A Dissertation Submitted on

MIGRATION FROM IPV4 TO IPV6

in the partial fulfillment of the requirements
for the award of the degree of

MASTER OF ENGINEERING
IN
COMPUTER TECHNOLOGY & APPLICATIONS

By:

Uttaran Dutta
Roll No. 8517
College Roll No. 06/CTA/04

Under the Guidance of

Dr. D Roy Choudhary

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
UNIVERSITY OF DELHI

CERTIFICATE

This is to certify that the dissertation titled **“Migration from IPv4 to IPv6”** has been submitted by **Uttaran Dutta,** student of final semester M.E. Computer technology & Applications of Delhi College Of Engineering as a part of his final year dissertation. The dissertation was carried under my guidance in the academic year 2005-06.

Dr. D Roy Choudhary
Dissertation Guide
Department of Computer Engineering
Delhi College of Engineering

ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extend my gratitude to everybody who helped me throughout the course of this thesis.
I would like to express my indebtedness to **D. Roy Choudhury**, Professor Department of Computer Engineering for the constant motivation and support during the duration of this thesis. It is my privilege and honor to have worked under his supervision. His invaluable guidance and helpful discussions at every stage of this project really helped me in materializing this thesis.
I would like to thank **Dr. Goldie Gabrani**, Head of the department, Computer Engineering, for providing facilities for this thesis.
I would also like to take this opportunity to present my sincere regards to my teacher viz. Dr S.K. Saxsena, Dr Asok De, Mr. Rajeev Kumar and Mrs. Rajni Jindal for their support and encouragement.
I would like to thank the staff and lab assistants of the Computer Centre at Delhi College of Engineering who kindly helped me in my endeavors to set up the IPv6 Network Lab within the Computer Centre.
Finally I am thankful to my parents, brother, friends and classmates for their unconditional support and motivation during this thesis.
Mail your comments, suggestion or reviews to me at: uttaranduutta@gamil.com

Uttaran Dutta
M.E. (Computer Technology & Applications)
College Roll No. 06/CTA/04
Delhi University Roll No. 8517

ABSTRACT

There has been an increased, and increasing, interest in IPv6. Here, the transition to IPv6 and co-existence of it with IPv4 in the academic networks will be examined. The Network setup in academic/technical institution environment has certain distinguishing characteristics e.g. large amount of variations is network types, topology and system, transparent accessible and programmable setup, support for old and outdated technologies etc. The migration from IPv4 to IPv6 of such academic/technical institution can be done by set by step execution gradual migration plan.

Firstly the new IPv6 network is set up in closed network within the institutions. There all the required server, software and applications are implemented or configured to run of the IPv6 are tried. It will be seen that mechanisms are ready or almost ready, and that new applications will drive the deployment of IPv6 especially in academic environment, though they may require some more time to mature.
The new IPv6 network then will open to communicate with the rest of in institution with the help of a NAPT translator gateway.

This is the point major testing and the comparison of IPv4 and IPv6 is done, their intercommunication with the help of the translator is also tested for its long term usage. Some major findings, such as the fast but bandwidth hungry nature of IPv6 will be reveled. This may be attributed to the in efficient implementations of IPv6 protocols in comparison to IPv4 protocols in to available network stacks today or it may also the true nature of the IPv6 standard. The translator in its test will show that it can only be setup as a stop-gap arrangement until the total migration to Ipv6.

The Future steps of migration such first implementation of the Dual Stack before the complete migration, due to unavailability of all compatible applications is discussed.

Table of Contents

List of Figures, Tables and Command/File Snapshots

Figures

Tables

Command/File Snapshots

1. Introduction

IPv6 is the new version of Internet Protocol which offers quite a few enhancements and possibilities. The transition to it has already begun even in operational networks and business networks, though IPv6 saw its birth place in the academic and the technical institutions, most of the technical institutions specifically in India are yet to migrate from IPv4 to IPv6.

The primary focus of this project is to briefly introduce the mechanisms that could be used, but more importantly, discuss why or why not certain mechanisms or approaches apply to the migration process. The process of migration discussed in the project was implemented in Delhi College of Engineering Bawana Road Delhi - 110042 .The network infrastructure and setup of Delhi College of Engineering is similar to any technical or academic institution in the country. Hence the project can be used as a general reference for any institution seeking the migration.

This project discusses steps to start using IPv6 network in an academic institution, different connectivity and translation mechanisms, and migration and co-existence approaches; a detailed introduction (refer RFCs 2460, 2466) to IPv6 as such is out of scope. To appreciate the process of migration and the definite advantages of IPv6 network over the IPv4, a brief introduction is as follows [7]

1.1IPv6

IPv6 (Internet Protocol version 6) or IPng (Internetworking Protocol next generation) is the new (proposed in 1994, drafted in 1998) network layer protocol standard set up IETF.IPv6 supports most of the same functions and applications that were supported in IPv4. Those functions that were not successful in IPv4 were either not included or improved in IPv6. The changes in IPv6 include:

- Increased address size from 32 bits to 128 bits. The larger address size supports a larger Internet base, a flexible and diverse Internet architecture and auto-configuration.IPv6 addresses the address space in the IP header to 128 bits to give the possibility of four billion x four billion x four billion x IPv4 address space [7]. This new address also allows for improvements in the routing structure of IP packets.
- The header format has been greatly simplified for IPv6. Some of the header fields have been removed and others have been moved to the optional IPv6 extension header, which is a separate header that travels between the normal IPv6 header and the transport-layer header in a packet. The IPv6 header is only twice the size of the IPv4 header, yet the IPv6 address is four times as large and it facilitates routing by allowing routers to quickly identify whether an option should be processed or ignored. More detailed discussion about the IPv6 header format and its comparison with IPv4 header is done at section 4.1.1.

- The creation of the Anycast address. The Anycast address is simply a unicast address that associated with more than one interface/node. When routed, the anycast address packet is only sent to the closest routable address.
- The addition of the scope field to the Multicast address. The scope field allows for the scalability of multicast routing. In this fashion, the Multicast address can be addressed globally specific; to routers, groups of routers, or special nodes, a detailed discussion of the multicast address is done at section 3.1.1.
- Quality-of-Service support in the Flow Label Field of the IPv6 Header. Packets may be labeled according to the type of traffic they contain, such as real-time service for videoconference links.
- Privacy and Security support is also included in IPv6. IPv6 extension headers now carry authentication options; this allows them to be processed by specific routers along the path, instead of forcing every router to process all of the options for every packet that it receives.

Automatic configuration with Internet Control Messaging Protocol version 6 (ICMPv6) allows interfaces to identify or verify addresses. Using ICMPv6 Neighbor Discovery; an interface can verify its Link-Local address. Using ICMPv6 router solicitation; an interface can identify an IPv6 prefix to create its unique global address, discussed in detail in 4.1.1.

From the above major upgrades that IPv6 has over IPv4 we can clearly identify the straight forward advantages IPv6 has over IPv4 like larger & efficiently managed address space ,enhanced security support, easy maintenance of administration TCP/IP ,elimination of the network address translation (NAT) role and better mobility support and Quality of Service(QoS) support

1.1.1The IPv6 address

128 bit IPV6 addresses belong to interfaces, not to nodes. A node can be identified by any unicast address associated with its interface. A single interface can be assigned more addresses of the same type or of different types (unicast, multicast, anycast).

Text Representation of Ipv6 Addresses

There are three conventional forms for representing IPv6 addresses as text strings:

- The preferred form is x:x:x:x:x:x:x:x, where the 'x's are the hexadecimal values of the eight 16-bit pieces of the address.

  Examples:

  FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
  080:0:0:0:8:800:200C:417A

  It is not necessary to write the leading zeros in an individual field, but there must be at least one numeral in every field (except for the case described in 2.).

- The use of "::" indicates multiple groups of 16-bits of zeros. The "::" can only appear once in an address. The "::" can also be used to compress the leading and/or trailing zeros in an address.

  Examples:

1080:0:0:0:8:800:200C:417A	a unicast address
FF01:0:0:0:0:0:0:101	a multicast address
0:0:0:0:0:0:0:1	the loopback address

  may be represented as:

1080::8:800:200C:417A	a unicast address
FF01::101	a multicast address
::1	the loopback address

- Another excepted format is x:x:x:x:x:x:d.d.d.d, where the 'x's are the hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the decimal values of the four low-order 8-bit pieces of the address (standard IPv4 representation).

  Examples:

  0:0:0:0:0:0:13.1.68.3
  0:0:0:0:0:FFFF:129.144.52.38

Text Representation of Ipv6 Address Prefixes

An IPv6 address prefix is represented by the notation: "IPv6-address/prefix-length" Where ipv6-address is an IPv6 address in any the notations listed in above and prefix-length is a decimal value specifying how many of the leftmost contiguous bits of the address comprise the prefix.

Example: The following are legal representations of the 60-bit prefix 12AB00000000CD3 (hexadecimal):

12AB:0000:0000:CD30:0000:0000:0000:0000/60
12AB::CD30:0:0:0:0/60
12AB:0:0:CD30::/60

1.2 The Migration

The Steps of migration

The Project of migration from of network from using ipv4 to using ipv6 in an academic setup like the Delhi College of Engineering can be described with the following goals in chronological order

1. **Identifying the environment**
   We identified the network environment required in an academic setup & analyzed the Ipv6 protocol with respect to an academic environment. Finding points to consider while planning the migration from ipv4 to ipv6 for such an environment.
2. **Setting up an ipv6 network**
   Implementing an ipv6 network in a closed small setup say called **The IPv6 Laboratory**. The above step consist of the following sub steps
   a. Identifying, procuring and implementing the required hardware and operating system platform for the implementation of ipv6 network.

   b. Identifying, various already general purpose application with ipv6 compatibility available, and install them.

3. **Connecting to the rest of the world**

   Connecting this setup to rest of the institution, an already existing ipv4 network and through this connect to the external network (i.e. the Internet).

   a. Identifying and study the various techniques of interoperability between an ipv4 and an ipv6 network.
   b. Implementing the most suitable of the above.

4. **Testing the connections**

   To do a comparative test of the three existing network connections viz:

   a. ipv6 to ipv6
   b. ipv4 to ipv4
   c. ipv6 to ipv4

   The above tests measure the performance, bandwidth requirement etc. of the network. We also indent to test the usage of the various ipv6 based applications.

5. **Gradual Shift**

   Planning a follow up for the gradual shift of the rest of the institution to ipv6 based network. The large scale migration cane be preceded by a Dual layer support first machine both supporting ipv4 and ipv6 protocol stacks. Another like this can only be possible by

   a. To implementing the proper APIs (application programming Interfaces) for the new protocol.
   b. To implement intermediate decoupling layer for the usage of earlier ipv4 based programs, applications and software.

2. The Academic Environment

2.1 Network Setup and its characteristics

The network environment at academic setup can be best described by the following keywords

1. Heterogeneous: The network is total mix of varied hardware platforms from low performance personal computers, for internet browsing and office application to high performance network server, file server etc. The operating system also with a mix of Windows 98, 2000, XP UNIX, GNU/Linux and Free BSDs.
2. Dynamically varying: Due to the inherent nature of some of the network Laboratories they dynamically vary in there nature and topology for e.g. the Network Laboratory where students practice to implement the various network systems, hence its networking structure keeps on varying.
3. Transparent: The network implementation should be transparent as rule so the research and engineering community can access it understand it and program for it
4. Not well written programs: It a typical engineering college network would require to support a large number of programs which are not well written, they directly interact with the network hardware and have high level modules which connect to the network protocol without use of any standard library or hardware.

Implications

The implications of the above findings on the migration procedure where:-

1. The Ipv6 was implemented both on Windows Xp and Fedora Core3.
2. The translators implemented here where open source C/C++ code.

3. Setting up IPv6 Network

3.1 Infrastructure Available

In order to setup the ipv6 network Laboratory the software and the hardware which provide ipv6 support are [4]

Operating systems

A number of operating systems support IPv6, including:

- Microsoft Windows Server 2003
- Microsoft Windows XP Service Pack 1 (SP1) and later
- Microsoft Windows CE .NET 4.1 and later
- IBM Advanced Interactive eXecutive (AIX®) 5.2 with maintenance level 3 (ML3) and later
- Hewlett Packard UNIX (HP-UX) 11i and later
- Sun Solaris 8.0 and later
- Red Hat Enterprise Linux (RHEL) Advanced Server with update 2.4 and later
- Fedora Core 3 and later
- Novell SUSE Enterprise Server 8.0 with SP3 and later
- Mac OS X 10.2 Jaguar and later

Older Windows versions do not support IPv6.

