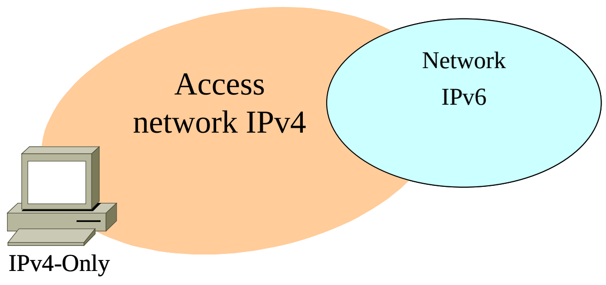

Figure 21 Scenario C.1: IPv4-only host (in IPv4 network)– IPv6 site

This scenario is the same that A scenario defined before, so the applicable mechanisms should be checked in that section.

4.3.2 Isolated -dual Stack host connection to IPv4 network - Scenario C.2

Here we assume that the user installs the IPv6 stack on its host, so it is a dual host in an IPv4 network trying to access an IPv6 site.

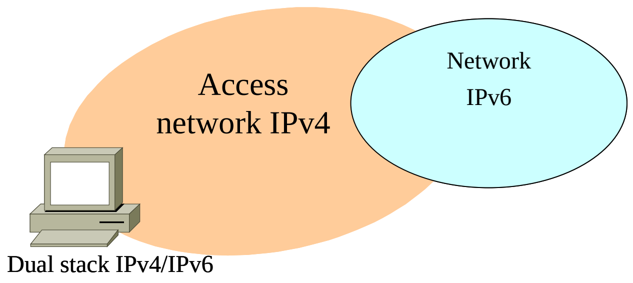

Figure 22 Scenario C.2: IPv4 host – IPv6 site

Configured Tunnels and Tunnel Broker

Applicable.

The main difference from previous scenarios is that the user can access the IPv6 site configuring a tunnel from its own host to the border router in the IPv6 site. If the IPv6 site is an IPv6 ISP which serves many IPv4 networks, it is advisable to install a Tunnel Broker in the ISP premises.

Automatic Tunneling

Applicable.

If the dual host placed in the IPv4 site owns a public IPv4 address, automatic tunneling can be used. The destination can be a host in the IPv6 network that should be also a dual host.

With this configuration, the IPv4 host needs no global IPv6 address to exchange packets with hosts in the IPv6 site.

6to4

Applicable.

In this scenario, the dual stack host would have to behave as a 6to4 router so that it would be possible to apply 6to4 to such a scenario. This use makes sense when the dual host is in a different IPv4 subnet that the border router.

6over4

Applicable.

This mechanism is used to allow isolated IPv6 hosts to exchange IPv6 traffic among them and with external IPv6 addresses, so this scenario is the best one to use this mechanism.

IPv4 multicast support must be configured in the IPv4 site, and the host MUST be dual stack.

DSTM

Not applicable.

DSTM was used in scenario C.1 or A allowing IPv6 hosts in the IPv6 site to access application servers in the IPv4 site, but in this scenario the opposite way is required, so this configuration is not valid.

NAT-PT

Applicable. (the same that in C.1 or A)

SOCKS 64

Applicable. SOCKS 64 can be installed in the dual node between the two networks in the same manner that in scenario C.1 or A.

TRT

Not applicable.

TRT was used in scenario C.1 or A allowing IPv6 hosts in the IPv6 site to access application servers in the IPv4 site, but in this scenario the opposite way is required, so this configuration is not valid.

4.3.3 Isolated- host connection to an IPv6 network - Scenario C.3

This scenario represents the situation of a user that is in an IPv6 network, and intends to access an IPv4 site. A transition mechanism is needed to use between the user host and the IPv4 network.

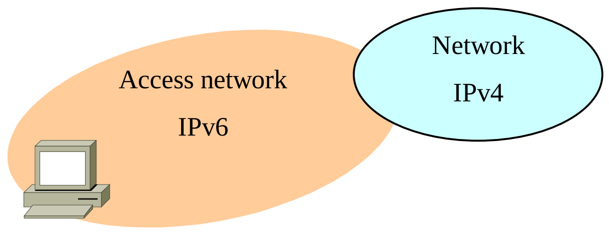

Figure 23 Scenario C.3: IPv6 host – IPv4 Site

Configured Tunnels and Tunnel Broker

Not applicable.

If IPv4-in-IPv6 configured tunnels are implemented some day, it will be possible to use them from the host placed in the IPv6 network to the border router. In that case that host should be a dual node.

Automatic Tunnels

Not applicable.

6to4

Not applicable.

6over4

Not applicable.

In this case, 6over4 should be installed in the remote IPv4 network and this issue is not the aim of this section. Maybe the IPv6 network can ask for this service to IPv4 site’s administrators.

DSTM

Applicable.

This scenario corresponds to the first case described in section 3.1.4.6 (subsection: "DSTM operation: Communication initiated by the IPv6 node"). Maybe this mechanism is the one that fits better in this access scenario and requires the host to be dual stack.

NAT-PT

Applicable.

SOCKS 64

Applicable.

SOCKS 64 can be used to provide access form the IPv6 host to the IPv4 remote site.

TRT

Applicable.

TRT can be used to give access to IPv4 application servers. There is no advantage for the host being a dual node.

5. Transition mechanisms evaluation

5.1 Theoretical evaluation and applicability of Transition Mechanisms

5.1.1 NAT-PT

Applicability

NAT-PT mechanism can be used when an IPv6 application running on an IPv6 host must communicate with an IPv4 application running on an IPv4 host. This mechanism enables the initiation of the communication in any of the two directions.

Management's costs

This mechanism requires a dual stack NAT-PT router working at the boundary between the IPv6 and IPv4 networks.

In Traditional-NAT-PT, communications must be initiated at IPv6 side. A pool of IPv4 addresses is managed by NAT-PT to lease them to IPv6 end-user hosts on request. NAT-PT router announces a prefix to IPv6 hosts that must be used to communicate to the IPv4 network. Therefore, there is no need to specific end-user host configuration.

In Bi-directional-NAT-PT and if there is no DNS service, this mechanism requires static mapping between IPv4 and IPv6 addresses. The Microsoft NAPT service includes two different configuration options to implement static mapping that were called by Microsoft: stateless and stateful.

In the stateful configuration mode, IPv4-IPv6 and IPv6-IPv4 mapping information must be configured in NAT-PT router through look-up tables. There is no need to specify end-user IPv6 host configuration but IPv6 host addresses need to be known to configure NAT-PT look-up tables.

In the stateless configuration mode, IPv6 addresses have to be configured in the hosts. The structure of IPv6 addresses assigned to hosts is based on the IPv4 address that is to be seen in the IPv4 side which make the address translation task automatic without the need for look-up tables. In the NAT-PT router, it is only necessary to configure the IPv4 address ranges assigned to IPv6 hosts.

It seems that the stateless configuration mode has less management costs since it does not require look-up table management. Although it requires address configuration in hosts, the stateful mode requires also that IPv6 host addresses must be checked by the manager.

Performance

As any other translation mechanism, NAT-PT introduces performance penalties on round-trip delays due to the address translation operation. The stateless configuration mode is better since it does not require look-up table queries as the stateful mode.

NAT-PT may include a selection of Application Level Gateways (ALGs), when in an IP application there are embedded addresses within the IP packet payload (e.g., FTP and DNS). ALGs will impose extra performance penalties.

Scalability

In Traditional-NAT-PT mechanism the scaling problems are of low relevance since the number of IPv4 addresses of the pool is usually significantly smaller than the total number of IPv6 end-user hosts (the total number of states in NAT-PT router is bounded by the dimension of the IPv4 address pool).

Bi-directional-NAT-PT mechanism in the stateful configuration mode has scaling problems. For larger networks, the number of look-up table entries strongly penalises its management cost and its delay performance.

NAT-PT mechanism in the stateless configuration mode has only minor scaling problems due to the IPv6 host configuration task.

Other

It is mandatory that all requests and responses pertaining to a session are routed via the same NAT-PT router. One way to guarantee this is to have NAT-PT based on a border router that is unique to a stub domain, where all IP packets are either originated from domain or destined to the domain (this is a generic problem with NAT). More complex network scenarios must be evaluated.

For Traditional-NAT-PT, it is not possible to have two redundant NAT-PT connecting an IPv4 with an IPv6 network in order to be router fault-tolerant

5.1.2 6over4

The 6over4 mechanism presents the following characteristics:

Applicability

6over4 relies on the existence of an underlying IPv4-enabled network. There must also exist a router supporting 6over4 and being directly connected to a native IPv6 network. Supposed that these requirements are met, 6over4 suits better when connecting isolated IPv6 hosts in IPv4 networks with IPv6-enabled networks.

Management's cost

Although the mechanism is not very complicated, it requires very skilled network administrators. Sometimes the problem of maintaining an IPv4 multicast-enabled network with administrative scooping is not very easy.

Performance

The performance is slightly worse than native IPv6 because of the overhead added by the implicit IPv6 over IPv4 tunnels used by the isolated IPv6 hosts. This performance reduction will be specially noted in bulk transfers because this need for extra encapsulation reduces slightly the MTU that can be used by applications.

Scalability

This mechanism appears to have the same scaling properties as native IPv6 over most media. However, in the case of ATM where IPv4 multicast relies in relatively complex mechanisms, it is expected that native IPv6 over ATM will perform better.

5.1.3 6to4

The evaluation of this transition mechanism is being done on a purely theoretichal perspective since it hasn't been deployed in our network yet.

Applicability

This transition mechanism is specially fitted for the connection of isolated IPv6 hosts or sites to other IPv6 hosts or sites in the same conditions.

Under these circumstances this transition mechanism is only applicable to the scenario B.1. There is another scenario, similar to B.1, to which 6to4 is also applicable: when there is a second IPv6 Network behind one of the IPv6 networks represented in B.1. This kind of scenario demands the existence of a relay router.

Another thing to have in consideration regarding 6to4 is that it is a temporary solution and sites using 6to4 should migrate as soon as possible to IPv6 prefixes and connectivity.

Management's cost

This mechanism is tipically configured on border routers, requiring very little configuration. Also the sites that use this method do not need any kind of configured tunnels or IPv6 addresses that are compatible with IPv4 addresses.

Since the IPv6 packets are encapsulated in IPv4 packets on border routers and are then transported through the global IPv4 network, it is not necessary to set up IPv6 exterior routing protocols.

Also, the presence of firewalls on border routers causes no apparent problems to this mechanism. The presence of NAT-PT requires a bit more of care in the configuration (it must guarantee that the IPv4 address embeded in the IPv6 address is not a private address).

Since 6to4 does not affect IPv4 routing it does not induce any kind of IPv4 routing loops. However a misconfigured 6to4 router could form a IPv6 routing loop.

6to4 mechanism assumes only IPv4 unicast capability. To support multicast it is necessary to have some multicast routing protocol. On the other hand, anycast addresses are perfectly compatible with the 6to4 IPv6 address scheme.