Application servers

Application servers that support IPv6 include:

- Microsoft Internet Information Services (IIS) 6.0
- IBM WebSphere® Application Server (WAS) 6.0 and later
- BEA WebLogic Server 9.0 and later

The following servers do not support IPv6:

- Macromedia JRun 4
- Oracle Application Server 9i

Databases

Numerous database systems support IPv6, including:

- IBM Informix® Dynamic Server (IDS) 10
- Microsoft SQL Server 2006
- Sybase OpenSwitch 15.0
- MySQL 5.0

The following databases do not support IPv6:

- IBM DB2® 8.2
- Oracle 10.1.0.4

Web browsers

Web browsers that support IPv6 include:

Mozilla 1.4 and later
Netscape 7.1 and later
Konqueror 1.4 and later
Mozilla Firefox 1.5 and later
Opera 7.2 and later

Internet Explorer does not support IPv6.

Web (HTTP) Server
Apache Web Server
Turix Web Script runner

3.2 Implementation of IPv6

3.2.1 Global Address assignment

The following IPv6 prefixes have been assigned to ERNET by APNIC:
2001:0E30::/32
According to the current global unicast addressing scheme the end-user Organizations that have a need to create subnets should obtain /48 prefixes. We plan to follow this recommendation [6].

Initial address allocations
This section summarizes the allocations valid at the time of issuing this report. The following addresses thus reflect the situation immediately with the new prefixes. ERNET will use standard procedures for further allocations.
The following table contains the Network/PoP prefixes.

PoP's Address Allocation

Terrestrial Network:
Delhi 2001:0E30:1800:/40
IIT Kanpur 2001:0E30:1400:/40
IISC 2001:0E30:1C00:/40
HUB 2001:0E30:1200:/40
Pune 2001:0E30:1A00:/40
Mumbai 2001:0E30:1600:/40
UoH 2001:0E30:1E00:/40
IIT Chennai 2001:0E30:1100:/40
Kolkatta 2001:0E30:1900:/40

VSAT Network:
VSAT Delhi 2001:0E30:2800:/40
VSAT Bangalore 2001:0E30:2400:/40

Based on the above a PoP can further distribute /48 addresses.

An application for global ipv6 for the Delhi College of Engineering has yet to be filed to ERNET.

3.2.2 Requirements for the Ipv6 Setup

Hardware:
PCs with 1.6 GHz higher processor clock speed recommended; 300 MHz minimum required (single or dual processor system); Intel Pentium/Celeron family, or AMD K6/Athlon/Duron family, or compatible processor recommended.
192 megabytes (MB) of RAM or higher recommended (128 MB minimum supported; may limit performance and some features)
10 gigabytes (GB) of available hard disk space
Super VGA (800 x 600) or higher-resolution video adapter and monitor

Network adapter appropriate for the type of local-area, or wide-area network and access to an appropriate network infrastructure.

10/100 fast Ethernet switch.

10/100 mbps cables and RJ-45 connectors.

Software:
Fedora Core 1/2/3 Operating system Installed complete.
DHCPv6 from sourcforge.net, dibbler-0.4.0-win32 , dibbler-0.4.0-linux.tar

Application:
Ethereal (network protocol analyzer), tcpdump, ifconfig, ping6, traceroute6, and network setting files.

3.2.3 Supporting IPv6 in Fedora Core 3

IPv6 support is enabled as a built-in kernel feature in Fedora Core 3.

- **Enabling global IPv6 support:**

  /etc/sysconfig/network file: NETWORKING_IPV6="YES"

- **Enabling IPv6 support on a particular interface:**

  /etc/sysconfig/network-scripts/ifcfg-etho file: IPV6INIT="yes"

- **Configure IPv6 interface address:**

  /etc/sysconfig/network-scripts/ifcfg-etho file:

  IPV6ADDR="2001:e30:DEAD:BEEF"

- **Default Gateway:**

  /etc/sysconfig/network file: IPV6_DEFAULTGW=IPv6 address[%interface]

- **Default route configuration:**

  /etc/sysconfig/static-routes-ipv6 file:

  eth0   2001::/3   3ffe:ffff:1234:0002:0:0:0:1

3.2.4 Testing the Configuration of IPv6 in Fedora Core 3

After enabling IPv6 support, we can verify the network interfaces by typing ifconfig at the command prompt [9]. The underlined lines below show the IPv6 address of the system. The result obtained is as follows:

```
[root@ns root]# ifconfig

eth1      Link encap:Ethernet  HWaddr 00:08:C7:CF:9D:0A
          inet addr:202.141.40.26  Bcast:202.141.40.63
Mask:255.255.255.192
          inet6 addr: fe80::208:c7ff:fecf:9d0a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:933227 errors:6 dropped:0 overruns:0 frame:6
          TX packets:771601 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:102613930 (97.8 Mb)  TX bytes:213553354 (203.6 Mb)
          Interrupt:21 Base address:0x4000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:10671 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10671 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1163707 (1.1 Mb)  TX bytes:1163707 (1.1 Mb)
```

Command Snapshot1:ifconfig

We can also test by pinging IPv6 loopback address ,our link local address and.

```
[root@ns root]# fe80::208:c7ff:fecf:9d0a
PING fe80::208:c7ff:fecf:9d0a 56 data bytes
64 bytes from 2001:e30:1400:1::5: icmp_seq=1 ttl=64 time=1.08 ms
64 bytes from 2001:e30:1400:1::5: icmp_seq=2 ttl=64 time=0.418 ms
64 bytes from 2001:e30:1400:1::5: icmp_seq=3 ttl=64 time=0.331 ms
64 bytes from 2001:e30:1400:1::5: icmp_seq=4 ttl=64 time=0.343 ms
64 bytes from 2001:e30:1400:1::5: icmp_seq=5 ttl=64 time=0.379 ms
--- 2001:0e30:1400:1::5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 0.331/0.510/1.083/0.289 ms
```

Command Snapshot2: ping of local Ipv6 address

3.2.5 Supporting IPv6 in Windows XP

IPv6 is best supported on Windows XP with Service pack 1.If Sp1 is not installed it can be downloaded from Microsoft web site.

We can now check our network configuration using the command ipconfig on Command Prompt and see if our machine has got an IPv6 address:
The underlined lines below show the IPv6 address of a Windows XP system. The result of ipconfig command is as follows:

```
C:\Documents and Settings\Administrator>ipconfig

Windows IP Configuration
Ethernet adapter Local Area Connection:

 Connection-specific DNS Suffix  . :
IP Address. . . . . . . . . . . . : 172.26.105.163
Subnet Mask . . . . . . . . . . . : 255.255.0.0
IP Address. . . . . . . . . . . . : fe80::202:44ff:fe84:c18f%6
IP Address. . . . . . . . . . . . : 2001:e30:1401:5:c574:a11d:96bb:20a0
IP Address. . . . . . . . . . . . : 2001:e30:1401:5:202:44ff:fe84:c18f
IP Address. . . . . . . . . . . . : fe80::202:44ff:fe84:c18f%4
Default Gateway . . . . . . . . . : 172.26.1.254
                                    fe80::20d:65ff:fef9:7090%4

Tunnel adapter Teredo Tunneling Pseudo-Interface:
        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : fe80::5445:5245:444f%5
        Default Gateway . . . . . . . . . :

Tunnel adapter Automatic Tunneling Pseudo-Interface:
        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : fe80::5efe:172.26.105.163%2
        Default Gateway . . . . . . . . . :
```

Command Snapshot3: ipconfig

3.3 Setting up IPv6 network resources

We have to implement the following services in Fedora Core 3 and Windows XP over IPv6 test bed

- DHCPv6
- Multicasting
- DNS server

3.3.1DHCPv6

DHCPv6 is actually DHCP for IPv6. DHCPv6 is the "stateful address autoconfiguration protocol" and the "stateful autoconfiguration protocol" referred to in "IPv6 Stateless Address Autoconfiguration"

Protocols and Addressing of DHCPv6

Clients and servers exchange DHCP messages using UDP. The client uses a link-local address or addresses determined through other mechanisms for transmitting and receiving DHCP messages [1].
DHCP servers receive messages from clients using a reserved, link-scoped multicast address. A DHCP client transmits most messages to this reserved multicast address,

so that the client need not be configured with the address or addresses of DHCP servers.
To allow a DHCP client to send a message to a DHCP server that is not attached to the same link, a DHCP relay agent on the client's link will relay messages between the client and server. The operation of the relay agent is transparent to the client.

Once the client has determined the address of a server, it may under some circumstances send messages directly to the server using unicast.

Comparison between DHCPv4 and DHCPv6
There are three key reasons for the differences:

- IPv6 inherently supports a new model and architecture for communications and autoconfiguration of addresses.
- DHCPv6 benefits from the new IPv6 features.
- New features were added to support the expected evolution and the existence of more complicated Internet network service requirements.

Changes due to IPv6 Architecture:

- The link-local address permits a node to have an address immediately when the node boots, which means all clients, have a source IP address at all times to locate a server or relay agent on the local link.
- The need for BOOTP compatibility and broadcast flags is removed.
- Multicast and address scoping in IPv6 permit the design of discovery packets that would inherently define their range by the multicast address for the function required.
- Stateful autoconfiguration has to coexist and integrate with stateless autoconfiguration supporting Duplicate Address Detection and the two IPv6 lifetimes, to facilitate the dynamic renumbering of addresses and the management of those addresses.
- Multiple addresses per interface are inherently supported in IPv6.
- Many DHCPv4 options are unnecessary now because the configuration parameters are either obtained through IPv6 Neighbor Discovery or the Service Location protocol.

Changes due to DHCPv6 Architecture:

- The message type is the first byte in the packet.
- IPv6 Address allocations are now handled in a message extension as opposed to the message header.
- Client/Server bindings are now mandatory and take advantage of the client's link-local address to always permit communications either directly from an on-link server, or from a remote server through an on-link relay-agent.
- Servers are discovered by a client solicit, followed by a server or relay-agent advertisement.
- The client will know if the server is on-link or off-link.
- The on-link relay-agent locates remote server addresses from system configuration or by the use of a site wide multicast packet.

- ACKs and NAKs are not used.
- The server assumes the client receives its responses unless it receives a retransmission of the same client request. This permits recovery in the case where the network has faulted.
- Clients can issue multiple, unrelated DHCP Request messages to the same or different servers.
- The function of DHCPINFORM is inherent in the new packet design; a client can request configuration parameters other than IPv6 addresses in the optional extension headers.
- Clients MUST listen to their UDP port for the new Reconfigure message from servers.
- New extensions have been defined.

With the above mentioned changes, we can support new user features in DHCPv6, including

- Configuration of Dynamic Updates to DNS
- Address deprecation, for dynamic renumbering.
- Relays can be preconfigured with server addresses, or use of multicast.
- Authentication
- Clients can ask for multiple IP addresses.
- Addresses allocated with too-long lifetimes can be reclaimed using the Reconfigure message.
- Integration between stateless and stateful address autoconfiguration.
- Enabling relay-agents to locate remote servers for a link.

Differences between DHCPv4 and DHCPv6

The following lists the differences between DHCPv4 and DHCPv6:
Unlike DHCPv4, IPv6 address allocation in DHCPv6 is handled using a message option.
The message types, such as DHCPDISCOVER and DHCPOFFER supported by DHCPv4 are removed in DHCPv6. Instead, DHCPv6 servers are located by a client SOLICIT message followed by a server ADVERTISE message.
Unlike DHCPv4 clients, DHCPv6 clients can request multiple IPv6 addresses.

Multicast Addresses used by DHCPv6

DHCP makes use of the following multicast addresses:

- All_DHCP_Relay_Agents_and_Servers (FF02::1:2)

This is a link-scoped multicast address used by a client to communicate with neighboring relay agents and servers. All servers and relay agents are members of this multicast group.

- All_DHCP_Servers (FF05::1:3)

This is a site-scoped multicast address used by a relay agent to communicate with servers, either because the relay agent wants to send messages to all servers or because it does not know the unicast addresses of the servers. Note that in order for a relay agent to use this address, it must have an address of sufficient scope to be reachable by the servers. All servers within the site are members of this multicast group.

UDP Ports used by DHCPv6
Clients listen for DHCP messages on UDP port 546. Servers and relay agents listen for DHCP messages on UDP port 547.

DHCP Message Types
DHCP defines the following message types. The numeric encoding for each message type is shown in parentheses.

SOLICIT (1) A client sends a Solicit message to locate servers.

ADVERTISE (2) A server sends an Advertise message to indicate that it is available for DHCP service, in response to a Solicit message receive from a client.

REQUEST (3) A client sends a Request message to request configuration parameters, including IP addresses, from a specific server.

CONFIRM (4) A client sends a Confirm message to any available server to determine whether the addresses it was assigned are still appropriate to the link to which the client is connected.