Performance

The greatest impact on performance when using this mechanism arrises from possible IPv4 fragmentation. Although this is not a terrible situation it is not desirable. The encapsulator should not set the IPv4 "do not fragment" bit as a solution to this problem.

Scalability

Althoug [RFC3056] only specifies the mechanism as it should be applied to a whole site, the mechanism is also applicable to single hosts and very small sites. This situation raises a problem since every one of these single hosts or very small sites needs an IPv4 address. In such a situation the solution actually contributes to the problem (the lack of IPv4 addresses).

Other than the above, the 6to4 mechanism should not have any scalability problems since there are no additions to the IPv4 routing tables.

Other

This mechanism may have some security considerations, namely the possibility of IPv6 “spoofing”.

5.1.4 BIS

Applicability

BIS mechanism can be considered as a NAT-PT plus a DNS ALG included in the IP stack of each host. It is used to provide IPv6 connectivity to IPv4-only applications located in the involved host. The main advantage of this mechanism is that it provides a fast way to integrate existing IPv4 applications to IPv6 networks. This method is particularly interesting for allowing interaction between different software entities, one IPv6-enabled, and another one IPv4-only that is not worth to be ported to IPv6. In some cases, further modules should be needed in order to properly translate the application protocol if it involves addresses.

Management cost

The BIS mechanism is a host scoped method, so it needs to be installed in every IPv4 system that needs IPv6 connectivity. Besides, every BIS system needs to be configured when it is installed. As we will see later, Ethernet adapter and IP stack need manual configuration; and also NAT tables must be managed manually. After the initial installation and configuration, further reconfiguration may be needed if network addresses change, i.e., topology change or renumbering. So, we can state that the high management cost is to be considered as one key element in the choice of the mechanism, specially if the solution is conceived as a long term option.

Performance

The per-packet translation, involving application-level translation, that is required on NAT-based mechanisms results in low performance operation. However given that BIS is a host-based solution, the load in a network, in what regards to the considered transition mechanism, is not centralized on a single device (in contrast to NAT, or TRT). Performance may become a key element when considering the use of BIS in a server environment.

Scalability

As a host based solution, scalability is granted by its own nature, besides the management cost considerations exposed in section “Management cost”.

5.1.5 DSTM

Applicability

Dual Stack Transition Mechanism (DSTM) allows the communication between nodes with dual-stack (IPv6/IPv4), which belong to an IPv6-only network, and IPv4-only remote nodes. It is not a solution for IPv6-only nodes.

Initially, it only provided unidirectional communication from dual nodes to IPv4-only nodes, but later extensions allow communication to be initiated by IPv4-only nodes.

With this mechanism, IPv4 applications may be used in mixed IPv4-IPv6 scenarios.

Management cost

One of the main advantages of the mechanism is that it simplifies the network management at the IPv6 network, as only IPv6 routing needs to be maintained.

On the other hand, the implantation of DSTM requires the maintenance of an AIIH server (AIIH stands for Assignment of IPv4 global addresses to IPv6 Hosts). It combines DHCPv6 and DNS functionality. DHCPv6 is not yet fully standardized. Furthermore, it uses DHCP options specifically defined for DSTM. And DNS dynamic updates should also be supported.

Performance

The establishment of the communication may be slow due to the many steps involved on it, as seen in previous sections. Furthermore, it may even not be established if the AIIH server runs out of IPv4 addresses.

Once the communication is established, there will be a reduction in the performance compared to native IPv6 due to IPv4-in-IPv6 encapsulation procedures. Besides, there is a reduction of the throughput due to the reduction of the MTU.

Scalability

The AIIH server maintains a pool of IPv4 global addresses which are dynamically assigned to the dual hosts. In that way, IPv4 addressing space is preserved. Furthermore, the border router maintains state for all the communications. This characteristic could limit scalability if many flows are passing through that router.

5.1.6 Configured Tunneling and Tunnel Broker

Applicability

Configured tunneling is mainly defined to be used by IPv6 sites/hosts willing to access other IPv6 sites or maybe an IPv6 ISP through an IPv6 interconnection network. This is the current case of one IPv6 site or single residential host connecting to the 6-Bone through the Internet.

When IPv6 ISPs appear they will deal with tunnels because many clients will reach their premises through IPv4 networks like the Internet or maybe IP networks established by local carrier companies.

For IPv6 ISP Tunnel Broker is a must, since it provides a web interface to the remote users willing to connect and also includes other facilities (timeout, security issues, etc).

Management's costs

The management costs of setting up a tunnel are not high since configured tunnels are ready to use in all IPv6 implementations without installing other software but the IPv6 stack itself.

The problem appears when one site provides access to many other with configured tunnels. In this case many addresses are used (2 IPv4 and 2 IPv6 for each tunnel) and the management tasks increase so that Tunnel Broker must be installed.

Scalability

A concrete router has a limit in the number of tunnels active that depends also on the amount of traffic routed by every tunnel. This limit defines the scalability of this mechanism and so that the number of clients per router for an IPv6 ISP using this transition mechanism.

This limit defined above should be measured and should be included in the performance definition of an IPv6 router.

Other

It is probably the transition mechanism which has been more tested since most sites accessing the 6-Bone use it.

5.1.7 Automatic Tunneling

Applicability

Automatic tunneling provides an easy way to use tunnels to interconnect IPv6 (dual) hosts distributed in an IPv4 infrastructure. These tunnels do not need to be configured one by one in every interface so they are very appreciated when interconnecting several machines in a private IPv4 network because in that case the number of configured tunnels would increase enough to be non advisable.

To activate these tunnels, this mechanism should be activated in every interface of these dual IPv6 hosts but then, these hosts can establish a tunnel with any other one automatically.

Because the mechanism has to be activated in every interface usually it is used to interconnect IPv6 hosts inside a private IPv4 network but the connections outside this network are usually set up with configured tunnels.

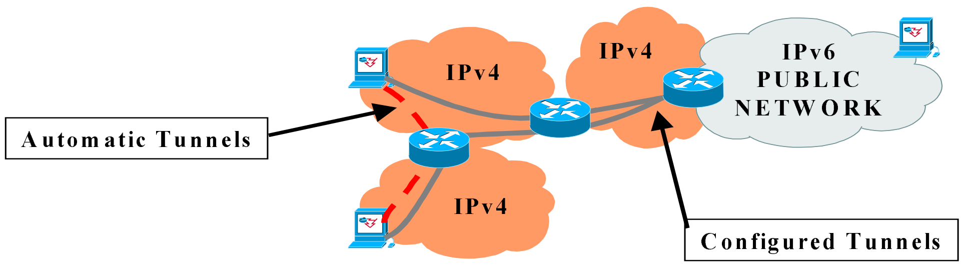

Figure 24 – Automatic Tunneling

Management's cost

The main reasons of using this kind of tunnels inside a private IPv4 network are the lower management costs of this mechanism compared to setting up all the needed configured tunnels and also because it saves resources (when two hosts are not exchanging IPv6 traffic, this tunnel does not exist so that it does not waste resources in those hosts).

Performance

The performance should be measured in terms of number of tunnels at the same time and traffic per tunnel.

Scalability

The scalability is one of the main advantages of this mechanism since only by activating it in one interface it can be used with any other host already configured in the same way.

5.1.8 SOCKS 64

Applicability

SOCKS 64 is a translation mechanisms that is intended to be used by small or medium sized organizations which use IPvx protocol and want to provide access to their hosts to application servers located in other IPvy sites.

SOCKS 64 is not applicable to give other sites the possibility of accessing application servers in our premises because connections are ever initiated by internal hosts.

Then, the main problem is that nobody could access WEB servers in our premises if SOCKS 64 is used.

Management's cost

To install SOCKS 64 mechanim a SOCKS 64 must be installed. This server runs in a dual node with at least one interface in each IPvx and IPvy networks. The server includes security mechanisms to give access only to the authorized nodes.

So this server should be installed and configured by the network administrator and its logs must be watched to detect security problems.

The clients nodes must run the client "socksified" applications, this can be done installing a new client by each service or better installing the "runsocks" library. This is another problem of this mechanism because all nodes willing to access the remote site must install this library and know how to use it.

Another problem is that maybe some services cannot be used, for instance if the remote server decides to open another TCP or UDP port and starts sending traffic it will not reach the client. All connections must be initiated by the client.

The last problem occurs in all translation mechanisms: if IPvx addresses are interchanged at application level an ALG is needed to manage it (DNS-ALG, FTP-ALG).

Performance

The performance of SOCKS 64 depends on the machine used to run the SOCKS server so that the performance should be measured in terms of number of users and traffic per user.

Scalability

SOCKS 64 is intended to small or even medium organizations but for large organizations NAT-PT should be considered.

5.1.9 TRT

Applicability

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

The TRT system can cover most of the daily applications: HTTP, SMTP, SSH, etc.

TRT supports bidirectional traffic only. Unidirectional traffic, such as multicast datagrams, is not supported.

Management's cost

This mechanism requires a stateful TRT system, usually a dual stack router, between an IPv6 and an IPv4 network. No extra modification on hosts is required.

Combined with a special DNS server (it is preferred to be run somewhere in the IPv6 network site), these systems support IPv6-to-IPv4 very well. It is harder to implement the other way, IPv4-to-IPv6, because non trivial mapping would be needed between DNS names to temporary IPv4 addresses, as presented in NAT-PT.

For address mapping, we reserve an IPv6 prefix (C6::/64) that should be a part of the IPv6 unicast address space assigned to the site. When the initiating host, with IPv6 address A:A:A:A:A:A, wishes to connect with the IPv4 host addressed B.B.B.B, the destination address to be used is C6::B.B.B.B/64. Routing information must be configured so those packets to C6::/64 are routed toward the TRT system.

As we do not relay IP layer packets between IPv4 and IPv6, we can decide IPv6 path MTU independently from IPv4 traffic.

Performance and Scalability

Scalability parameters would be number of connections accepted by the operating system kernel, number of connections a userland can forward, and the maximum number of transport relaying processes on a TRT system.

It is possible to have multiple TRT systems in a site by taking the following steps:

- Configure multiple TRT systems.
- Configure different dummy prefix to them.
- Let the initiating host pick a dummy prefix randomly for load-balancing.

5.2 Configuration and first experiences on Transition Mechanisms

5.2.1 NAT-PT

Introduction

Microsoft NAPT software is a tool that runs on Windows OS and enables a host with two NICs (Network Interface Cards) to implement the NAT-PT translation mechanism between an IPv4 network and an IPv6 network.

NAPT implements the Bi-directional NAT-PT mechanism, which means that the communication can be started in any of both sides of the NAPT host.

NAPT implements static address mapping translation. This means that hosts are able to communicate through NAPT only if translation addresses are previously assigned (an IPv6 host must have an assigned IPv4 address and an IPv4 host must have an assigned IPv6 address).

Description of the Functioning Mode

NAPT address translation can be set in two different configuration modes that are referred to by Microsoft as the stateless mode and the stateful mode. In the stateless configuration mode, there is no need to configure mapping information in the NAPT host. In the stateful configuration mode, all IPv4-IPv6 address mapping pairs must be configured in the NAPT host. In the following, the two configuration modes are separately described.