RENEW (5) A client sends a Renew message to the server that originally provided the client's addresses and configuration parameters to extend the lifetimes on the addresses assigned to the client and to update other configuration parameters.

REBIND (6) A client sends a Rebind message to any available server to extend the lifetimes on the addresses assigned to the client and to update other configuration parameters; this message is sent after a client receives no response to a Renew message.
REPLY (7) A server sends a Reply message containing assigned addresses and Configuration parameters in response to a Solicit, Request, Renew, Rebind message received from a client. A server sends a Reply message containing configuration parameters in response to an Information-request message. A server sends a Reply message in response to a Confirm message confirming or denying that the addresses assigned to the client are appropriate to the link to which the client is connected. A server sends a Reply message to acknowledge receipt of a Release or Decline message.

RELEASE (8) A client sends a Release message to the server that assigned addresses to the client to indicate that the client will no longer use one or more of the assigned addresses.

DECLINE (9) A client sends a Decline message to a server to indicate that the client has determined that one or more addresses assigned by the server are already in use on the link to which the client is connected.

RECONFIGURE (10) A server sends a Reconfigure message to a client to inform the client that the server has new or updated configuration parameters, and that the client is to initiate a Renew/Reply or Information-request/Reply transaction with the server in order to receive the updated information.

INFORMATION-REQUEST (11) A client sends an Information-request message to a server to request configuration parameters without the assignment of any IP addresses to the client.

RELAY-FORW (12) A relay agent sends a Relay-forward message to relay messages to servers, either directly or through another relay agent. The received message, either a client message or a Relay-forward message from another relay agent, is encapsulated in an option in the Relay-forward message.

RELAY-REPL (13) A server sends a Relay-reply message to a relay agent containing a message that the relay agent delivers to a client. The Relay-reply message may be relayed by other relay agents for delivery to the destination relay agent. The server encapsulates the client message as an option in the Relay-reply message, which the relay agent extracts and relays to the client.

Format of DHCP Message

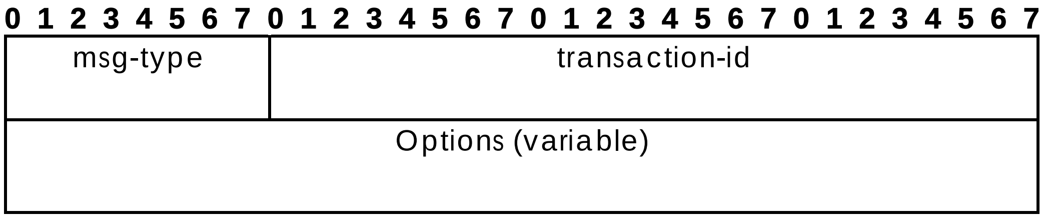


Figure 1, Format of DHCP Message

The DHCP message has the basic format as above. It contains

msg-type	Identifies the DHCP message type
transaction-id	The transaction ID for this message exchange.
options	Options are used to carry additional information and parameters in DHCP messages.

Format of DHCP Options

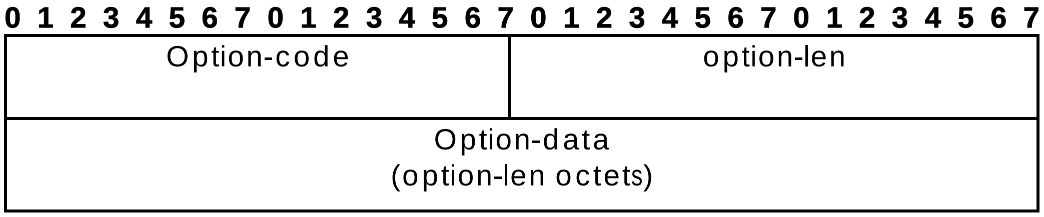


Figure 2, Format of DHCP Options

DHCP options have the following contents:

- **option-code** An unsigned integer identifying the specific option type carried in this option.
- **option-len** An unsigned integer giving the length of the option-data field in this option in octets.
- **option-data** The data for the option; the format of this data depends on the definition of the option.

DHCPv6 options are scoped by using encapsulation. Some options apply generally to the client, some are specific to an IA, and some are specific to the addresses within an IA.

DHCP Unique Identifier (DUID)

Each DHCP client and server has a DUID. DHCP servers use DUIDs to identify clients for the selection of configuration parameters and in the association of IAs with clients. DHCP clients use DUIDs to identify a server in messages where a server needs to be identified Clients and servers MUST treat DUIDs as opaque values and MUST only compare DUIDs for equality.
A DUID consists of a two-octet type code represented in network byte order, followed by a variable number of octets that make up the actual identifier. A DUID can be no more than 128 octets long (not including the type code).The following types are currently defined:

1 Link-layer address plus time (DUID-LLT)
2 Vendor-assigned unique ID based on Enterprise Number (DUIDEN)
3 Link-layer addresses. (DUID-LL)

Identity Association

An Identity-Association (IA) is a construct through which a server and a client can identify, group, and manage a set of related IPv6 addresses. Each IA consists of an IAID and associated configuration information.
A client must associate at least one distinct IA with each of its network interfaces for which it is to request the assignment of IPv6 addresses from a DHCP server. The client uses the IAs assigned to an interface to obtain configuration information from a server for that interface. Each IA must be associated with exactly one interface.
The IAID uniquely identifies the IA and must be chosen to be unique among the IAIDs on the client. The IAID is chosen by the client. For any given use of an IA by the client, the IAID for that IA must be consistent across restarts of the DHCP client. The client may maintain consistency either by storing the IAID in non-volatile storage or by using an algorithm that will consistently produce the same IAID as long as the configuration of the client has not changed. There may be no way for a client to maintain consistency of the IAIDs if it does not have non-volatile storage and the client's hardware configuration changes.
The configuration information in an IA consists of one or more IPv6 addresses along with the times T1 and T2 for the IA. Each address in an IA has a preferred lifetime and

a valid lifetime. The lifetimes are transmitted from the DHCP server to the client in the IA option.

Operation of DHCPv6

The client sends SOLICIT message to link-local multicast address. To this message the server responds by sending a unicast address to the client. The message Request / Reply provides configuration information to the client, but sends no address. The Confirm / Reply message will assist in determining whether client has moved to a different location. Reconfigure initiates a client reconfiguration. Renew message is sent by the client to extend the lease time of the same address contained by the client, if the client gets Reply message in response to its Renew message it will keep the IP address, else client will send Rebind message. In Rebind message the client can request for a new IP address from the same or different server.

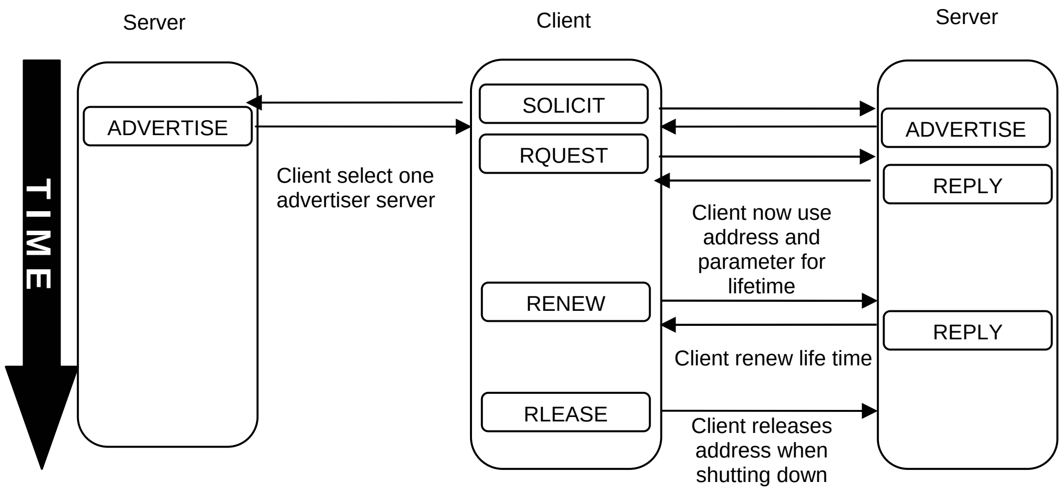

Server
Client
Server
TIME
ADVERTISE
SOLICIT
RQUEST
ADVERTISE
REPLY
Client select one advertiser server
Client now use address and parameter for lifetime
RENEW
REPLY
Client renew life time
RLEASE
Client releases address when shutting down

Figure3: Operations of DHCPv6

Ongoing Project on DHCPv6:

- The following are the prerequisites to install the DHCPv6 2.001 software depot on the HP-UX 11i v1 operating system:
    - o 32- or 64-bit HP-UX 11i v1 (B.11.11) PA-RISC system
    - o IPv6 bundle (IPv6NCF11i) or Transport Optional Upgrade Release (TOUR) 2.0/2.2
    - o Transport patch - PHNE_28895 (or its superseded patch)
    - o We can download the IPv6 software from the HP software depot at http://www.software.hp.com (Search for the HP-UX IPv6 Product)
    - o We can download the PHNE_28895 Transport patch (or its superseded patch) from http://www.itrc.hp.com.
- Dibbler is a portable DHCPv6 implementation on Linux 2.4/2.6 and Windows XP and Windows 2003.
  This project was started as master thesis by Tomasz Mrugalski and Marek Senderski of Computer Science faculty on Gdansk University of Technology. [11

- IP version 6 Dynamic Host Configuration Protocol
  Development Status: 3 - Alpha
  Intended Audience: Developers, Telecommunications Industry
  License: BSD License
  Operating System: All POSIX (Linux/BSD/UNIX-like OSes), Linux
  Programming Language: C
  Topic: Networking
  Translations: English

Implementation of DHCPv6

DHCPv6 have been implemented in our project, using two methods which have been described below. For its implementation we have made the network setup using three systems connected using a switch. The three systems were in the same LAN as they were connected using switch. In the three systems, one system worked as the DHCPv6 server and the rest two systems as the DHCpv6 client.

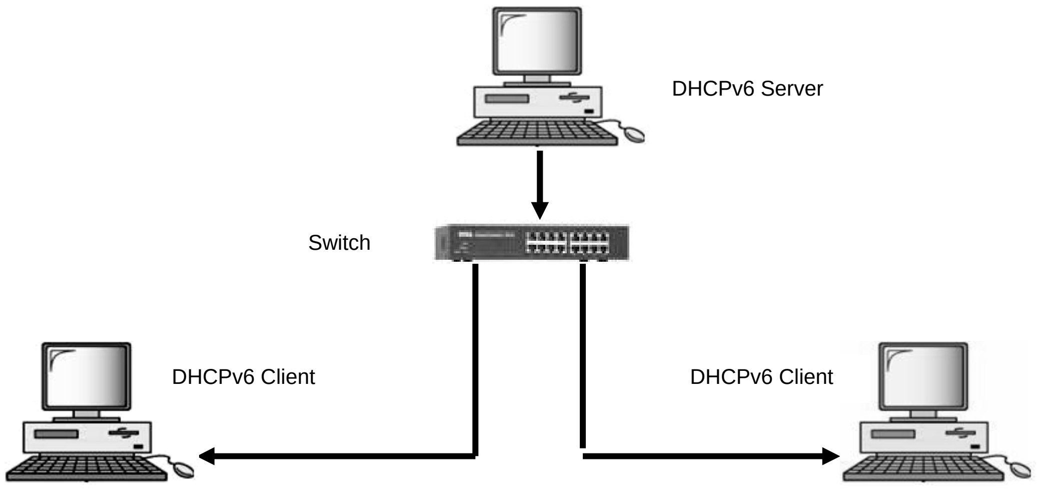


Figure4:DHCPv6 Implementation Setup

DHCPv6 from Sourceforge.net

For the first method of implementation of DHCPv6 we have downloaded the following tar file from the website sourceForge.net
dhcp-0.10.tgz

This tar file is a the source code of DHCPv6. So, we have to first configure it before installing it. After downloading the file, we have to extract it using the following command:
tar –zxvf dhcp-0.10.tgz

After that we have to change our directory to the above extracted directory, which is dhcp-0.10. Then make some changes two files in this directory. The two files are config.c and dhcp6s.c.

Now comes the process of configuring the software using the command:

./configure

To install the DHCPv6 server in the system, we use the following command:
make dhcp6s

And to install the DHCPv6 client in the system, we use the following command:
make dhcp6c

Then making changes in the configuration files made DHCPv6 to work.

Configurations of DHCPv6 in Fedora Core 3
The following files are configured in fedora Core 3 :
For Server configuration[9],
/etc/sysconfig/dhcp6s and /etc/dhcp6s.conf.
For Client configuration,
/etc/sysconfig/ network-scripts/ifcfg-eth0 and /etc/dhcp6c.conf.

Dhcpv6 server
In the file /etc/sysconfig/dhcp6s, we have to specify the interface for dhcp6s, which is as follows

```
DHCP6SIF=eth0
```

The file /etc/dhcp6s.conf is as follows

```
  interface eth0
  {
       server-preference 255;
       renew-time 60;
       rebind-time 90;
       prefer-life-time 130;
       valid-life-time 200;
       allow rapid-commit;
       link AAA {
               pool{
                       range 2001:0E30:1402:1::4 to
                       2001:0E30:1402:1::ffff/64;
                       prefix 2001:0E30:1402::/48;
               };
       };
 };
```

File SnapShort4:dhcpv6.conf

Dhcpv6 client
In the file /etc/sysconfig/network-scripts/ifcfg-eth0, we have to specify the following

```
IPV6INIT=yes
DHCP6C=yes
```

The file /etc/dhcp6c.conf is as follows

```
      interface eth0
           {
                 #information-only;
                send rapid-commit;
                 #request prefix-delegation;
                 #request temp-address;

                  address
                      {
                           2001:0E30:1402:1:9656:3:4:56/64;
                       };
               };
```

File SnapShort5:dhcpv6.conf

Testing of DHCPv6 in Fedora Core 3

Dhcpv6 server

Running of DHCPV6 server in debug mode in foreground, we have to use the following command:

```
[root@pc08 dhcp-0.10]# ./dhcp6s -dDf eth0
...........
...........
Jun/15/2006 00:47:32 add lease for 2001:e30:1402:1::5/64 iaid 3820474368
with preferlifetime 130 with validlifetime 200
Jun/15/2006 00:47:32 hash_add an iaidaddr 3820474368 for client duid
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e
............
............
```

Command Snapshort6 #./dhcp6s -dDf eth0 .

To view the full output of the command refer the Appendix

In the above underlined line we see that the DHCPv6 server has given the address to the DHCPv6 client.

The following command shows the IP address of the server:

```
[root@pc08 dhcp-0.10]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:D0:B7:E3:D1:0E
          inet addr:172.31.5.80  Bcast:172.31.255.255  Mask:255.255.0.0
          inet6 addr: 2001:e30:1401:2:2d0:b7ff:fee3:d10e/64 Scope:Global
          inet6 addr: fe80::2d0:b7ff:fee3:d10e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:76915 errors:0 dropped:0 overruns:0 frame:0
          TX packets:999 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5225109 (4.9 MiB)  TX bytes:141594 (138.2 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1218 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1218 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1380200 (1.3 MiB)  TX bytes:1380200 (1.3 MiB)
```

Command Snapshort7 :ifconfig

Dhcpv6 client

Running of DHCPV6 client in debug mode in foreground, we have to use the following command [10]:

```
[root@pc08 dhcp-0.10]# ./dhcp6s -dDf eth0
...........
...........
Jun/15/2006 00:44:00 add an address 2001:e30:1402:1::5 on eth0
Jun/15/2006 00:44:00 renew time 60, rebind time 90
...........
...........
```

Command Snapshort8 #./dhcp6s -dDf eth0 .

To view the full output of the command refer the Appendix

In the above underlined line we see that the DHCPv6 client have received the the address from the DHCPv6 server.

The following command shows the IP address of the client:

```
[root@pc212 ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:D0:B7:E3:D1:3E
          inet addr:172.31.5.78  Bcast:172.31.255.255  Mask:255.255.0.0
          inet6 addr: 2001:e30:1402:1::5/64 Scope:Global
          inet6 addr: 2001:e30:1401:2:2d0:b7ff:fee3:d13e/64 Scope:Global
          inet6 addr: fe80::2d0:b7ff:fee3:d13e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:67446 errors:0 dropped:0 overruns:0 frame:0
          TX packets:115 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4128924 (3.9 MiB)  TX bytes:9610 (9.3 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:144 errors:0 dropped:0 overruns:0 frame:0
          TX packets:144 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:10018 (9.7 KiB)  TX bytes:10018 (9.7 KiB)
```

Command Snapshort9 #ifconfig

Using Red Hat Package manager

In the second method of implementation of DHCPv6 we have downloaded the following rpm file from the website:

dhcpv6-0.10-11_FC3.i386.rpm

Then we have installed the rpm file and making changes in the configuration files made DHCPv6 to work.

Installation of DHCPv6 in Fedora Core 3

Dhcpv6 server

We have to update the existing dhcpv6s in Fedora Core 3, which contains a lot of bugs, with dhcpv6-0.10-11_FC3.i386.rpm using the following command:

```
# rpm -U  dhcpv6-0.10-11_FC3.i386.rpm
```

Then we have to create a database directory. It is done using the command:

```
#mkdir /var/db/dhcpv6
```

After that a sample server configuration file is copied in /etc directory using the command:

```
# cp dhcp6s.conf  /etc/dhcp6s.conf
```

Dhcpv6 client

We also have to update the existing dhcpv6c in Fedora Core 3, which contains a lot of bugs, with dhcpv6_client-0.10-11_FC3.i386.rpm using the following command:

```
# rpm -U  dhcpv6_client-0.10-11_FC3.i386.rpm
```

Then a sample client configuration file is copied in /etc directory using the command:

```
# cp dhcp6c.conf  /etc/dhcp6c.conf
```

Configurations of DHCPv6 in Fedora Core 3

The files of DHCPv6 server and client are configured as it was configured in dhcpv6 sourceforge.net. We have to make changes in /etc/sysconfig/dhcp6s and /etc/dhcp6s.conf file for server configuration and in /etc/sysconfig/ network-scripts/ifcfg-eth0 and /etc/dhcp6c.conf files for client configuration in Fedora Core 3.

Testing of DHCPv6 in Fedora Core 3

We have to start the server daemon in debug mode in foreground using the command:

```
            #dhcp6s –dDf eth0
```

Then we have to restart the network service of client, using the following command:

```
#service network restart
```

Then we start the client daemon in debug mode in foreground using the command:

```
#dhcp6c –dDf eth0
```

To see the address assigned to the DHCP client by the DHCP server we use the command:

```
# ifconfig
```

Dibbler

Dibbler is a portable DHCPv6 solution. It features server, client and relay. Currently there are ports available for Windows XP and 2003 and Linux systems. It supports both stateful (i.e. IPv6 address granting) and stateless (i.e. options granting) autoconfiguration. Besides basic functionality1, it also offers several enhancements, e.g. DNS servers and domain names configuration. Dibbler is a freeware.

As for now, Dibbler offers these features:

- Basic server discovery and address assignment messages
- Best server discovery
- Many servers support
- Relay support
- Unicast communication
- Address renewal messages
- Duplicate address detection messages
- Power failure/crash support
- IA Option
- Rapid Commit Option

Except RFC 3315-specified behavior, Dibbler also supports several enhancements:

- DNS Servers Option
- Domain Name Option
- Time Zone Option
- NTP Servers Option
- SIP Servers Option
- SIP domain name
- NIS, NIS+ server Option
- NIS, NIS+ domain name Option
- Option renewal mechanism (Lifetime Option)

Implementation of Dibbler

For the implementation of Dibbler we had the above network setup as the above DHCPv6 implementation.

Installation of Dibbler in Windows

Dibbler runs on Windows XP and 2003. In XP systems, at least Service Pack 1 is required. To install Dibbler server and client as services, administrator privileges are required. Both Client and server are installed in the same way. Installation method is different in Windows XP and Linux systems.

Fedora Core 3 installation

We have to download the RPM packages dibbler-0.3.1-1.i386.rpm for Fedora Core 3.Then to install rpm package, we have to issue the following command:

```
rpm -i dibbler-0.3.1-1.i386.rpm
```

To start the dibbler either as a server or as a client, the following commands are used:

```
dibbler-server start
dibbler-client start
```

Start parameter needs a little comment. It instructs Dibbler to run in daemon mode – detach from console and run in the background. Dibbler will present its messages on the console. To finish it, we press ctrl-c.

To stop server or client running in daemon mode, type:

```
dibbler-server stop
dibbler-client stop
```

To see, if client or servers are running, type:

```
dibbler-server status
dibbler-client status
```

Windows XP installation

We have to download the portable dibbler setup, which is dibbler-0.4.0-win32 for Windows XP. Then install it following screen instructions. Dibbler will be installed and all required links will be placed in the Start menu.

Configuration files of Dibbler

For Dibbler Server

For Fedora Core 3, we have to make changes in /etc/dibbler/server.conf file and in Windows XP, we have to make changes in config file of server in dibbler. We have made the following changes which are as follows:

```
log-level 7
log-mode short
iface eth0{
   T1 1000
   T2 2000
   class{
           pool 2001:0e30:1402:1::4-2001:0e30:1402:1::ffff
        }
           }
```

File Snapshort10/etc/dibbler/server.conf (Server)

For Dibbler Client

For Fedora Core 3, we have to make changes in /etc/dibbler/client.conf file and in Windows XP, we have to make changes in config file of client in dibbler. We have made the following changes which are as follows:

```
log-mode short
log-level 7

iface eth0
{    ia
      {
          address
            {
             }
        }
} log-mode short
log-level 7

iface eth0
{
    ia
      {
          address
            {
             }
        }
}
```

File Snapshort11/etc/dibbler/client.conf (Client)

Running Dibbler

For Dibbler Server

To run Dibbler server in foreground, we use the following command[10]:

```
[root@pc08 dibbler]# ./dibbler-server run
........
........
Client requested ::, got 2001:e30:1402:1::d17d (IAID=2,
pref=180 0,valid=3600).
........
........
20:03 Notice    Sending REPLY on eth0/2,transID=0x273fdd, opts: 1 3 2, 0
relays.
```

Command Snapshort12./dibbler-server run

To view the full output of the command refer the Appendix

The above underlined line shows that a client has requested an address from the server and obtained it.

For Dibbler Client

To run Dibbler client in foreground, we use the following command:

```
[root@pc08 dibbler]# ./dibbler-client run
........
........
16:31 Notice    Address 2001:e30:1402:1::d17d added to eth0/2 interface
........
........
```

Command Snapshort13./dibbler-client run

To view the full output of the command refer the Appendix

The above underlined line shows that the client has received the address from the Dibbler server. We can also check this by seeing the IP address of the client system. The above address is seen to be obtained by the client using the following command:

```
[root@pc212 ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:D0:B7:E3:D1:3E
          inet addr:172.31.5.78  Bcast:172.31.255.255  Mask:255.255.0.0
          inet6 addr: 2001:e30:1402:1::d17d/128 Scope:Global
          inet6 addr: 2001:e30:1401:2:2d0:b7ff:fee3:d13e/64 Scope:Global
          inet6 addr: fe80::2d0:b7ff:fee3:d13e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:132290 errors:0 dropped:0 overruns:0 frame:0
          TX packets:593 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8785841 (8.3 MiB)  TX bytes:71396 (69.7 KiB)
................
................
```

Command Snapshort14: ifconfig

Summary

Dibbler and dhcpv6 available at sourceforge.net both have their limitation and drawback. The address leased by dibbler server cannot be pinged. In the word of dibbler developer

"It's rather difficult problem. DHCP's job is to obtain address and it exactly does that. To ping any other host, routing should be configured. And this should be done using Router Advertisements."

Along with this following enhancement is promised by the dibbler dhcpv6 project in near future.

crash/power outage recovery (CONFIRM/REBIND message)
Authentication option
Prefix Delegation option
RECONFIGURE support
temporary addresses support (IA_TA option)
FQDN option (DNS updates)
allow DB storing in real database (PostgreSQL probably)
failover - full redundancy on server side

While in the case of dhcpv6 of sourceforge.net its client is unable to run on only IPv6 network. According to protocol client can initiate the message transfer without IPv4 address. A mail from Shirley Ma IBM Linux Technology Center developer of dhcpv6 says that

More ever following bugs are still visible and yet to be resolved
No support for VLAN interface names
Not working on IPv6 only interfaces

bug in config.c
failed to compile netlink.c
FNAME-define not working properly

Still work is going on these two projects and the IPv6 product in FC3 is DEFINITELY still under development. It's not ready for prime time at all. As promises by the dhcpv6 project following enhancement is going to done in near future.
Name Service Search Option
Timezone Specifier Option
Simple Network Time Protocol Configuration Option

3.3.1 Multicasting

Multicast Addresses
An IPv6 multicast address is an identifier for a group of nodes. A node may belong to any number of multicast groups. Multicast addresses have the following format:

8	4	4	112 bits
11111111	Flags	Scope	Group ID

0	0	0	T

Figure 5 IPv6Multicast address; format of the flag field

11111111 at the start of the address identify the address as being a multicast address. ‘Flags’ is a set of 4 flags.
The high-order 3 flags are reserved, and must be initialized to 0.
T = 0 indicates a permanently-assigned ("well-known") multicast address, assigned by the global internet numbering authority.
T = 1 indicates a non-permanently-assigned ("transient") multicast address.
‘Scope’ is a 4-bit multicast scope value used to limit the scope of the multicast group.
Multicast addresses are split into scopes and types, which are as following.