- **Stateless Configuration Mode**

In this mode, address configuration is required in IPv6 hosts that have to be able to communicate with IPv4 hosts. Each IPv6 host must be configured with an IPv6 address in the form:

::FFFF:0:D.D.D.D

where the *D.D.D.D* portion of the address is the IPv4 address assigned to this host in the IPv4 network part. Microsoft designates this kind of address as an IPv4-translated IPv6 address. When an IPv6 host initiates a communication to an IPv4 host with address *A.B.C.D*, it must use a destination address in the form:

::FFFF:A.B.C.D

Microsoft designates this kind of address as an IPv4-mapped IPv6 address. In this way, the translation task of NAPT host is embedded in the address information and there is no need to maintain state information about the IPv4-IPv6 address mappings. When an IPv4 initiates a communication to an IPv6 host, it uses the *D.D.D.D* portion of its IPv4-translated IPv6 address as the IPv4 destination address.

The IPv4-translated format is proposed by Microsoft when the IPv4-mapped address format is standardized in [RFC2373].

The address translation task of NAPT can be viewed as prefix insertion/removing of addresses. In the direction from IPv6 to IPv4, NAPT removes the prefix *::FFFF:0:0:0/96* of IPv6 origin address and *:: FFFF:0:0/96* of IPv6 destination address. In the direction from IPv4 to IPv6, NAPT inserts prefix *::FFFF:0:0/96* to the IPv4 origin address and *::FFFF:0:0:0/96* to the IPv4 destination address.

In NAPT host, the only address information that must be configured is the set of pools of IPv4 addresses that were assigned to IPv4-translated addresses.

- **Stateful Configuration Mode**

In this mode, there is no need to address configuration in IPv6 hosts. If a particular IPv6 host with IPv6 address *F:G:H:I:J:K:L:M:N:O:P:Q:R:S:T:U* has to be able to communicate with IPv4 hosts, an IPv4 address *A.B.C.D* must be assigned and an address mapping entry must be explicitly configured in NAPT host in the form

F:G:H:I:J:K:L:M:N:O:P:Q:R:S:T:U ⇔ A.B.C.D

A similar procedure must be done for IPv4 hosts that have to be able to communicate with IPv6 hosts. For each IPv4 host, an address mapping entry must be explicitly configured in order to assign to the IPv4 host an IPv6 address to be used in the IPv6 network part.

Windows

Configuration

NAPT software runs on Windows 2000 OS and requires Microsoft Research IPv6 (MSRIPv6) driver version 1.4. It does not work with Microsoft IPv6 Technology Preview driver although this is a later version.

NAPT has no routing protocol. Therefore, routing information must be manually configured.

In the IPv6 side, the configuration of NAPT host requires the following steps.

- Microsoft IPv6 protocol stack creates, in an automatic way, several logical interfaces for each NIC that can be addressed internally by an *ifindex* number. Let us designate the native IPv6 interface *ifindex* number by the letter *x* on any IPv6 host (including NAPT host).
- NAPT host must be configured to direct all packets for IPv6 addesses through this IPv6 native interface. This is done through the command:

  ```
  ipv6 rtu PREFIX x
  ```

  The *PREFIX* field depends on if NAPT is to be set in stateful or stateless mode. In the stateful mode, it can be any prefix set by system administration. In the stateless mode, it must be the prefix of IPv4-translated addresses (*::FFFF:0:0:0/96*). This interface must be configured to forward incoming IPv6 packets through the command:

  ```
  ipv6 ifc x forwards
  ```

  and to send router advertisements in order to make the other IPv6 hosts aware that this interface is the gateway for other IP networks. This is done through the command:

  ```
  ipv6 ifc x advertises
  ```

- Then, through appropriate registry configuration, the set of pools of IPv4 addresses (in stateless mode) or the IPv4-IPv6 address mappings (in the stateful mode) must be configured.

In the IPv4 side, the configuration of NAPT host requires the configuration of its IPv4 address and gateway.

In IPv4 network, two cases arise:

- All the pools of IPv4 addresses configured in NAPT host belong to its IPv4 network: in this case, nothing more needs to be done.
- Some of the pools of IPv4 addresses configured in NAPT host do not belong to its IPv4 network: in this case, since NAPT has no routing protocol embedded, it is necessary to configure routes in IPv4 routers to the NAPT host for IPv4 networks of the configured pools.

In the IPv6 network, two cases arise:

- NAPT was configured in the stateful mode: in this case, nothing needs to be done.

- NAPT was configured in the stateless mode: an IPv4-translated IPv6 address must be configured in each IPv6 host through the command:

  ```
  ipv6 adu x/::FFFF:0:D.D.D.D
  ```

  where the *D.D.D.D* portion should belong to one of the pools of addresses configured in NAPT host.

Performed Tests

- **Stateless Configuration Mode**

In order to validate the operation of NAPT in the stateless mode, the network set-up shown in Figure 25 was implemented.

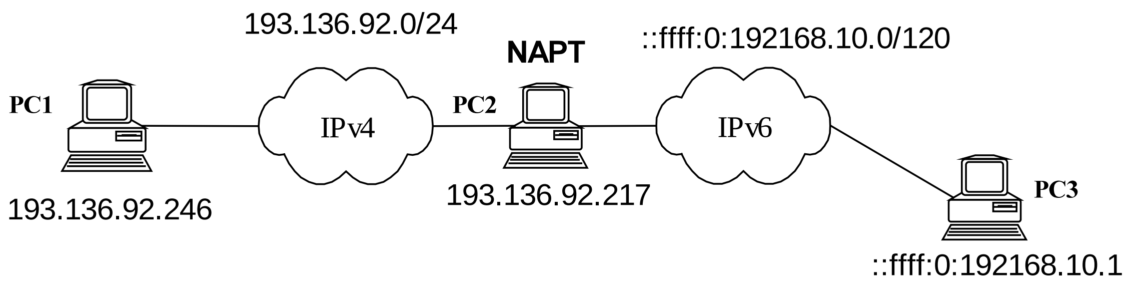

Figure 25 - NAT-PT: Stateless Configuration Mode

In PC1, a route for *192.168.10.0/24* IPv4 network was configured through the command:

```
route add 192.168.10.0 mask 255.255.255.0 193.136.92.217
```

In PC2, NAPT was configured with the following pool of addresses

```
192.168.10.1   192.168.10.10
```

The *ifindex* number of native IPv6 interface created by the IPv6 protocol stack was 6 for the NIC connected to IPv6 network. The sequence of routing information commands was:

```
ipv6  rtu  ::FFFF:0:0:0/96  6
ipv6 ifc 6 forwards
ipv6 ifc 6 advertises
```

In PC3, the *ifindex* number of native IPv6 interface created by the IPv6 protocol stack was 4. The IPv4-translated IPv6 address was configured through command:

```
ipv6  adu 4/::FFFF:0:192.168.10.1
```

The first experiment was to execute a *ping6* command from PC3 to IPv6 destination address *::FFFF:193.136.92.246* (which is the address assigned for PC1 in the IPv6 network). The second experiment was to execute a *ping* command from PC1 to IPv4 destination address *192.168.10.1* (which is the address assigned for PC3 in the IPv4 network). Both experiences were successfully run.

- **Stateful Configuration Mode**

In order to validate the operation of NAPT in the stateful mode, the network set-up shown in Figure 26 was implemented.

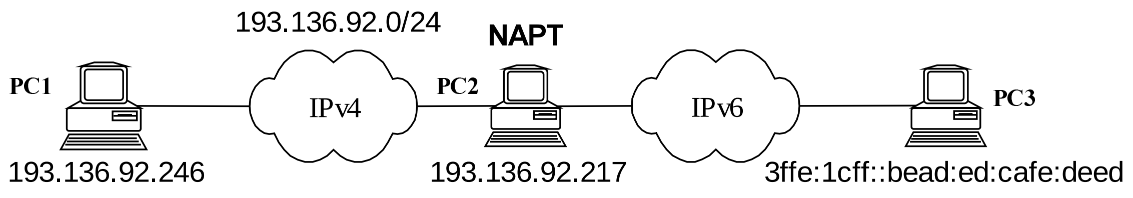

Figure 26 - NAT-PT: Stateful Configuration Mode

In PC1, a route for *192.168.10.0/24* IPv4 network was configured through the command:

```
route add 192.168.10.0 mask 255.255.255.0 193.136.92.217
```

In PC2, NAPT was configured with the following address mapping translation table (the first address is the real address and the second is the translated address):

```
193.136.92.246  BEEF:FEED::1234:5678
3FFE:1CFF::BEAD:ED:CAFÉ:DEED  192.168.10.1
```

The *ifindex* number of native IPv6 interface created by the IPv6 protocol stack was 6 for NIC connected to IPv6 network. The sequence of routing information commands was:

```
ipv6  rtu  3FFE:1CFF::/64  6
ipv6 ifc 6 forwards
ipv6 ifc 6 advertises
```

In PC3, the *ifindex* number of native IPv6 interface created by the IPv6 protocol stack was 4. The IPv6 address was configured through command:

```
ipv6  adu 4/ 3FFE:1CFF::BEAD:ED:CAFÉ:DEED
```

The first experiment was to execute a *ping6* command from PC3 to IPv6 destination address *BEEF:FEED::1234:5678* (which is the address assigned for PC1 in the IPv6 network). The second experiment was to execute a *ping* command from PC1 to IPv4 destination address *192.168.10.1* (which is the address assigned for PC3 in the IPv4 network). Both experiences were successfully run.

Note that in this experience, an IPv6 address was manually configured in PC3. It could have been created automatically if the first of the routing information commands in NAPT host was changed by the following command:

```
ipv6  rtu  3FFE:1CFF::/64  6 pub
```

With this change, PC3 automatically creates an IPv6 address with prefix *3FFE:1CFF::/64* and an Interface ID (the last 64 bits of the address) based on its Ethernet address. However, the NAPT full configuration requires that this address must be checked first in order to configure the address mapping translation table in NAPT host.

Conclusions

The experiences carried out proved that the NAPT software can be used for the implementation of Bi-directional NAT-PT mechanism with static address mapping translation. Additional experiences showed that both configuration modes can be used at the same time to configure NAT-PT mechanism. Stateless configuration mode requires the use of IPv6 addresses in IPv4-translated and IPv4-mapped format. Although not necessary, these formats can also be used in stateful configuration mode. Finally, NAPT software does not include any Application Level Gateway (ALG) module.

FreeBSD

*BSD uses the tool `natptconfig` that configures and controls the IPv6-to-IPv4 translator implemented in the BSD kernel.

The first step of the installation is to compile the kernel with support for `natpt` with the following command in the kernel configuration file:

```
option       NATPT
```

Then, we must define the status of the interfaces as *internal* (belonging to the IPv6 subnet) or *external* (belonging to IPv4 the subnet):

```
natptconfig interface rl0 internal #rl0 and rl1 are physical interfaces – but
they can be pseudo interfaces like gif, pseudo interface for manual tunnels


natptconfig interface rl1 external
```

In this implementation, if you want to change the configuration of interfaces, the system has to be rebooted.