Multicast scopes
Multicast scope is a parameter to specify the maximum distance a multicast packet can travel from the sending entity. Currently, the following regions (scopes) are defined:

- ffx1: node-local, packets never leave the node.
- ffx2: link-local, packets are never forwarded by routers, so they never leave the specified link.
- ffx5: site-local, packets never leave the site.
- ffx8: organization-local, packets never leave the organization (not so easy to implement, must be covered by routing protocol).
- ffxe: global scope.
- others are reserved

Multicast types

There are many types already defined/reserved. Some examples are:

- All Nodes Address: ID = 1h, addresses all hosts on the local node (ff01:0:0:0:0:0:0:1) or the connected link (ff02:0:0:0:0:0:0:1).
- All Routers Address: ID = 2h, addresses all routers on the local node (ff01:0:0:0:0:0:0:2), on the connected link (ff02:0:0:0:0:0:0:2), or on the local site (ff05:0:0:0:0:0:0:2)

Comparison between IPv6 multicasting and IPv4 multicasting

Although the basic notion of multicasting is common to IPv4 and IPv6, several new characteristics are introduced in IPv6 multicasting based on the results of IPv4 multicasting. For example, IPv6 explicitly limits the scope of a multicast address by using a fixed address field, whereas the scope was specified using TTL (Time to Live) of a multicast packet in IPv4.

In IPv4, however, multicasting was introduced as an extension of the basic specification; hence, IPv4 nodes do not necessarily support multicasting. On the other hand, specifications of IPv6 require that all IPv6 nodes support multicasting.

IPv4 multicasting use unicast addresses to identify a network interface. All IPv4 multicast routing daemons use the structure containing a member, which specifies an IPv4 address, which serves as an interface identifier. However, such an approach is not suitable for IPv6 for the following reasons. First, an IPv6-capable node may assign multiple addresses on a single interface, which tends to cause a configuration mismatch. Also, a link-local address is not necessarily unique within a node; consequently, it may not identify a single interface. A user must specify the interface index as well as the address in such a case. Since the specified index itself should identify a single interface, the address is actually redundant.

Scenario of Showing Multicasting

Here we show a scenario of multicasting, in which there is a multicast server streaming a stream in network in the multicast address ff1e::1.
In the first case there are two clients receiving the stream from the network. In the second case where a client wants to join the stream send a join ff1e::1 request message to the network. In the third case we see that all the three clients are receiving the stream from the network. In the fourth case when a client doesn’t want to further receive the stream from the network, it will give leave ff1e::1 request message to the network. In the fourth case we find only two clients receiving the stream and the third client is no longer receiving it.

```
Server (vlc)                  Network                      Clients (vlc)
Stream   --------------------->  ff1e::1      -------------------->  client n°1
                                             |------------------->   client n°2
```

```
Server (vlc)              Network                              Clients (vlc)
                                         <----------------------  client n°3
                                              join ff1e::1          (join)

Server (vlc)              Network                              Clients (vlc)
                                                |---------------> client n°1
  stream ----------------------> ff1e::1 -----------------------------> client n°2
                                                |---------------> client n°3

Server (vlc)              Network                              Clients (vlc)
                                         <----------------------  client n°1
                                            leave ff1e::1          (leave)

Server (vlc)              Network                              Clients (vlc)
 stream ----------------------> ff1e::1       ---------------------> client n°2
                                              --------------------> Client n°3
```

Implementation of Multicasting

For the implementation of multicasting in our project, we have made various network setups. We have connected many computers for this purpose. In multicasting, as we know there are different scopes which enables multicast packets to travel in the network.

So, firstly we have connected two computers using cross cable. Then we made one system as the multicast server and the other as the multicast client. And with this setup, we used link local scope which is ff12::1 for multicasting purpose. If we multicast using other scope like site local, organizational local or global scope, even then the client receives the packets sent by the multicast server.

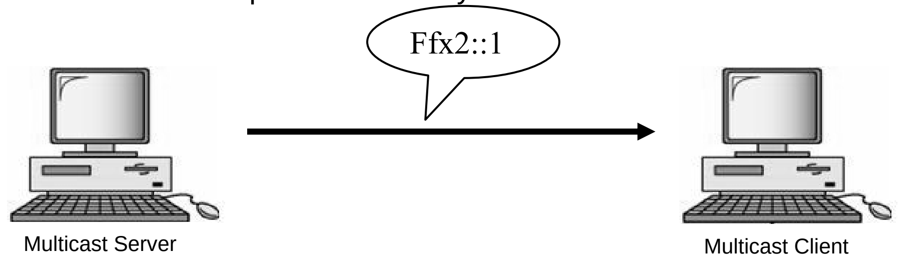


Figure 6 Cross cable Bi-node multicast setup

Then we connected five systems using a switch. In this kind of setup shows that all the systems are in the same LAN. In this setup we have a multicast server and three systems as multicast client. The one extra system in this setup shows that it is connected in this network but it does not act as a multicast client. To multicast packets in this setup we used link local address ff12::1. Other multicast scopes like site local, organizational local or global scope will also allow packets to be received by the clients here.

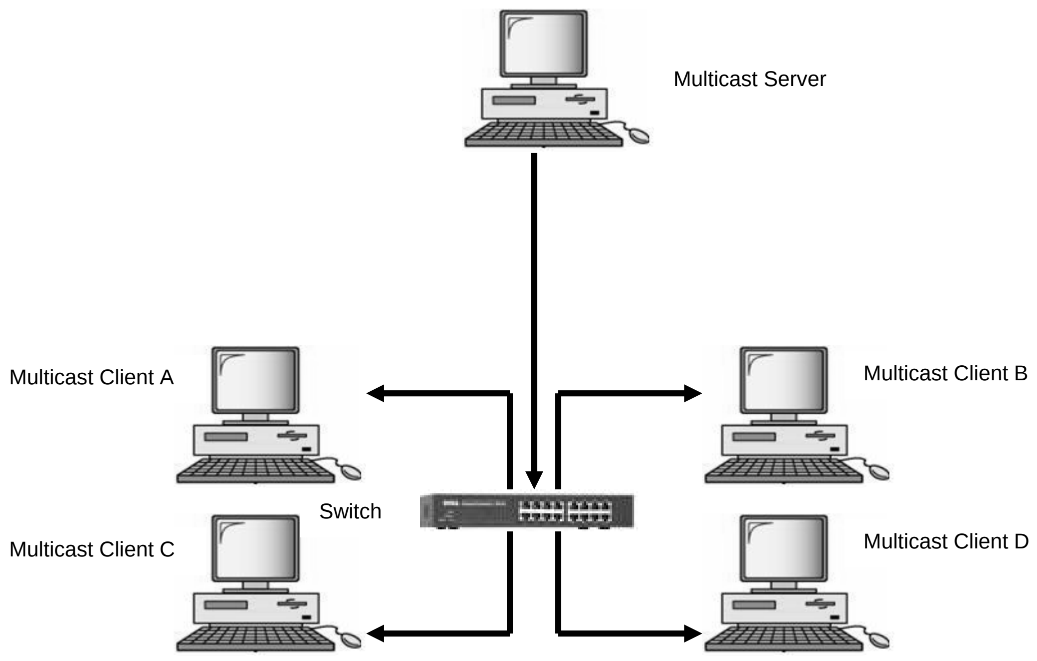


Figure 7Larger Multicast setup

Implementation of multicasting has been done using Video LAN Client (VLC) media player. We have tested it on fedora core 3 and windows XP.

Video LAN Client (VLC) media player
VLC is a cross platform media player that works on many platforms: Linux, Windows, Mac OS X, BeOS, *BSD, Solaris, Familiar Linux, Yopy/Linupy and QNX. It can play:

- MPEG-1, MPEG-2 and MPEG-4 / DivX files from a hard disk, a CD-ROM drive, ...
- DVDs, VCDs, and Audio CDs
- from satellite card (DVB-S),
- Several types of network stream: UDP Unicast, UDP Multicast (MPEG-TS), HTTP, RTP/RTSP, MMS, etc.
- From acquisition or encoding cards (on GNU/Linux and Windows only)

VLC can also be used as a streaming server to stream in unicast and multicast in IPv4 or Ipv6 on a high-bandwidth network.

Advantage of using multicasting in VLC
We encounter certain problems when we use unicasting or broadcasting in the network. With unicasting, when there are many clients who want to receive the stream, the network interface of the server becomes saturated. Therefore, the number of clients is very limited, especially when the stream is big. With broadcasting, the machines that do not want to receive the stream are polluted

and there are certain devices that do not like to receive huge broadcasts. If we send several streams at the same time, the network becomes oversaturated.

With multicast, the packets are sent on the network to a multicast IP group, which is designated by its IP address. The machines can join or leave a multicast group by sending a request to the network. The request is usually sent by the kernel of the operating system. The VLC takes care of asking the kernel of the operating system to send the join request. It is possible for one client to belong to several groups.

Implementation VLC Media Player

Installing and Running VLC media player

Fedora Core 3

We downloaded VLC Binaries packages vlc-binary.tar.gz for Fedora Core 3 which is the latest RPM x86 packages tarballs and the tarballs listed in the section below and uncompress them in the same directory:

```
 $ tar zvxf vlc-binary.tar.gz
```

Then, as root, install the packages:

```
   # rpm -U vlc/* --force --nodeps
```

Windows XP

Windows contain the self-extracting package vlc-0.8.1-win32.exe**.** We downloaded this package and installed it.

Running VLC media player

For server: For multicasting, we select UDP as the stream output with the following settings:

address: ff1e and port : 1234

For client: we receive – the UDP/RTP multicast stream with again the same settings:

address: ff1e and port : 1234

Summary

The demonstration of multicasting in IPv6 has been done using VLC Media player. The purpose of using VLC media player was that this was the only player available till now that supports multicasting in IPv6 and it is a freeware. And this player serves both as the server and the client. This media player has support in multiple platforms. So, when we send a stream to network in one platform we can receive the stream in another platform. We have done the testing with Fedora Core 3 and Windows XP.

We encountered difficulty in installing vlc-0.8.1 on Fedora. There are certain library files that VLC media player doesn't support on Fedora Core 3. So, we have installed it with nodeps flags. When we start VLC, there is the following error message of libhal.so.0.

3.3.3 DNS Server

Basic Terminology and Commands
DNS -A Hierarchical, distributed database is called domain name system.

Domain Name – Data stored in DNS is identified by domain name.

Domain – Each node of the tree of DNS is called a domain and is given a level.

Name server - A computer running a program that converts domain name into appropriate IP addresses and vice versa. Name servers are the backbone of internet system.

Resolver Library – Client uses resolver library to look up information in the DNS. It sends queries to one or more name servers and interprets the responses

Zone – The name space is partitioned into areas called zones. The data for each zone is stored in name server which answers queries about the zone using DNS protocol.

Caching server - A server that remember only the domain that has already been accessed. It cannot provide information to outside source. It only speed up search since domain information is already stored in memory and the server knows where to go rather than having to send out request for domain information.

Resource Record type - A record type is defined to store a host's IPv6 address. A host that has more than one IPv6 address must have more than one such record.

AAAA record type - The AAAA resource record type is a record specific to the Internet class that stores a single IPv6 address. The IANA assigned value of the type is 28 (decimal).

AAAA data format - A 128 bit IPv6 address is encoded in the data portion of an AAAA resource record in network byte order (high-order byte first).

AAAA query - An AAAA query for a specified domain name in the Internet class returns all associated AAAA resource records in the answer section of a response.
A6 record type -The A6 record type is specific to the IN (Internet) class and has type number 38 (decimal)
The RDATA portion of the A6 record contains two or three fields.

Prefix length -1 octet
Address suffix – 0...16 octets
Prefix name - 0.....255 octets

IPv6 addresses: AAAA RR vs A6 RR

Working group consensus as perceived by the chairs of the DNSEXT and NGTRANS working groups is that:

- AAAA records are preferable at the moment for production deployment of IPv6, and
- That A6 records have interesting properties that need to be better understood before deployment.
- It is not known if the benefits of A6 outweigh the costs and risks.

IP6.ARPA Domain - A special domain is defined to look up a record given an IPv6 address. The intent of this domain is to provide a way of mapping an IPv6 address to a host name, although it may be used for other purposes as well. The domain is rooted at IP6.ARPA.
An IPv6 address is represented as a name in the IP6.ARPA domain by a sequence of nibbles separated by dots with the suffix ".IP6.ARPA". The sequence of nibbles is encoded in reverse order, i.e., the low-order nibble is encoded first, followed by the next low-order nibble and so on. Each nibble is represented by a hexadecimal digit. For example, the reverse lookup domain name corresponding to the address
4321:0:1:2:3:4:567:89ab would be
b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.0.0.0.0.1.2.3.4.IP6.ARPA.

IP6.INT Domain - A special domain is defined to look up a record given an address. The intent of this domain is to provide a way of mapping an IPv6 address to a host name, although it may be used for other purposes as well. The domain is rooted at IP6.INT.
An IPv6 address is represented as a name in the IP6.INT domain by a sequence of nibbles separated by dots with the suffix ".IP6.INT". The sequence of nibbles is encoded in reverse order, i.e. the low-order nibble is encoded first, followed by the next low-order nibble and so on. Each nibble is represented by a hexadecimal digit. For example, the inverse lookup domain name corresponding to the address
4321:0:1:2:3:4:567:89ab would be
b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.0.0.0.0.1.2.3.4.IP6.INT.

BIND - BIND (Berkeley Internet Name Domain, previously: Berkeley Internet Name Daemon) is a project, in which a group maintains the DNS-related software suite that runs under Linux. The most well known program in BIND is named, the daemon that responds to DNS queries from remote machines.

DNS Clients - A DNS client doesn't store DNS information; it must always refer to a DNS server to get it. The only DNS configuration file for a DNS client is the /etc/resolv.conf file, which defines the IP address of the DNS server it should use. No other files need to be configured.

Authoritative DNS Servers - Authoritative servers provide the definitive information for our DNS domain, such as the names of servers and Web sites in it. They are the last word in information related to our domain.

Method used by DNS Servers to find out site information
There are 13 root authoritative DNS servers that all DNS servers query first. These root servers know all the authoritative DNS servers for all the main domains -.com, .net, and the rest. This layer of servers keeps track of all the DNS servers that Web site systems administrators have assigned for their sub domains.

DNS Caching Name Server
Most servers don't ask authoritative servers for DNS directly, they usually ask a caching DNS server to do it on their behalf. The caching DNS servers then store, the most frequently requested information to reduce the lookup overhead of subsequent queries.
If we want to advertise our Web site www.mywebsite.com to the rest of the world, then a regular DNS server is what we require. Setting up a caching DNS server is fairly straightforward and works whether or not our ISP provides us with a static or dynamic Internet IP address.
After we set up our caching DNS server, we must configure each of our home network PCs to use it as their DNS server. If our home PCs gets their IP addresses using DHCP, then we have to configure our DHCP server to make it aware of the IP address of our new DNS server, so that the DHCP server can advertise the DNS server to its PC clients. Off-the-shelf router/firewall appliances used in most home networks usually can act as both the caching DNS and DHCP server, rendering a separate DNS server is unnecessary.

Static DNS Server
If our ISP provides us with a fixed or static IP address, and we want to host our own Web site, then a regular authoritative DNS server would be the way to go. A caching DNS name server is used as a reference only; regular name servers are used as the authoritative source of information for our Web site's domain. Regular name servers are also caching name servers by default.

Dynamic DNS Server
If our ISP provides our router/firewall with its Internet IP address using DHCP then we must consider dynamic DNS. Dynamic DNS is a system for allowing an Internet domain name to be assigned to a varying IP address. This makes it possible for other sites on the Internet to establish connections to the machine without needing to track the IP address themselves.

Method of getting our own Domain
Whether or not we use static or dynamic DNS, we need to register a domain.

Dynamic DNS providers frequently offer us a subdomain of their own site, such as mywebsite.dnsprovider.com, in which we register our domain on their site.

If we choose to create our very own domain, such as mywebsite.com, we have to register with a company specializing in static DNS registration and then point our registration record to the intended authoritative DNS for our domain

If we want to use a dynamic DNS provider for our own domain, then we have to point our registration record to the DNS servers of our dynamic DNS provider.

Basic DNS Testing of DNS Resolution

As we know, DNS resolution maps a fully qualified domain name (FQDN), such as www.linuxhomenetwrking.com, to an IP address. This is also known as a forward lookup. The reverse is also true: By performing a reverse lookup, DNS can determining the fully qualified domain name associated with an IP address.

There are a number of commands we can use do these lookups. Linux uses the host command, for example, but Windows uses nslookup.

The host command

The host command accepts arguments that are either the fully qualified domain name or the IP address of the server when providing results. To perform a forward and reverse lookup, syntax used is [9]

```
[root@bigboy tmp]# host www.linuxhomenetworking.com
 www.linuxhomenetworking.com has address 65.115.71.34
[root@bigboy tmp]# host 65.115.71.34
34.71.115.65.in-addr.arpa domain name pointer 65-115-71-   34.myisp.net.
```

Command Snapshot 15: Forward and Reverse DNS Lookup (host)

As we see that the forward and reverse entries don't match. The reverse entry matches the entry of the ISP.

The nslookup command

The nslookup command provides the same results on Windows PCs. To perform forward and reverse lookup, use [10]

```
C:\> nslookup www.linuxhomenetworking.com
        Server:  192-168-1-200.my-site.com
        Address:  192.168.1.200

        Non-authoritative answer:
        Name:    www.linuxhomenetworking.com
      Address:  65.115.71.34
C:\> nslookup 65.115.71.34
        Server:  192-168-1-200.my-site.com
        Address:  192.168.1.200


        Name:    65-115-71-34.my-isp.com
      Address:  65.115.71.34
```

Command Snapshot 16: Forward and Reverse DNS Lookup (nslookup)

Implementing the DNS

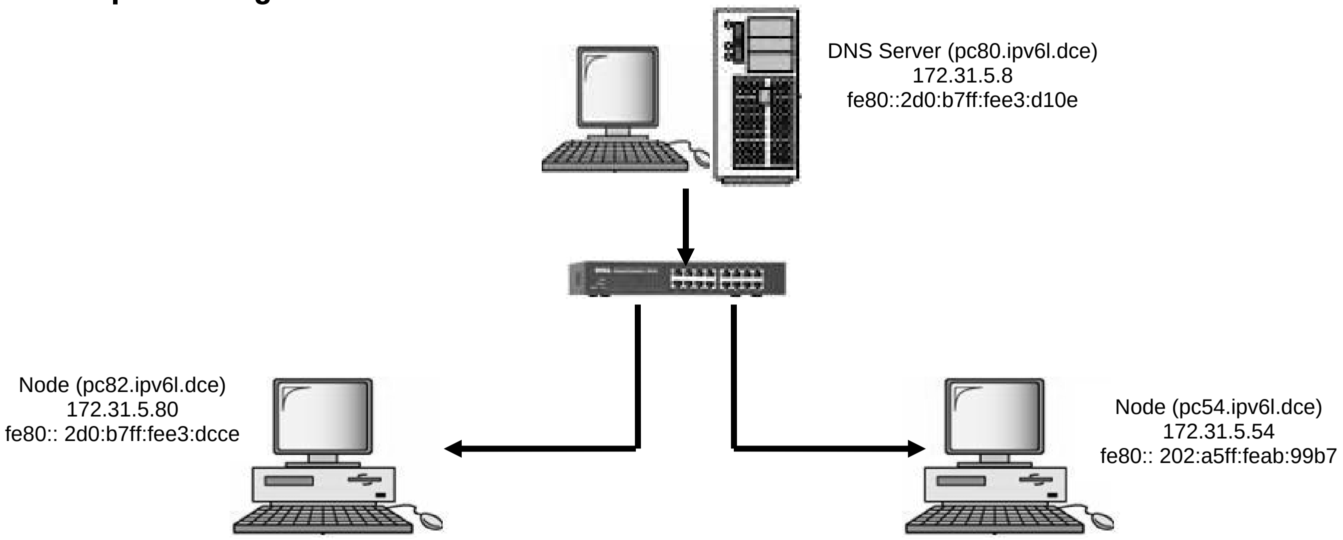


Figure 8 DNS Client and Server Setup

DNS server pc08	Link-local address ipv6	fe80::2d0:b7ff:fee3:d10e
	MAC	00:02:b7:e3:d1:0e
pc82	Link-local Address ipv6	fe80:: 2d0:b7ff:fee3:dcce
	MAC	00:02:b7:e3:dc:ce
pc54 DNS server pc08	Link-local Address Ipv6	fe80::202:a5ff:feab:99b7
	Link-local address ipv6	fe80::2d0:b7ff:fee3:d10e

Table 1 IPv6 Address assigned to the hosts

BIND Configuration file The following files are configured for the configuration of DNS server:

- /var/named/chroot /etc/named.conf
- /var/named/chroot/var/named/2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ipv6.arpa
- /var/named/chroot/var/named/localdomain.zone
- /var/named/chroot/var/named/named.ca
- /var/named/chroot/var/named/named.zero
- /var/named/chroot/var/localhost.zone
- /var/named/chroot/var/named/named.ipv6.local
- /var/named/chroot/var/named/ipv6l.dce
- /var/named/chroot/var/named/named.local
- x20010e3014010002-64

The above configuration files are as follows:

/var/named/chroot/etc/named.conf:

```
options {
        directory "/var/named";
        dump-file "/var/named/data/cache_dump.db";
        statistics-file "/var/named/data/named_stats.txt";
        listen-on-v6 { any ; };
        };

zone "." IN {
        type hint;
        file "named.ca";
};

zone "localdomain" IN {
        type master;
        file "localdomain.zone";
        allow-update { none; };
};

zone "localhost" IN {
        type master;
        file "localhost.zone";
        allow-update { none; };
};

zone "0.0.127.in-addr.arpa" IN {
        type master;
        file "named.local";
        allow-update { none; };
};

zone
"0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.arpa"
IN {
        type master;
        file "named.ip6.local";
        allow-update { none; };
};

zone "255.in-addr.arpa" IN {
        type master;
        file "named.broadcast";
        allow-update { none; };
};

zone "0.in-addr.arpa" IN {
        type master;
        file "named.zero";
        allow-update { none; };
};
zone "ipv6l.dce" {
                type master;
                  file "ipv6l.dce";
};

zone "2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ip6.arpa" {
            type master;
            file "2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ip6.arpa";

};
include "/etc/rndc.key";
```

File Snapshot 17/var/named/chroot/etc/named.conf

/var/named/chroot/var/named/2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ipv6.arpa :

```
;
; 2001:0e30:1402:1::/64
;
; Zone file built with the fpsn.net IPv6 Reverse DNS zone builder
; http://tools.fpsn.net/ipv6-inaddr
;
$TTL 3d ; Default TTL (bind 8 needs this, bind 9 ignores it)
@       IN SOA 2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ip6.arpa. root.ipv6l.dce.
(
                200605190       ; Serial number (YYYYMMdd)
                24h             ; Refresh time
                30m             ; Retry time
                2d              ; Expire time
                3d              ; Default TTL (bind 8 ignores this,
                                  bind 9 needs it)
)

                                ; Name server entries
                                IN     NS     pc08.ipv6.dec.
                                IN     NS     pc08.ipv6.dce.
; IPv6 PTR entries

; Subnet #1
$ORIGIN 2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ip6.arpa.

e.0.1.d.3.e.e.f.f.f.7.b.0.d.2.0         IN      PTR     pc08.ipv6l.dce.
8.c.9.9.b.a.e.f.f.f.5.a.2.0.2.0         IN      PTR     pc82.ipv6l.dce.
6.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0         IN      PTR     pc54.ipv6l.dce.;
```

FileSnapshort18: /var/named/chroot/var/named/2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ipv6.arpa

/var/named/chroot/var/named/localdomain.zone :

```
$TTL    86400
@         IN SOA  localhost root (
                    42            ; serial (d. adams)
                    3H            ; refresh
                    15M           ; retry
                    1W            ; expiry
                    1D )          ; minimum
           IN NS         localhost
localhost      IN A          127.0.0.1
```

FileSnapshort19: /var/named/chroot/var/named/localdomain.zone

/var/named/chroot/var/named/named.ca :

```
;       This file is made available by InterNIC
;       under anonymous FTP as
;           file                /domain/named.cache
;           on server           FTP.INTERNIC.NET
;       -OR-                    RS.INTERNIC.NET
;
;       last update:    Jan 29, 2004
;       related version of root zone:   2004012900
;
;
; formerly NS.INTERNIC.NET
;
.                        3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.      3600000      A     198.41.0.4
;
; formerly NS1.ISI.EDU
;
.                        3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.      3600000      A     192.228.79.201
;
; formerly C.PSI.NET
...........
...........
```

FileSnapshort20: /var/named/chroot/var/named/named.ca
To view the full output of the command refer the Appendix

/var/named/chroot/var/named/named.zero :

```
$TTL    86400
@          IN SOA  localhost       root (
                      42              ; serial (d. adams)
                      3H              ; refresh
                      15M             ; retry
                      1W              ; expiry
                      1D )            ; minimum
        IN      NS      localhost
```