Next, you must define the NAT prefix and define the outbound and inbound mappings you want. For example, the configuration file of `natptconfig` could be:

```
cat /etc/natptconfig.conf


prefix natpt 3FFE:3333::/96


map outbound from 3FFE:3328:6:FFFF::/60 to 163.117.139.166 port 28001 - 29000
map outbound from 3FFE:3328:1:1F::/60 to 163.117.139.166  port 29001 - 30000
map outbound from 2002:A375:8BA6: 1F::/60 to 163.117.139.166 port 30001 -
31000
map outbound from 3FFE:3328:6:0003::/64 to 163.117.139.166 port 31001 - 32000


map inbound from 163.117.139.166 port 53 to 3FFE:3328:6:
FFFF:2C0:26FF:FE70:1358 port 5353


map enable
```

To enable or change the mappings, you can execute as many times as you want the following command:

```
natptconfig –f /etc/natptconfig.conf
```

We must note that the translation is performed when the packet is received by an external/internal interface, before the packet is delivered to the application or to the routing daemon, so it is impossible for a packet generated in the system where the NATPT runs to be translated.

The outbound mapping process is performed when the packet is received by an internal interface and a mapping for the IPv6 source address of the packet exists. The packet is translated as follows:

1. IPv6 source address and port are translated into the IPv4 address and port of the tag 'to'.
2. IPv6 destination address is translated into the IPv4 address which is inside the IPv6 fake destination address.
3. Internally, a dynamic inbound mapping is created to translate returning packets.

The inbound mapping is carried out when the packet is received by an external interface and a mapping for the IPv4 destination address of the packet exists. The packet is translated as follows:

1. IPv4 destination address and port are translated into the IPv6 address and port of the tag 'to'.
2. IPv4 source address is translated to the fake address composed of NAT prefix plus the IPv4 inside.

For example, consider the NAT prefix is `3FFE:3333::/96` and it is defined the following outbound mapping:

```
map outbound 3FFE:3328:6:FFFF::/64 to 163.117.139.166 port 31000 – 32000
```

Imagine that the NAT box received the following IPv6 packets from an internal interface:

```
Source: 3FFE:3328:6:FFFF::1 port 1024
Destination: 3FFE:3333::163.117.139.44 port 23
```

This packet address and port are translated into the following:

```
Source: 163.117.139.166 port 31000
Destination: 163.117.139.44 port 23
```

And the following dynamic mapping is also defined inside NATPT:

```
map inboud 163.117.139.166 port 31000 to 3FFE:3328:6:FFFF::1  port 1024
```

Some time later, the following answer is received by the NATPT box:

```
Source: 163.117.139.44 port 23
Destination: 163.117.139.166 port 31000
```

When this answer is received by the NATPT box, and come from a external interface, the following translation applies to it, taking into account the dynamic mapping previously defined:

```
Source: 3FFE:3333::163.117.139.44 port 23
Destination: 3FFE:3328:6:FFFF::1 port 1024
```

And the machines are able to communicate transparently.

Note that the configuration of a DNS proxy is not described in this section, but is an integral part of the NAT mechanism, and should also be installed.

5.2.2 6to4

Configuration

- **Linux**

For configuring a 6to4 tunnel in Linux, the following steps must be applied. Note that 6to4 is supported from kernel version 2.4.0-test12 or later.

Configure the `gif0` with the following lines:

```
ifconfig sit0 up
ifconfig sit0 add 2002:<hex. IPv4 address>::<any  #>/16
```

The second command adds automatically the following new route (that can be checked with `netstat –nr -A inet6`):

```
2002::/16 ::  UA  256 0   0   sit0
```

If you would like the node to behave as a relay router, the prefix length for the IPv6 interface 6to4 address needs to be 16, so that the node would consider any 6to4 destination as ‘‘on-link”

If the host acts like a relay router, it applies the following steps and also it must be configured to announce the prefix `2002::/16` to his IPv6 native domain.

- **FreeBSD**

For configuring 6to4 tunnels with *BSD, the kernel have to be re-compiled to support for the `stf` (six-to-four) interface, adding the following line to the kernel `config` file:

```
pseudo-device   stf    1  # number of pseudo interfaces to make tunnels 6to4
```

Next, you must assign a 6to4 address to the interface with a command like:

```
ifconfig stf0 inet6 add 2002:<hex. IPv4 address>::<any #> prefixlen 16 alias
```

If you would like to restrict 6to4 peers to be inside certain IPv4 prefix, you may want to configure IPv6 prefix length as ”16 + IPv4 prefix length”. The `stf` interface will check the IPv4 source address on inbound packets, if the IPv6 prefix length is larger than 16 and it will only accept packets that match the IPv4 prefix of the `stf` configured interface. Also, for outbound packets, it will only forward the packets with destination addresses that match the `stf` configured prefix. For more information, examples and restrictions about this topic see “`man 4 stf`”.

Here is an example of 6to4 tunnel test with a FreeBSD-4.3-RELEASE box:

```
root@pulgon:~ $ ifconfig stf0
stf0: flags=4001<UP,LINK2> mtu 1280
        inet6 2002:A375:8BB1::1 prefixlen 16
root@pulgon:~ $ traceroute6 to 6to4.ipv6.microsoft.com
(2002:836B:9820::836B:9820) from 2002: A375:8BB1::1, 30 hops max, 12 byte
packets
 1  2002:836B:9820::836B:9820  327.903 ms *  306.566 ms
```

5.2.3 BIS

Introduction

In this section we will present the installation procedures for Toolnet6, a BIS implementation by Hitachi. Further information can be found at: http://www.hitachi.co.jp/Prod/comp/network/pexv6-e.htm.

System requirements

Item	Sytem Requirements
PC	PC/AT compatible computer with CPU: Pentium(R)or higher processor RAM: more than 32MB Hard-disk space: 1MB minimum
OS	Windows 95 (Hitachi PC only) Windows 98 WindowsNT4.0
Ethernet Interface	NE2000 Compatible Adapter Ether Link III

Elements

PEXv6.exe: It is the basic module of the tool. It implements translation features between IPv4 and IPv6.

V6NATMAN.exe: (NAT manager) is a configuration tool of the NAT table which is referred to by PEXv6. The NAT table is the address lookup table that maps IPv6 addresses to IPv4 addresses.

Installation instructions

This section is extracted from “inst-nte.txt: Installation note for WindowsNT” from Hitachi.

The steps that have to be followed are:

1 - Create a floppy disk for installation

2 - Install Ethernet board and driver

- Start WindowsNT, select "Settings" from Start menu and double-click "Network" icon. Ethernet Adapter will be recognized automatically. Follow the system's message to install the NE2000 Driver Program.

(NOTE: Here, the NE2000 Driver Program is installed and then it is replaced to the PEXv6 in the following step.)

- Click "Add..." button to install the PEXv6. Select Network Adapter dialog box, and choose "Novell NE2000 Compatible Adapter(IPv6)" from the installation floppy disk containing the driver required. Then "Setup Novell NE2000 Compatible Network Card" dialog box will appear automatically to configure all registry parameters.

3 - Configure registry parameters

- In the dialog box, nine registry parameters have to be configured.

(a) IRQ level which is required by hardware (decimal). Example: 5

(b) I/O port address (hexadecimal). Example: 0x300

(c) IPv6 address. Example: 3FFE:501:811:8001::10

(d) IPv6 Prefix Length (decimal). Example: 64

(e) IPv6address of default gateway. Example: 3FFE:501:811:8001::1

(f) IPv4 address. Example: 192.168.192.10

(g) IPv4 subnet mask. Example: 255.255.255.0

(h) Address Pool. Example: 192.168.192.20

(i) Number of Pool Addresses (decimal). Example: 10

(h) and (i) configure the address space that will be used to fake IPv6 addresses. (h) specifies the starting IPv4 address of the buffer pool. The subnet mask number for this IPv4 address must be equivalent to the subnet number for the local host IPv4 address. No addresses in the buffer pool can be used for the IPv4 address in the NAT table and for the IPv4 address of the local host.

The installation for PEXv6 will begin and after this, the NAT manager will be started automatically in order to configure NAT table.

4 - Configure NAT table

The configuration of the NAT table allows to register mappings between IPv4 addresses and IPv6 addresses. The IPv4 addresses must belong to the host's subnet.

Example:

The host address is 192.168.5.10 and subnet mask is 255.255.255.0.

192.168.5.100 is available.

To register "3FFE:501:811:8001::1" in NAT manager window,

```
IPv6 address     IPv4 address  Comment
------------------------------------------------
3FFE:501:811:8001::1   192.168.5.100 hostname
```

Quit the NAT manager when registration was completed.

5 - Configure TCP/IP properties

Configuration instructions:

- ***NAT table setup***

Extracted from "NAT Manager Help" from Hitachi.

1. Start the V6NATMAN.EXE

2. To set a new address: click [NewAddress] menu and click [NewAddress].

   Type the IPv6 Address, IPv4 address and comment. When you finish to configure parameters, click OK.

Note: The subnet mask number for the IPv4 address specified in this dialog must be equivalent to the subnet mask number for the local host IPv4 address.

3. To modify an address: select the IPV6 address you want to modify, click [NewAddress] menu and click [Property].Modify parameters, and click OK.
4. To delete an address: s Select the IPV6 address to be deleted, click [NewAddress] menu, and click [Delete]

Comments

As we can see through the installation and configuration procedures, this BIS implementation works at a very low level and it strongly depends on hardware implementation. To illustrate this, we can recall that a new network adapter driver is needed, or that the Windows 95 version only works on Hitachi´s PCs. Moreover, supported network card adapters, as we can see in the system requirements, are limited to NE2000 compatible cards or Ether Link cards. Besides, layer architecture is not followed, considering that the main module performs data link layer and network layer tasks as well as application tasks such as DNS translation. Furthermore, Toolnet6 was developed in 1998 for Windows95/98 and NT 4.0 and we have no knowledge of new versions supporting versions of Windows or further network adapters, such as PCI ones.

5.2.4 DSTM

<u>Introduction</u>

The known implementations of DSTM are one FreeBSD implementation to be run on its 3.4 version, and one NetBSD to be run on its 1.4.1 version. The source code may be found at ftp://ftp.inria.fr/network/ipv6/freebsd/ or ftp://ftp.ipv6.rennes.enst-bretagne.fr/pub/ipv6/. The file name is New.tar.gz. It is being developed at ENST Bretagne (France) and it is based on the INRIA FreeBSD TCP/IP-stack. It is still in its beta phase, thus, some of the components are still being developed.

Installation

For the mechanism to work, it is necessary to compile the new kernel with IPv6 and DTI support. It must be compiled at both, the DSTM router acting as TEP, and the client machines.

Steps to be followed during the installation process are explained in HOWTO-INSTALL_FreeBSD file (available at ftp://ftp.inria.fr/network/ipv6/freebsd/docs_tar.tar.gz):

How to install IPv6:

```
1 - Install FreeBSD 3.4 (optionaly install ALTQ 2.1 too)
2 - Install IPv6 files:
 * gunziped and untar New.tar.gz (use the z option of GNU tar)
 (I use "gzip -d < New.tar.gz | (cd /; tar xvBpf -)" command)
 * NOTE: the script inst-new is provided for the next step.
 You use the list of new files (src-new):
  + if an entry is a directory the Makefile in the parent is up to date
    then you have nothing to do.
```

```
        + if an entry is a regular file copy it in the ordinary version
          (I use "cp -p XXX-new XXX")
      3 - rebuild everything ("make world" in /usr/src for all the
          utilities (it takes time)), "config XXX" and "make depend; make"
          for the kernel.
      4 - read KERNEL and HOWTO-USE
```

Once the new kernel is built, addresses for both IPv6 and IPv4 interfaces must be statically configured, as there is no DHCPv6 implementation available.

Limitations

- No DHCPv6 implementation available. Configuration of addresses must be static.
- The number of DTIs in the border router (TEP) is limited.
- Runs with FreeBSD 3.4 and NetBSD 1.4.1, which are not the latest ones.

5.2.5 Configured Tunnels

Here we include a figure with two nodes having a configured tunnel with concrete IPv4 and IPv6 addresses and then, we show the configuration for different operating systems for the left host:

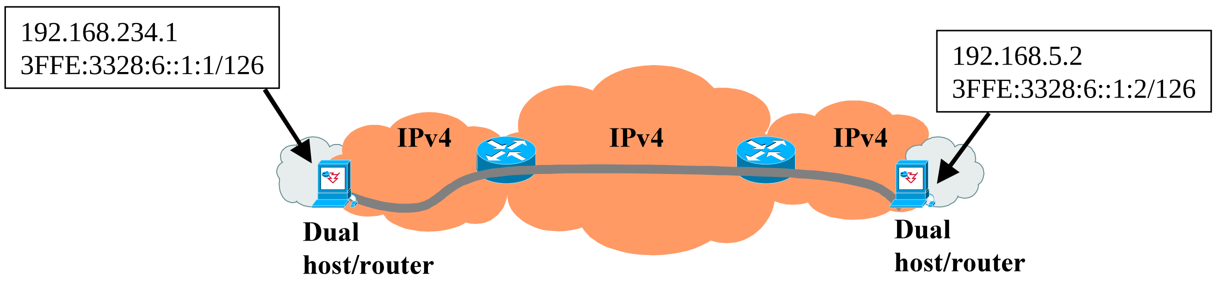

Figure 27– DSTM: Configured Tunnels

Linux

- Commands to add the tunnel:

```
/sbin/iptunnel add sit1 mode sit remote 192.168.5.2 local 192.168.234.1
/sbin/ifconfig sit1 up
/sbin/ifconfig -i sit1 add 3FFE:3328:6::1:1/126
```

- Command to add a default IPv6 route through the tunnel:

```
/sbin/route add -A inet6 0::0/0 gw 3FFE:3328:6::1:2
```

Solaris

- Command to add the tunnel interface:

```
Root@kheops#/usr/ipv6/sbin/ifconfig ip1 ipv4dst 192.168.5.2 ipv6addr
3FFE:3328:6::1:1 ipv6dst 3FFE:3328:6::1:2 ipv6mask 64 up
```

- Command to add a default route via the tunnel:

```
Root@kheops# /usr/ipv6/sbin/route add ipv6 0::0/0 3FFE:3328:6:1::2 1
```

Cisco

- Configuration example:

```
Router# show running
[…]
interface Tunnel1
 description EXAMPLE OF TUNNEL
 no ip address
 ipv6 enable
 ipv6 address 3FFE:3328:6::1:1/126
 tunnel source Ethernet0
 tunnel destination 192.168.5.2
 tunnel mode ipv6ip
!
[…]
interface Ethernet1
 ip address 192.168.234.1 255.255.255.0
 […]
```

Windows NT 4.0

- Command to add the tunnel interface:

```
ipv6 adu 2/3FFE:3328:6::1:1
```

- Command to add a default IPv6 route to go through the tunnel:

```
ipv6 rtu ::/0 2/::192.168.5.2 pub life 1800
```

5.2.6 Automatic Tunnels

Cisco

- To activate Automatic tunnels in Cisco the following commands are used:

```
Router# configure terminal
Router# ipv6 auto-tunnel
```

- To test it some commands of debugging are used:

```
Router# debug ipv6 packet
Router# debug ipv6 routing
Router# debug tunnel
```

- Finally, we try to reach host 192.168.5.2 (0xC0A80502) and see the debug output:

```
Router# ping ::C0A8:0502
11w4d: IPv6: nexthop 0::192.168.234.2,
11w4d: IPV6: source 3FFE:3300::29:2 (local)
11w4d: dest 0::192.168.5.2 (Tunnel1)
11w4d: priority 0, flow 0x0, len 100+0, prot 58, hops 255, auto-tunnel orig,
IPv4 next-hop.
```

- In the remote host the ICMPv4 encapsulated packets arrive:

```
12:11:17.116955 192.168.234.1 > 192.168.5.2: v6-in-v4
                 3FFE:3300::29:2 > ::192.168.5.2 icmpv6: echo request
```

5.2.7 Tunnel Broker

The implementation tested was ipv6tb, CSELT's IPv6 Tunnel Broker Implementation, with an IPv4 Linux box as Tunnel Broker, a FreeBSD box with KAME kit as Tunnel Server, and a dual stack Linux box as test client.

Required Software

Before installing ipv6tb, it is necessary to install and configure additional software in order to run the Tunnel Broker:

- A web server able to run perl CGIs. The one used was Apache 1.3.6.
- mSQL 2.0.3, a simple database used to store configuration and user information.
- Perl, version 5.004 or higher. This procedure has been tested with perl v5.005_03, and some scripts have to be modified in order to work properly, as stated below.
- The following perl modules:
    - CGI v2.46
    - Data::Dumper v2.10
    - Data::Showtable v3.3
    - DBI v1.02
    - DBD:mSQL v1.2
    - GD v1.18

These versions are suggested and so newer versions are likely to work although we have not tested them.

Installation and Configuration

1. Unpack the distribution tarball to a directory (e.g. /usr/local/ipv6tb)
2. Create a new directory under web server htdocs directory (e.g. htdocs/ipv6tb)
3. Copy the content of the ipv6tb/html directory of the distribution to the newly created one.
4. Create the mSQL database that will be used to store session information, by typing the following command (`msqladmin` is located in the bin directory of the mSQL installation)
    ```
    msqladmin create TunnelBroker
    ```
5. Check permission on both ipv6tb/distrib (readable and writeable world) and ipv6/install (writeable world) directories.

6. If the perl version in use is 5.005 or higher, is necessary to modify AdminTB.pl, tb.pl and tb_cron.pl (in usr/local/ipv6tb/distrib) scripts in order to make them work properly.

- Replace the lines between `START CONFIGURATION` and `END CONFIGURATION` with the following content:

```
my %mSQL (
    "host"  =>     <msql_host>
    "user"  =>     <msql_user>
    "pass"  =>     <msql_pass>
    "db"    =>     TunnelBroker
    );
```

  where `msql_host` is the host where the database is installed, and `msql_user` and `msql_pass`, the credentials needed to connect to the database (if not established, just put `undef`).

- Replace the following function declaration:

  `sub make_query(%;$)`

  `sub make_num(%;$)` and

  `sub make_hash(%;$)`

  with

  `sub make_query($)`

  `sub make_num($)` and

  `sub make_hash($)`

- Replace the line

  `my (%sql,$query_string) = @_;`

  with

  `my ($query_string) = @_;`

- Substitute each occurrence of the string `$sql` with the string `$mSQL`

- And finally, substitute each call to the functions `make_query`, `make_num` and `make_hash` of the form `function(%mSQL,query)` with `function(query)`

7. Copy the AdminTB.pl script in the cgi-bin directory and check its permissions (executable world)

8. Access with a browser the URL http://host.domain/cgi-bin/AdminTB.pl. This page offers entry points to configuration and administration subsections.

- Tunnel Broker Database and Software Location: It is necessary to enter the following information
  - mSQL Host: the name or ipv4 address of the machine hosting the mSQL database
  - mSQL User: the identifier of the user that can access the mSQL daemon. If you do not configure any access list (no **msql.acl** file), you can leave the value **undef**
  - mSQL Pass: the password associated with the above mentioned user. If you do not configure any access list (no **msql.acl** file), you can leave the value **undef**

- mSQL DB: the name of the database. The default value is **TunnelBroker**
- Preserve Database: if you want to maintain the configuration information stored in the database, you can select the value **Yes**. **During first installation**, use the value **No**.
- Software Distribution: The directory where the software has been unpacked
- Web cgi-bin directory: The path to the cgi-bin directory of your web server.

Once filled the form, you can submit it by pressing the *submit* button. By using the inserted parameters, the script performs some checks and, if all the entered information is correct :

- The script creates the needed tables in the indicated database.
- It configures the *AdminTB.pl* script in the cgi-bin directory.
- It configures the scripts located in the *ipv6tb/distrib* directory and creates configured scripts under the *ipv6/install* directory.

• Tunnel Broker Administrator(s): This area summarizes the information related to the IPv6 Tunnel Broker administrators:

- Administrator(s) Host(s): defines an ACL for the *AdminTB.pl* script by inserting the IPv4 address of the host that can use the script.
- Administrator(s) E-Mail(s): a comma separated list of e-mail addresses, used by the IPv6 Tunnel Broker to notify certain events (e.g. new tunnel router request).

• Script Location and Configuration: This area summarizes the information related to the IPv6 Tunnel Broker scripts:

- Use Script: by selecting No, you configure the IPv6 Tunnel Broker to run in 'debug' mode. The requests are processed by the *tb.pl* script but no interaction with the designated tunnel server and DNS will take place. Use this option during first installation if you want to check the IPv6 Tunnel Broker environment setup. In normal operation mode, set this field to the value Yes.
- Script Directory: the path to the script handling the interactions with the Tunnel Servers and the DNS, as explained before (eg. /usr/local/ipv6tb/script).
- Session Timeout: defines the duration of a user session. When the session expires, the user has to log again.

• Web Location and Configuration: This area summarizes the information related to the Web server hosting the IPv6 Tunnel Broker:

- Main Page Title: used on the html pages produced by the *tb.pl* script (e.g. **My Tunnel Broker**).
- Main Page URL: base URL of all the IPv6 Tunnel Broker html pages.

- Logo Image: the relative URL to your logo image (e.g. **images/my_logo.gif**). If you leave this field blank, no logo image will be displayed.
- Back URL: the URL for the back link on the IPv6 Tunnel Broker welcome page. If you leave this blank, no back link will be displayed on the welcome page.
- Back URL Text: text associated to the above mentioned URL.
- Contact Name: name of the contact person. If empty, no contact hyperlink will be displayed on the welcome page.
- Contact E-Mail: E-Mail address of the contact person.

• DNS: This subsection highlights the DNS information:
  - The first field in the form activates/deactivates the interaction with the DNS.
  - The second field defines the IPv6 Tunnel Broker domain space.