FileSnapshort21: /var/named/chroot/var/named/named.zero

/var/named/chroot/var/localhost.zone :

```
$TTL    86400
@             IN SOA  @       root (
                      42              ; serial (d. adams)
                      3H              ; refresh
                      15M             ; retry
                      1W              ; expiry
                      1D )            ; minimum

                IN NS           @
                IN A            127.0.0.1
                IN AAAA         ::1
```

FileSnapshort22: /var/named/chroot/var/localhost.zone

/var/named/chroot/var/named/named.ipv6.local :

```
$TTL    86400
@       IN      SOA     localhost. root.localhost.  (
                          1997022700 ; Serial
                          28800      ; Refresh
                          14400      ; Retry
                          3600000    ; Expire
                          86400 )    ; Minimum
              IN      NS      localhost.

1      IN      PTR     localhost.
```

FileSnapshort23: /var/named/chroot/var/named/named.ipv6.local

/var/named/chroot/var/named/ipv6l.dce :

```
;; File:ipv6l.dce
; IPv6 iitk kanpur
; IP v6 test network
;
$TTL    86400
@               IN      SOA     ipv6l.dce.     root.ipv6l.dce. (
                                 20060516;serial
                                 3H      ; refresh
                                 15M     ; retry
                                 1W      ; expiry
                                 1D )    ; minimum
                IN      NS              pc08.ipv6l.dce.
;;
;;
$ORIGIN ipv6l.dce.
; Local hosts
; ------------------
pc08  IN      A       172.31.5.80
pc08  IN      AAAA    2001:0e30:1401:2:2d0:b7ff:fee3:d10e
pc82  IN      A       172.31.5.82
pc82  IN      AAAA    2001:0e30:1401:2:202:a5ff:feab:99c8
pc54  IN      A       172.31.5.54
pc54  IN      AAAA    2001:0e30:1401:2::202:a5ff:feab:99b7
```

FileSnapshort24: /var/named/chroot/var/named/ipv6l.dce

/var/named/chroot/var/named/named.local :

```
$TTL    86400
@       IN      SOA     localhost. root.localhost.  (
                            1997022700 ; Serial
                            28800      ; Refresh
                            14400      ; Retry
                            3600000    ; Expire
                            86400 )    ; Minimum
              IN      NS      localhost.

1       IN      PTR     localhost.
```

FileSnapshort25: /var/named/chroot/var/named/named.local :

x20010e3014010002-64 :

```
;file x20010e301402001
;;
;; x20010e3014020001-64.ip6.arpa
;;
$TTL            86400
;$ORIGIN \[x20010e3014010002/64].ip6.arpa.
@               IN      SOA      \[x20010e3014010002/64].ip6.arpa.
root.ipv6l.dce. (
                        2002021602  ; Serial - YYYYMMDDXX
                        10800       ; Refresh
                        3600        ; Retry
                        3600000     ; Expire
                        86400 )     ; Minimum
;;
;; Nameservers
;;
        IN      NS      pc08.ipv6l.dce.


;;
;; Hosts on the ethernet 2001:0e30:1402:0001::/64
;;
$ORIGIN \[x20010e3014010002/64].ip6.arpa.
\[x02d0b7fffee3d10e]            IN      PTR     pc08.ipv6l.dce.
\[x0202a5fffeab99c8]            IN      PTR     pc82.ipv6l.dce.
\[x0000000000000008]            IN      PTR     pc54.ipv6l.dce.
```

FileSnapshort26: x20010e3014010002-64

Testing of DNS server

We check the working of the DNS server by checking the output of the forward zone file by the command ‘dig’. The underlined lines in Command Snapshort25 shows that the DNS server setup above is been referred. The output of the reverse zone file by the command ‘dig –x’. The underlined lines in Command Snapshort26 and the DNS server setup above is been referred. The output of the system that we have set up as the DNS server is shown in Command Snapshort27 [9].

```
[root@pc08 ~]# dig pc08.ipv6l.dce
; <<>> DiG 9.2.4 <<>> pc08.ipv6l.dce
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62342
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;pc08.ipv6l.dce.             IN      A

;; ANSWER SECTION:
pc08.ipv6l.dce.      86400   IN      A       172.31.5.80

;; AUTHORITY SECTION:
ipv6l.dce.             86400   IN      NS      pc08.ipv6l.dce.

;; ADDITIONAL SECTION:
pc08.ipv6l.dce.      86400   IN      AAAA
2001:e30:1401:2:2d0:b7ff:fee3:d1 0e

;; Query time: 2 msec
;; SERVER:
2001:e30:1401:2:2d0:b7ff:fee3:d10e#53(2001:e30:1401:2:2d0:b7ff:fee3:d
10e)
;; WHEN: Wed Jun 15 18:39:56 2006
;; MSG SIZE  rcvd: 93
```

CommandSnapshort27 dig pc08.ipv6l.dce

```
[root@pc08 ~]# dig -x 2001:e30:1401:2:2d0:b7ff:fee3:d10e

; <<>> DiG 9.2.4 <<>> -x 2001:e30:1401:2:2d0:b7ff:fee3:d10e
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44503
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
;; QUESTION SECTION:
;e.0.1.d.3.e.e.f.f.f.7.b.0.d.2.0.2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ip6.arp
a.
IN PTR
;; ANSWER SECTION:
e.0.1.d.3.e.e.f.f.f.7.b.0.d.2.0.2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ip6.arpa
.
259200 IN PTR pc08.ipv6l.dce.
;; AUTHORITY SECTION:
2.0.0.0.1.0.4.1.0.3.e.0.1.0.0.2.ip6.arpa. 259200 IN NS pc08.ipv6l.dce.
;; ADDITIONAL SECTION:
pc08.ipv6l.dce.      86400   IN      A       172.31.5.80
pc08.ipv6l.dce.      86400   IN      AAAA
2001:e30:1401:2:2d0:b7ff:fee3:d10e
;; Query time: 2 msec
;; SERVER:
2001:e30:1401:2:2d0:b7ff:fee3:d10e#53(2001:e30:1401:2:2d0:b7ff:fee3:d10e
)
;; WHEN: Wed Jun 15 18:41:45 2006
;; MSG SIZE  rcvd: 179
```

CommandSnapshort28 dig -x 2001:e30:1401:2:2d0:b7ff:fee3:d10e

```
[root@pc08 ~]# dig localhost

; <<>> DiG 9.2.4 <<>> localhost
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28474
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;; QUESTION SECTION:
;localhost.                     IN      A
;; ANSWER SECTION:
localhost.              86400   IN      A       127.0.0.1
;; AUTHORITY SECTION:
localhost.              86400   IN      NS      localhost.
;; ADDITIONAL SECTION:
localhost.              86400   IN      AAAA    ::1
;; Query time: 2 msec
;; SERVER:
2001:e30:1401:2:2d0:b7ff:fee3:d10e#53(2001:e30:1401:2:2d0:b7ff:fee3:d10e
)
;; WHEN: Wed Jun 15 18:43:48 2006
;; MSG SIZE  rcvd: 85
```

CommandSnapshort29 dig localhost

Summary

The two future aspects of DNS in near future is implementation of some advance concepts like Dynamic update of DNS using allow update Incremental zone transfer (IXFR), and split DNS. One common reason for setting up a DNS system this way is to hide "internal" DNS information from "external" clients on the Internet. It allow internal networks that are behind filters or in RFC 1918 space (reserved IP space, as documented in RFC 1918) to resolve DNS on the Internet.

Other aspect include Host requests DNS configuration using DHCPv6 where address is leased by dhcpv6 server. Since yet work on dhcpv6 server is going on hence this module is also a matter of research.

Setting up a DNSSEC secure zone is also come in next version of Bind ,BIND 9 ships with several tools that are used in this process, which are explained in more detail below. In all cases, the "-h" option prints a full list of parameters. DNSSEC tools require the keyset and signedkey files to be in the working directory, and that the tools shipped with BIND 9.0.x are not fully compatible with the current ones.

4 Connecting to rest of the world

Unfortunately, IPv4 and IPv6 are not directly compatible, so programs and systems designed to one standard can not communicate with those designed to the other.

Various methods

Dual Stack – Since it is cost prohibitive to replace all existing IPv4 devices in the Internet Infrastructure with IPv6 devices, most devices will be only IPv4. New devices must support both IPv4 and IPv6. Since there will be two stacks running, performance benchmarking needs to be conducted with different levels of both types of traffic.

Tunneling – Initially, IPv6 traffic will have to cross an IPv4 infrastructure. The impact of having to encapsulate or "tunnel" IPv6 traffic inside IPv4 packets (6over4) is to be measured. As time passes and more IPv6 device are deployed, IPv4 traffic will be tunneled in IPv6 packets (4over6).

Translation – Servers and end devices today only support IPv4, meaning most IPv6 traffic will have to be converted into IPv4 (6to4). These devices will then have to convert the IPv4 traffic back to IPv6 to send the response back to the original host (4to6). The ability of these devices to do this at high rates will help in the deployment of IPv6.

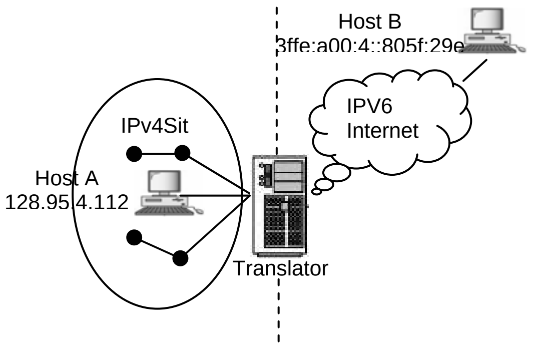
As observed and decided in academic environment where applications are largely sporadic and contain direct interaction with the network, Tunneling seems a difficult. More over as the steps of the migration stated in the beginning show the rest of the network may not be ready for a new protocol implementation which is required by both Dual Stack and Tunneling. The most appropriate way for the IPv6 network to communicate with the rest of institution is Translation as it requires only the intermediate gateway to be configured; the two separate networks can remain unmodified.

4.1 Translation

We have designed and implemented a transparent transition service that translates packet headers as they cross between IPv4 and IPv6 networks.
We believe that an IPv6/IPv4 network address and protocol translator is complementary to other transition strategies from IPv4 to IPv6 (e.g., dual-stack IPv6/IPv4 hosts, Assignment of IPv4 addresses to IPv6 Hosts). In particular, we believe that IPv6/IPv4 translation will be a valuable tool to developers porting applications from IPv4 to IPv6. For instance, a server application ported to IPv6 can be tested without having to port the client as well [3].

4.1.1Network Address and Protocol Translation

The address and protocol translation presented in this section enables both the communication between nodes in an IPv4 site with nodes in the IPv6 network, and between nodes in an IPv6 site with nodes in an IPv4 node. Figures 4 and 5 illustrate these scenarios, and the following paragraphs describe them in more detail.



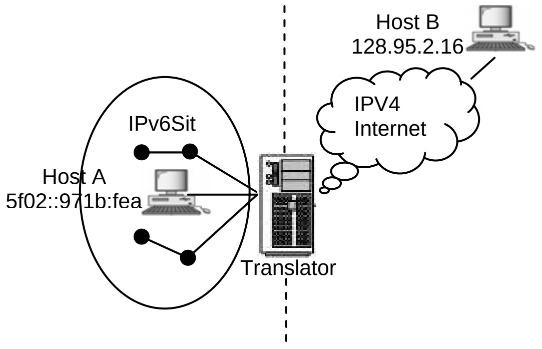


Figure 9 Translator for Ipv6 site

Figure10 Translator for Ipv4 site

Figure 9 illustrates a translator for an IPv6 site communicating with nodes in an IPv4 network. The internal routing of the IPv6 site must be configured such that packets intended for IPv4 nodes route to the translator. Hosts in the IPv6 site send packets to nodes in the IPv4 network using IPv6 addresses that map to individual IPv4 hosts. For this scenario, a design presented in proposes that IPv6 nodes use an *IPv4-compatible IPv6* address as their own address and an *IPv4-mapped IPv6* address when communicating with IPv4-only nodes. An IPv4-compatible IPv6 address holds an IPv4 address in the low-order 32-bits, with a unique high-order 96-bit prefix of 0:0:0:0:0:0 (all zero bits), and always identifies an IPv6/IPv4 or IPv6- only node; they never identify an IPv4-only node. Similarly, an IPv4-mapped IPv6 address identifies an IPv4-only node and its high-order 96-bits bear the prefix 0:0:0:0:0:FFFF. The address of any IPv4- only node may be mapped into the IPv6 address space by prefixing 0:0:0:0:0:FFFF to its IPv4 address. The benefit of this approach is that the translator can be stateless. However, regardless of the 96-bit IPv6 prefix that is used to map between the IPv4 and IPv6 address domains it still remains necessary to identify a host in the IPv6 site with a unique IPv4 address. That is, in Figure 9, for Host B to communicate with Host A Requires an IPv4 address that can be routed through the IPv4 Internet. To overcome this limitation a stateful translator could multiplex several IPv6 hosts onto a single, globally unique IPv4 address using the TCP/UDP port translation technique.