• Tunnel Administration: This section gives the access to tunnel administration pages.
  - On the left side, a list of the running tunnels is shown. By selecting a tunnel, you can examine the associated information.
  - If needed, you can release the tunnel by following the *delete* hyperlink.

• Tunnel Request Administration: This section gives access to tunnel request administration pages:
  - By selecting a tunnel request from the list, you access the associated information.
  - At this point, you can *accept* or *delete* the pending request. The requesting user will be informed via e-mail of your selection.

• User Administration:
  - By selecting a user from the list, you access the associated information.
  - By choosing the *delete* link, you can remove this user and the associated active tunnel (if any).

• Tunnel Server Administration: This area shows the status of the configured tunnel servers.
  - On the left side, you can see the list of the already configured tunnel servers and the *add* link.
  - The selection of a tunnel server in the tunnel server list, brings you to the page showing tunnel server parameters. This page is form-based as you can change (via the *modify* link) the information stored in the database.
  - The selection of the *add* link brings you to another form-based page very similiar to the previous one, with the only difference that fields are empty.
  - The form collects the relevant data for the new tunnel server via the fields:
    - o Identifier: the tunnel server identifier (e.g. jester).

- IPv4 Address: the address of the tunnel server.
- IPv6 Prefix: the IPv6 prefix managed by this tunnel server.
- Tunnel Server OS Type: the location and OS type of the tunnel server. In this popup menu, the prefix local means 'co-located with the Tunnel Broker'.
- Max Standalone Tunnels: max number of standalone tunnels managed by this tunnel server.
- Max Router Tunnels: max number of tunnels, terminated on a client IPv6 router.

9. Once Tunnel Broker is properly configured, it is necessary to move `tb.pl` script from `/usr/local/ipv6tb/install` to the `cgi-bin` directory of the web server and check its permissions (executable word)
10. It is also necessary to add `tb_cron.pl` script in `crontab` file of `root` in order to manage tunnel expiration.

    ```
    15 0-23 * * * /usr/local/ipv6tb/install/tb_cron.pl > /dev/null 2>&1
    ```
11. As mentioned above, some scripts can be configured to handle the interaction with tunnel and DNS servers. These scripts are usually located under /usr/local/ipv6tb/script.
    - Tunnel Server Configuration Scripts: These scripts handle the interaction with tunnel servers and are under the server subdirectory. They must be owned by root and the sticky bit must be set. If tunnel servers are remote, also proper configuration of `.rhosts` is needed, as they are based on `rcp` and `rsh`. Only scripts for FreeBSD INRIA stack and FreeBSD KAME stack are provided.
    - Tunnel Clients Configuration Scripts: These scripts are located under the client subdirectory and are made available to clients for download and off-line execution. Only scripts for FreeBSD INRIA stack, FreeBSD KAME stack and Windows 2000 Ipv6 stack are provided.
    - DNS Configuration Scripts: These scripts handle DNS updates and are located in the DNS subdirectory. They are not included with the distribution.

Client Interaction

Once the Tunnel Broker server is properly installed and configured, dual stack clients can access http://host.domain/ipv6tb to register and ask for IPv6 tunnels. The Tunnel Broker will choose proper address(es) and prefix(es), and generate the appropriate scripts for the clients to activate and deactivate tunnel interfaces. It will (if configured for so) create the tunnel end points in the tunnel server and add the needed entries in the DNS servers.

Comments

Although the procedure seems to be complex in accordance with the guide showed here, in fact it is just following the instructions. Configuration and set up of the tunnel broker do not take much time. The main problems we have encountered here deal with differences in the versions of the packages required. Unfortunately perl code changes significantly from versions

5.004 and 5.005 (and higher), so different workarounds need to be applied. Apart from that the user interface is much better than the one found with other mechanisms.

5.2.8 SOCKS 64

The implementation tested on Linux and Solaris boxes is NEC'S SOCKSv5 implementation. To show an example of configuration a concrete network is described in Figure 28.

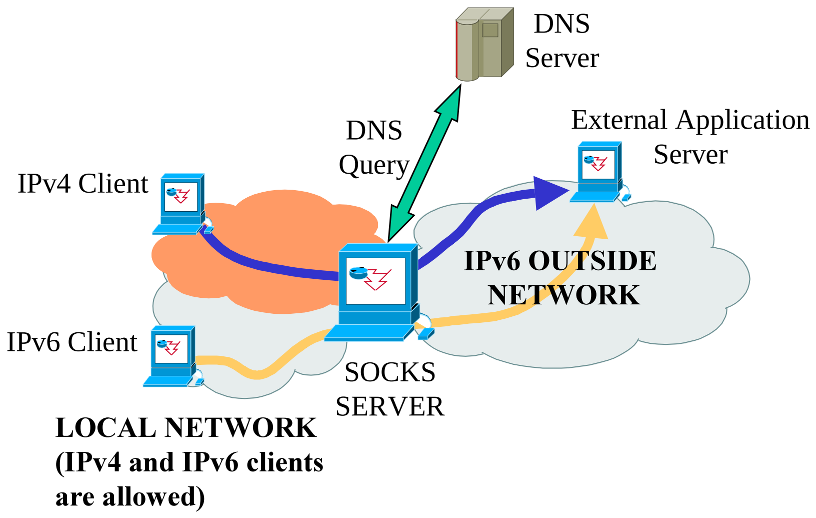

Figure 28– SOCKS64

Installation

In the NEC's implementation there are two packets to install, first "Socks5-v1.0r9" package must be installed in the node acting as SOCKS server. Then, the patch file "Socks-Trans" must be applied, since IPv6 support will be merged in the original SOCKS source tree in the next release, which will be a commercial release.

In the IPv4 client nodes only "Socks5-v1.0r9" package need to be installed but, obviously, in IPv6 clients nodes (if any) the patch file is also needed.

Socks Server Node

The Server Node must be a dual stack host and have full IP connectivity.

Once "Socks5-v1.0r9" package and "Socks-trans" patch file are installed in the server node, the software must be compiled. The easiest way to do it is following these steps:

1. Change to the directory containing the SOCKS5 source code and configure SOCKS5 for that system. To do this run ***./configure***. While configure runs, it prints informational messages listing the features it is checking.
2. To compile SOCKS5, type: ***make***
3. To install programs, data files, and documentation, type: ***make install.***

After obtaining the binaries and installing them, the SOCKS server (a common Unix daemon) should be configured before running. To do this a file "/etc/socks5.conf" is created with the following content:

```
# more /etc/socks5.conf
permit k - 111.111.111. - - -
permit n – [IPv4_adress] - - -
permit n – [IPv6_address] - - -
#filter ftpv6 - c - - - - -
```

With each permit line a host is allowed to use the SOCKS service. Instead of specifying the IPv4 or IPv6 address of a host, a pattern "net/prefix" can be used to give access to multiple IP address with only one permit sentence.

Also filters for some applications can be used (ftpv6 filter, for instance).

One good idea is to configure the system to automatically start the SOCKS daemon at boot time. In Solaris this is done by creating a script to start/stop the SOCKS daemon in the */etc/init.d* directory.

The SOCKS daemon can be started running ***/usr/local/bin/socks5 –d 3***, to configure other options in the server they can be configured at configuration/compile time and some of them at the command line.

Client Nodes

The "Socksified" clients are included in "Socks5-v1.0r9" package and "Sock-trans" patch file has to be applied if the client will communicate with de SOCKS server using IPv6.

This software provides some clients linked statically with SOCKS libraries (***rtelnet***, ***rftp***, etc.) but also provides "*runsock [command]*"which triggers the execution of this command dynamically linked with SOCKS libraries (For instance, you can socksify Netscape browser with: ***runsocks /usr/bin/netscape***)

Installing and Compiling the software is similar as in the SOCKS server node.

Configuration and Operation

Client Nodes

To configure the client there should be the following environment variables (they can be stored in a ".profile" file):

```
SOCKS_SERVER = (explained in next paragraph).
SOCKS5_FAKEALLHOSTS=1
SOCKS5_NONETMASKCHECK=1
SOCKS5_LOG_SYSLOG=1
SOCKS5_DEBUG=3
```

Server Nodes

To configure where the SOCKS server is, two cases can be found:

- **Client uses IPv4 to talk to the SOCKS server:** In this case, the environment variable SOCKS_SERVER in the client has the value "[IPv4_address]:1080" which defines where the SOCKS server is, and in which TCP port listens for connections.
- **Client uses IPv6 to talk to the SOCKS server:** Now, SOCKS_SERVER has the value "[IPv6_address]:1080" which belongs to the SOCKS server.

Note that the way to write the IPv6 address and the port in the environment variable value follows RFC 2732 (“Format for Literal IPv6 Addresses in URL's”).

There are other variables that can be configured, they can be studied in the “README” and “README.trans” files stored in main installation SOCKS directory.

5.2.9 TRT

Only one implementation of the TRT has been found and it is called FAITH. The FAITH IPv6-to-IPv4 TCP relay translator first appeared in WIDE hydrangea IPv6 stack. Nowadays the source code is freely available in the KAME project stack kit.

The KAME project aims to provide free reference implementations of IPv6/IPsec. The code is developed on top of 4 BSD variants, and 6 different versions:

- BSD/OS 3.1 and 4.2.
- FreeBSD 2.2.8 and 4.2.
- NetBSD 1.5.
- OpenBSD 2.8.

The following *BSD official releases integrate KAME kit:

- FreeBSD 4.0 and beyond.
- NetBSD 1.5 and beyond.
- OpenBSD 2.7 and beyond.
- BSD/OS 4.2 and beyond.

The difference between KAME SNAP kit and *BSD releases would be like below:

- KAME SNAP kit comes with more experimental protocols/APIs support and userland programs. This would attract researchers more, as not everything in KAME tree will be merged into *BSD tree.
- KAME kit will always be based on public release version of *BSD, so you can enjoy latest IPv6 tree on top of stable *BSD platforms. This would be nice feature to have when your goal is to do IPv6 experiment, not to chase *BSD-current.
- The merged *BSD releases will give tightly integrated IPv6-ready operating system. KAME kit is supplied as diffs to *BSD release versions, so it has less integrated feeling.

Summarising, the KAME kit includes more recent IPv6/IPsec changes by KAME team. It also includes more experimental code than the *BSD distributions, as well as additional functionalities which have not been integrated into *BSD yet. Assuming we want to install the TRT to test it and learn how it works, we choose to install the whole KAME kit, and not only a *BSD release which includes FAITH.

The latest KAME snap can be downloaded from: ftp.kame.net/pub/kame/snap.

Nowadays, it is updated every Monday.

Installation procedure I - preparation of the tree

Once you have already downloaded the KAME kit and uncompressed it, you are ready to tackle the installation. The first step consists in preparing the tree. Let's call the entire file-tree under this directory "the KAME tree".

This directory contains several information files and two directories. The "kame" directory contains files created by the KAME project, which are shareable among NetBSD, FreeBSD, and other operating systems. The other directory is for your target operating system. For example, if you are on FreeBSD, the KAME tree looks like this:

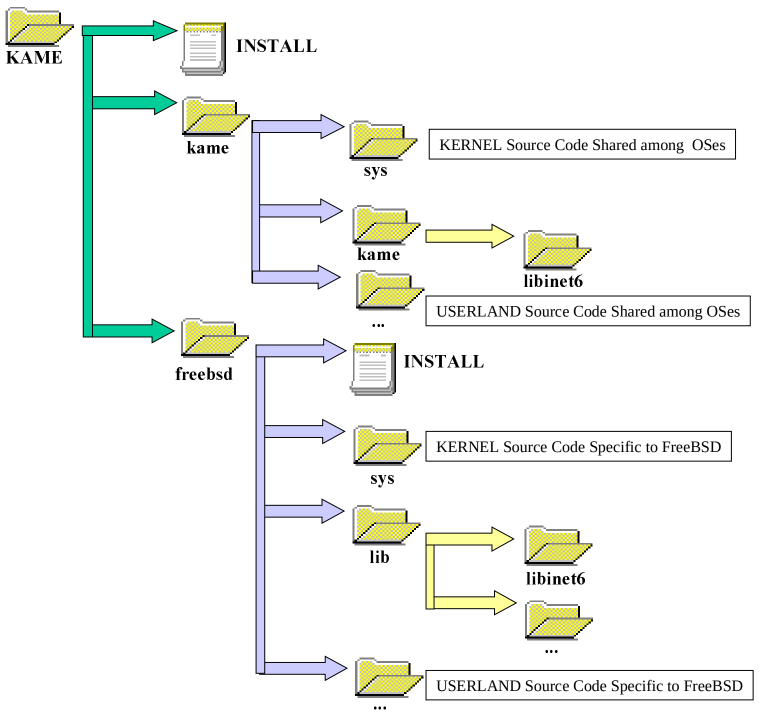
KAME
INSTALL
kame
sys
KERNEL Source Code Shared among OSes
kame
libinet6
...
USERLAND Source Code Shared among OSes
freebsd
INSTALL
sys
KERNEL Source Code Specific to FreeBSD
lib
libinet6
...
...
USERLAND Source Code Specific to FreeBSD

Figure 29 – TRT: Preparation of the Tree

To prepare the tree, follow the next instructions:

1. If you have already an installed KAME kit, you may need to remove the files under "/usr/local/v6" to avoid conflicts or misuse of the old files during building. We recommend you to remove the following, at least:
   ```
   /usr/local/v6/lib/*
   ```
   Also, do not forget to update "/usr/include" in step 5 (see below), otherwise the include files and source code become out-of-sync.
2. See the "VERSION" file and identify a symbol name for your target operating system. Along this document, we assume "freebsd" for explanation.
3. Be sure to have the complete KAME tree in the KAME tar.gz file.
4. Invoke the following:
   ```
   % make TARGET=freebsd prepare
   ```
   This will create necessary symbolic links for building (from the "freebsd" tree to the "kame" tree).

NOTE: You'll need the "perl" interpreter to invoke this. If you see any error message from the process, you skipped some of steps required. Do not proceed further.

5. Some parts of the KAME tree will be compiled, only if there are certain libraries installed beforehand.

   For instance, "usr.sbin/racoon" compiles only when OpenSSL 0.9.5a (or later) is installed beforehand, and shell search path is configured to reach "openssl" program.

   Please, install those libraries before building.

   KAME supports the following installations:

   - Your system has OpenSSL installed by default. FreeBSD 4.2-RELEASE, BSD/OS 4.2 and NetBSD 1.5 fall into this category.
   - A system with OpenSSL, installed manually with default installation path. This means that you gave no option to "./configure".

     In this case you should have /usr/local/ssl/include/openssl/des.h.
   - A system with OpenSSL, installed via FreeBSD ports system. Note that we do not support ports-current. Please stick to the ports directory came with the FreeBSD revision you are using, or the one came with KAME kit. In this case you should have /usr/local/include/openssl/des.h. We do not support ports with non-standard LOCALBASE/PREFIX.

   If you install OpenSSL with some other ways, the KAME tree may fail to find OpenSSL and omit compilation of, for example, usr.sbin/racoon.

6. Go down to the target directory that you have specified (e.g. "freebsd"), and read document named "INSTALL" (placed in somewhere like "freebsd/INSTALL").

<u>Installation procedure II – installation of FreeBSD 4.x</u>

You'll need to follow the steps below. They can be separated into two major sections: kernel build and userland build.

A. Backup

The following procedure replaces, or overwrites, the following files/directories:

- /kernel
- /usr/include

So, you may want to back them up before going through the steps.

A.1. Backup your kernel file, if you need to.

```
# cp /kernel /kernel.previous
```

A.2. Backup /usr/include directory, if you want an untouched tree to be preserved somewhere.

```
# cd /usr
# mkdir include.clean
# cd include.clean
# (cd ../include; tar Bpcf - . ) | tar Bpxf -
```

B. Kernel build

B.0. It is assumed that you are in the platform-specific directory (kame/freebsd4). If you are not, chdir to there.

B.1. Go down to sys/i386/conf.

```
% cd sys/i386/conf
```

B.2. Make a kernel configuration file, and invoke "/usr/sbin/config CONFIGFILE".

GENERIC.KAME includes KAME extensions as well as common FreeBSD definitions, so you may want to copy the file to begin with.

```
% cp GENERIC.KAME CONFIGFILE
% vi CONFIGFILE
% /usr/sbin/config CONFIGFILE
```

B.3. Build the kernel. The process will make kernel file named "kernel".

```
% cd ../../compile/CONFIGFILE
% make depend
% make
```

B.4. Install the kernel file to root directory, as root.

```
# make install
```

C. Userland build

We supply updated userland binaries for KAME-origin userland tools, and in some cases, for FreeBSD-origin userland tools.

C.0. It is assumed that you are in the platform-specific directory (kame/freebsd4). If you are not, chdir to there.

C.1. Invoke "make includes" as normal user, then "make install-includes" as root. This will populate KAME-ready include files into /usr/include.

```
% make includes
# make install-includes
```

C.2. As normal user (or root, if you prefer) perform "make".

```
% make
```

C.3. As root, perform "make install". This will install necessary userland tools into /usr/local/v6/{bin,sbin,whatever}. This should not replace existing IPv4-only userland tools, so it is safe.

```
# make install
```

D. Reboot

D.1. Reboot with the command you like.

```
# fastboot
```

E. Configurations, in short

E.0. You should configure IPv6 programs in this snapshot instead of the ones in original FreeBSD4.x, because some of them do not work on this snapshot. See E-3 for more details.

E.1. Most of configuration files are shipped with normal OS distribution, and are located in /etc. If you need any special configuration file for KAME-origin daemons, configuration files need to be placed in /usr/local/v6/etc. You may want to copy /usr/local/v6/etc/foo.sample into /usr/local/v6/etc/foo, and edit as necessary.

E.2. If you wish to use ALTQ, you may need to invoke /dev/MAKEDEV.altq.

```
# cd /dev
# sh MAKEDEV.altq all
```

E.3. Commands modified by KAME project are placed in /usr/local/v6/{bin,sbin}. For daily use, you will need to add both of (or either of) them into your command search path (consult manpage for your favourite shell) and /etc/rc's PATH definition.

Make sure to make them used before /usr/bin or /usr/sbin, otherwise you end up using OS- supplied commands, which might not work as mentioned in E-0.

In particular, FreeBSD 4.2's ifconfig(8) has a serious bug, which prevents IPv6 addresses from being configured. Since address configuration is usually done in the boot sequence, you should

Probably add the following line to your /etc/rc.conf:

```
PATH=/usr/local/v6/sbin:/usr/local/v6/bin:${PATH}
```

E.4. There are batch of documents installed into /usr/local/v6/man and /usr/local/v6/share/doc. They are more recent than those found in /usr/share/doc/IPv6. Please read them as necessary. If you wish to develop your own programs, we suggest you to read through supplied documents, RFCs, and other documents to learn how.

Configuring FAITH IPv6-to-IPv4 TCP relay

Introduction

FAITH is a IPv6-to-IPv4 TCP relay. It performs TCP relay just as some of firewall-oriented gateway does, but between IPv6 and IPv4 with address translation.

TCP connections have to be made from IPv6 node to IPv4 node. FAITH will not relay connections for the opposite direction.

To perform relays, FAITH daemon needs to be executed on a dual stack router between your local IPv6 site and outside IPv4 network. The daemon needs to be invoked per each TCP services (TCP port number).

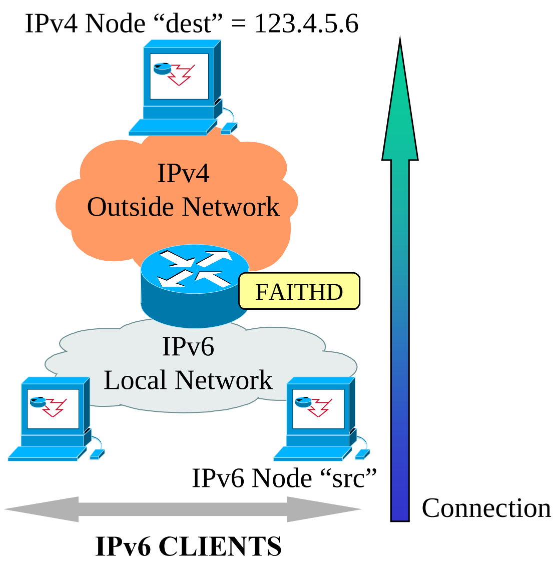

Figure 30 – Configuring FAITH IPv6-to-IPv4 TCP relay

You will have to allocate an IPv6 address prefix to map IPv4 addresses into.

The following description uses 3FFE:0501:FFFF:0000:: as example. (Please use a prefix which belongs to your site) FAITH will make it possible to make a IPv6 TCP connection From IPv6 node "src", toward IPv4 node "dest", by specifying FAITH-mapped address 3FFE:0501:FFFF:0000::123.4.5.6 (which is, 3FFE:0501: FFFF:0000:0000:0000:7B04:0506). The address mapping can be performed by hand, by special names server on the network, or by special resolver on the source node.

Setup

The following example assumes:

- You have assigned 3FFE:0501: FFFF:0000:: as FAITH address prefix.
- You are willing to provide IPv6-to IPv4 TCP relay for telnet.

<u>On the translating router on which faithd runs:</u>

1. If you have IPv6 TCP server for the telnet service, i.e., telnetd via inet6d, disable that daemon. Comment out the line from "inet6d.conf" and send the HUP signal to "inet6d".
2. Execute sysctl as root to enable FAITH support in the kernel.

```
# sysctl -w net.inet6.ip6.keepfaith=1
```

3. Route packets toward FAITH prefix into "faith0" interface.

```
# ifconfig faith0 up
# route add -inet6 3ffe:0501:ffff:0000:: -prefixlen 64 ::1
# route change -inet6 3ffe:0501:ffff:0000:: -prefixlen 64 -ifp faith0
```

4. Execute "faithd" by root as follows:

```
# faithd telnet /usr/local/v6/libexec/telnetd telnetd
```

1st argument is a service name you are willing to provide TCP relay. (it can be specified either by number "23" or by string "telnet")

2nd argument is a path name for local IPv6 TCP server. If there is a connection toward the router itself, this program will be invoked.

3rd and the following arguments are arguments for the local IPv6 TCP server. (3 rd argument is typically the program name without its path.)

More examples:

```
# faithd login /usr/local/v6/libexec/rlogin rlogind
# faithd shell /usr/local/v6/libexec/rshd rshd
# faithd ftpd /usr/local/v6/libexec/ftpd ftpd -l
# faithd sshd
```

If inetd(8) on your platform have special support for faithd, it is possible to setup faithd services via inetd(8). Consult manpage for details.

Regarding Routing issues:

5. Make sure that packets whose destinations match the prefix can reach from the IPv6 host to the translating router.

On the IPv6 host:

There are two ways to translate the IPv4 address to IPv6 address:

(a) Faked by DNS

(b) Faked by /etc/hosts.

6. Install "newbie" and set up FAITH mode. See kit/ports/newbie.
7. Add an entry into /etc/hosts so that you can resolve hostname into faked IPv6 address. For example, add the following line for www.netbsd.org:

```
3FFE:0501:FFFF:0000::140.160.140.252      www.netbsd.org
```

On the translating router on which faithd runs:

8. To see if "faithd" works, watch "/var/log/daemon".

   Note: please setup "/etc/syslog.conf" so that LOG_DAEMON messages are to be stored in "/var/log/daemon".

   Example:

```
daemon.*   /var/log/daemon
```

Access control

Since FAITH implements TCP relaying service, it is critical to implement proper access control to cope with malicious use. Bad guy may try to use your relay router to circumvent access controls, or may try to abuse your network (like sending SPAMs from IPv4 address that belong to you). Install IPv6 packet filter directives that would reject traffic from unwanted source. If you are using inetd-based setup, you may be able to use access control mechanisms in inetd.

Advanced configuration

If you would like to restrict IPv4 destination for translation, you may want to do the following:

```
    #route add -inet6 3FFE:0501:FFFF:0000::123.0.0.0 -prefixlen 104 ::1
    #route  change  -inet6  3FFE:0501:  FFFF:0000::123.0.0.0  -prefixlen  104  \  -ifp
faith0
```

By this way, you can restrict IPv4 destination to 123.0.0.0/8. You may also want to reject packets toward 3FFE:0501:FFFF:0000::/64, which is not in 3FFE:0501: FFFF:0000::123.0.0.0/104. By doing this, you will be able to provide your IPv4 web server to outside IPv6 customers, without risks of unwanted open relays.

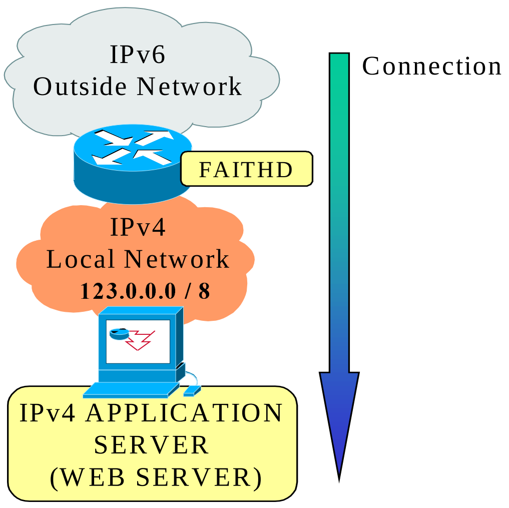

Figure 31- Configuring FAITH IPv4-to-IPv6 TCP relay

6. Conclusions

The IPv4 will not satisfy the increasing number of equipment that needs to be connected to the networks in the future. In addition, the new services and the users' demand to access to a set of services using mobile devices are also reasons for the IPv6 deployment.

The deployment of IPv6 must be done in a smooth way to prevent the disruption of the current IPv4 infrastructure. To allow a gradual transition, several mechanisms have been developed to guarantee the interoperability of IPv4 and IPv6.

The studies done to support this document lead us to conclude that the existing mechanisms satisfy the majority of the transition scenarios prerequisites.

All the transition mechanisms have different characteristics. Their applicability depends on the requirements defined to a determinate transition scenario. Also, one transition scenario may require the simultaneous use of several mechanisms. However, in this document it is not discussed the combination of several mechanisms.

It is combination of several evaluation criteria that lead to the choice of a determinate TM, as such as network topologies, management's cost and scalability. These issues were discussed in this document, which allow us to choose the appropriate TM for determinate real scenario.

The available implementations are still limited. Some of them are implemented on only certain operating systems. The majority of the implementations work in personal computers but in the future the manufacturers of network equipments will implement these mechanisms.

7. Glossary and Abbreviations

AH	Authentication Header
AIIH	Assignment of IPv4 global addresses to IPv6 Hosts
ACL	Access Control List
ALG	Application Level Gateway
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BIS	Bump-In-The-Stack
CGI	Common Gateway Protocol
DHCP	Dynamic Host Configuration Protocol
DHCPv4	Dynamic Host Configuration Protocol version 4
DHCPv6	Dynamic Host Configuration Protocol version 6
DNS	Domain Name System
DSTM	Dual Stack Transition Mechanism
DTI	Dynamic Tunneling Interface
ENR	Extensions Name Resolver
ESP	Encapsulation Security Payload
FQDN	Full Qualified Domain Name
HTML	Hypertext Mark-up Language
ICMP	Internet Control Message Protocol
ICMPv4	Internet Control Message Protocol version 4
ICMPv6	Internet Control Message Protocol version 6
IETF	Internet Engineering Task Force
IHL	Internet Header Length
IP	Internet Protocol
IPSec	Security Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISP	Internet Service Provider
LAN	Local Area Network
LONG	Laboratories Over Next Generation networks
MSRIPv6	Microsoft Research IPv6
MTU	Maximum Transmission Unit
NAT-PT	Network Address Translation – Protocol Translation
ND	Neighbor Discovery
NIC	Network Interface Card
PCI	Peripherals Communication Interface

QoS	Quality of Service
SIIT	Stateless IP/ICMP Translation Algorithm
SQL	Standard Query Language
TLA	Top Level Aggregate (referred to IPv6 addresses hierarchy)
TCP	Transmission Control Protocol
TEP	Tunneling End Point
TM	Transition Mechanisms
TOS	Type of Service
TRT	Transport Relay Translator
TTL	Time To Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VPN	Virtual Private Network
V4	Version 4 (referred to IPv4 issues)
V6	Version 6 (referred to IPv6 issues)

8. References

[Coffman] Internet growth: Is there a “Moore’s Law” for data traffic? K. G. Coffman and A. M. Odlyzko. AT&T Labs – Research. Preliminary version, July 11, 2000

[HITACHI] http://www.hitachi.co.jp/Prod/comp/network/pexv6-e.htm

[ISC2001] Internet Software Consortium, January 2001. www.isc.org.

[RFC792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC1928] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones, “SOCKS Protocol Version 5”, RFC1928, March 1996.

[RFC2365] D. Meyer, “Administratively Scoped IP Multicast”, RFC2365, July 1998.

[RFC2373] R. Hinden, S. Deering, “IP Version 6 Addressing Architecture”, RFC2373, July 1998.

[RFC2461] S. Thomson, T. Narten, Neighbor Discovery for IP Version 6 (IPv6), RFC2462, December 1998.

[RFC2462] S. Thomson, T. Narten, IPv6 Stateless Address Autoconfiguration, RFC2462, December 1998.

[RFC2463] Conta, A. and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6, (IPv6)", RFC 2463, December 1998.

[RFC2529] Carpenter, B., Jung, C., " Transmission of IPv6 over IPv4 Domains without Explicit Tunnels ", RFC 2529, March 1999.

[RFC2732] R. Hinden, B. Carpenter, L. Masinter “Format for Literal IPv6 Addresses in URL's”, RFC 2732, December 1999.

[RFC2765] Nordmark, E., " Stateless IP/ICMP Translation Algorithm (SIIT)", RFC 2765, February 2000.

[RFC2767] K. Tsuchiya, H. Higuchi, Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS), RFC 2767, February 2000.

[RFC2893] Gilligan, R., Nordmark, E.," Transition Mechanisms for IPv6 Hosts and Routers", RFC 2893, August 2000.

[RFC3056] B. Carpenter, K. Moore, Connection of IPv6 Domains via IPv4 Clouds, RFC3056, February 2001.

9. Bibliography

6to4

R. Gilligan, E. Nordmark, “Transition Mechanisms for IPv6 Hosts and Routers”, RFC 1933, April 1996

Gilligan, R., Nordmark, E.," Transition Mechanisms for IPv6 Hosts and Routers", RFC 2893, August 2000

B. Carpenter, K. Moore, “Connection of IPv6 Domains via IPv4 Clouds without Explicit Tunnels”

draft-ietf-ngtrans-6to4-03.txt , Oct 1999, http://www.6bone.net/misc/6to4.txt

Tunnel Broker

Alain Durand, A., Fasano, P., Lento, D., "IPv6 Tunnel Broker",draft-ietf-ngtrans-broker-06.txt, November 2000 (work in progress)

6over4

Carpenter, B., Jung, C., “Transmission of IPv6 over IPv4 Domains without Explicit Tunnels”, RFC2529, March 1999

DSTM

Bound, J., Toutain , L., Dupont, F., Afifi, H., Durand, A.,“Dual Stack Transition Mechanism (DSTM)”, draft-ietf-ngtrans-dstm-03.txt, February 2001 (work in progress)

Bound, J., Toutain , L., Dupont, F., Afifi, H., Durand, A, “Dual Stack Transition Mechanism (DSTM) extensions”, draft-ietf-ngtrans-dstmext1-aiih-00.txt, February 2001 (work in progress)

EURESCOM. ‘Transition Mechanisms Survey’. http://www.eurescom.de/~public-webspace/P1000-series/P1009/index.html

ENST Bretagne. DSTM implementation for FreeBSD (INRIA stack). Available at ftp://ftp.inria.fr/network/ipv6/freebsd/.

NAT-PT

G. Tsirtsis, P. Srisuresh, “Network Address Translation - Protocol Translation (NAT-PT)”, RFC 2766, February 2000

BIS

K. Tsuchiya, Y. Atarashi, Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS),

RFC 2767, February 2000

Nordmark, E., " Stateless IP/ICMP Translation Algorithm (SIIT)", RFC 2765, RFC 2765, February 2000

[HITACHI] http://www.hitachi.co.jp/Prod/comp/network/pexv6-e.htm

SOCKS64

Kitamura, H., “A SOCKS-based IPv6/IPv4 Gateway Mechanism”, draft-ietf-ngtrans-socks-gateway-05.txt, July 2000

Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., Jones, L., “SOCKS Protocol Version 5”, RFC1928, March 1996.

TRT

Hagino, J., Yamamoto, K., “An IPv6-to-IPv4 transport relay translator“, draft-ietf-ngtrans-tcpudp-relay-04.txt, April 2001

SIIT

Nordmark, E., " Stateless IP/ICMP Translation Algorithm (SIIT)", RFC 2765, RFC 2765, February 2000