Figure 10 illustrates a translator for an IPv4 site communicating with nodes in an IPv6 network. Hosts in the IPv4 site send packets to nodes in the IPv6 network using IPv4 destination addresses assigned by the translator that map to individual IPv6 hosts. For this to work, the internal routing of the IPv4 site must contain routes to the translator for packets with the destination field using one of these IPv4 addresses. The translator, upon receiving such packets, will do the IPv4-to-IPv6 translation and forward the packet to the IPv6 network. In contrast to the above scenario, the translator can use unique IPv6 addresses to refer to nodes in the IPv4 site in order to do IPv6-to-IPv4 translation for packets it receives from the IPv6 network. These IPv6 addresses may come from a pool that is dynamically assigned to the set of IPv4 hosts communicating with IPv6 hosts. A better approach is to assign unique and routable IPv6 addresses to all nodes in the IPv4 site and to register them with DNS. This should be easily possible given that the IPv6 address space is sufficiently large, and also has the benefit that arbitrary hosts in the IPv6 Internet can easily lookup and initiate sessions with nodes in the IPv4 site via the translator.

In summary, the subtle difference between these two scenarios is that the former involves mapping a pool of *global* IPv4 addresses referring to IPv6 addresses, whereas the latter can leverage site *private* IPv4 addresses to refer to IPv6 addresses. Global IPv4 addresses will be scarce and mechanisms are required to dynamically assign a pool of these IPv4 addresses on a temporary basis to IPv6 nodes so that they can communicate with IPv4 nodes. On the other hand, there is a large pool of roughly 17 million site private IPv4 addresses, which can be used by the translator to map to IPv6 addresses. Our translator is designed to support all of the scenarios just described. To enable communication between an IPv4 and IPv6 node, a translator needs to do both address and protocol translation. Protocol translation involves mapping most of the fields illustrated in Figure 6 from one version of IP to the other. Address translation involves converting addresses for packets crossing the protocol boundary.

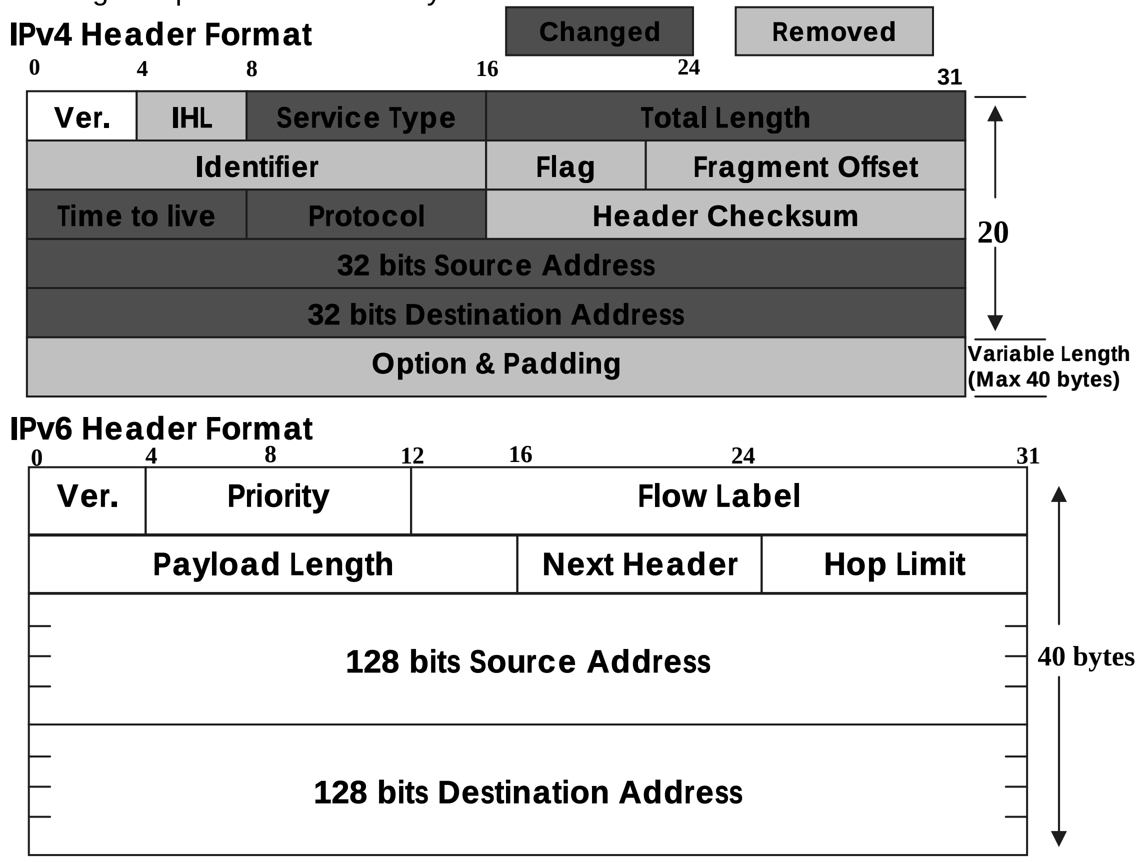


Figure11 Pv4 and IPv6 Header Format

Address Translation

Address translation is trivial when using IPv4-mapped and IPv4-compatible IPv6 addresses. For the IPv6-to- IPv4 direction the translator simply extracts the lower 32-bits of an IPv6 address to obtain an IPv4 address. For the opposite direction the translator sets the lower 32-bits of the IPv6 source/destination addresses to the IPv4 source/destination addresses, and sets the upper 96-bits of the IPv4 source and destination addresses to the IPv4-mapped and IPv4-compatible prefix, respectively. However, it is considered to be a very bad idea to use IPv4-mapped address as it has the drawback of requiring IPv6 routers to contain routes to IPv4- mapped addresses. The alternative is to use IPv6- only addresses to refer to IPv4 nodes, which requires the translator to maintain an explicit mapping between IPv4 and IPv6 addresses. For clarity, we introduce an IPxNODEy notation to disambiguate among the types of addresses used in the translation process.

IpxNODEy	**Definition**
IP4NODE4	V4 address of a V4 node
IP6NODE6	V6 address of a v6 node
IP6NODE4	V6 address of a v4 node
IP4NODE6	V4 address of a v6 node

Table2. IP address definition

Table 2 defines the four types of addresses in terms of this notation. The first two rows define the addresses that are native to the IPv4 and IPv6 nodes. The last two rows define address aliases, which addresses used by translation process are assigned by the translator, used to translate between the IPv4 and IPv6 address domains. As an example of using this IPxNODEy notation consider the following scenario: an arbitrary IPv6-only host wishes to communicate with our IPv4-only web server via the translator. For an IPv6 host to communicate with our IPv4 web server requires an IPv6 address that is an alias (IP6NODE4) address for the web server's native IPv4 host (IP4NODE4) address. Similarly, for the web server to reply to the IPv6 host requires an IPv4 address that is an alias (IP4NODE6) address for the IPv6 host's native (IP6NODE6) address. That is, the translator maps the IP6NODE4 address to the IP4NODE4 address of the web server, and the IP4NODE6 address to the IP6NODE6 address of the IPv6 host. The translation of addresses has three phases: address binding, address lookup and translation, and address unbinding, which we describe in the following subsections.

Address Binding

Address binding is the phase where an IPv4 address is associated with an IPv6 address and vice versa.

Key-to-Value	**Definition**
IP6NODE4-to-IP4NODE4	v6 address mapped to v4 address
IP4NODE6-to-IP6NODE6	v4 address mapped to v6 address

Table 3 Mapping between IPv4 and Ipv6 address used by the translation process

The translator maintains key-to-value tuples, listed in Table 3, to map between IPv4 and IPv6 addresses. For addresses that are statically mapped, the binding happens when the translator is initialized. If the translator is configured to use IPv4 mapped/compatible IPv6 addresses then all the bindings are implicitly static as they are defined by these special IPv6 addresses. Other static mappings could be setup between arbitrary IPv4 and IPv6 addresses. For example, the binding of addresses for an IPv4 node to an IPv6 node could be done statically by a network manager when assigning IPv6 addresses to existing nodes in the IPv4 site. That is, IP6NODE4-to-IP4NODE4 is the static mappings of IPv6 addresses assigned to IPv4 hosts. Otherwise, the binding between addresses needs to happen dynamically. IPv6 addresses are larger than IPv4 addresses and it is not possible to create a one-to-one IP4NODE6-to-IP6NODE6 binding. Consequently, it will be necessary to reuse IP4NODE6 addresses to bind them to other IP6NODE6 addresses.

Address Lookup and Translation

Once a binding is established it can be used for address lookup and translation.

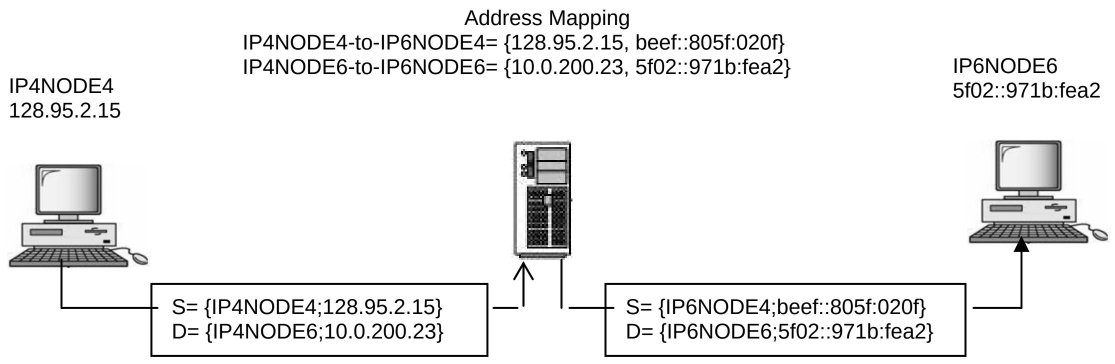

Address Mapping
IP4NODE4-to-IP6NODE4= {128.95.2.15, beef::805f:020f}
IP4NODE6-to-IP6NODE6= {10.0.200.23, 5f02::971b:fea2}
IP4NODE4
128.95.2.15
IP6NODE6
5f02::971b:fea2
S= {IP4NODE4;128.95.2.15}
D= {IP4NODE6;10.0.200.23}
S= {IP6NODE4;beef::805f:020f}
D= {IP6NODE6;5f02::971b:fea2}

Figure 12 Basic Address Translation Operations

The example in Figure 12 illustrates the translation using the IPxNODEy notation defined earlier. When the IPv4 node sends a packet to the IPv6 node it is routed through the translator. The translator receives the packet, translates the 128.95.2.15 to beef::805f:020f source address using the IP4NODE4- to-IP6NODE4 mapping, and translates the 10.95.2.23 to 5f02::971b:fea2 destination address using the IP4NODE6-to-IP6NODE6 mapping. Likewise, IP packets on the return path go through a reverse address translation. Notice that this requires no changes to hosts or routers. As far as the IPv4 host is concerned, IP4NODE6=10.0.200.23 is the address used by the IPv6 hosts. Conversely, the IPv6 host believes that IP6NODE4= beef::805f:020f is the address used by the IPv4 hosts. The address translation is transparent to both hosts.

Address Unbinding

Address unbinding is the phase when the association between an IPv4 and IPv6 address is broken. We expect the number of bindings of the IP6NODE4-to- IP4NODE4 mapping to remain fairly constant during the day-by-day operation of the translator; new bindings are only necessary when adding new hosts to the site. On the other hand, the numbers of bindings of the IP4NODE6-to-IP6NODE6 mapping are more dynamic and depend on the number of connections established to different hosts in the network. The number of reserved IP4NODE6 addresses used by the translator limits the number of bindings possible for the IP4NODE6-to- IP6NODE6 mappings. For the scenario where the translator is providing service for an IPv6 site (as illustrated in Figure 9), the IP4NODE6 addresses are a small number of unique IPv4 addresses. It is crucial for the translator to detect when an IP4NODE6 address can be reused in order to create new bindings; otherwise, new sessions may be refused if there are no IP4NODE6 addresses available. For the scenario where a translator is providing service to an IPv4 site (as illustrated in Figure 10), the IP4NODE6 addresses may come from a relatively large pool of private network addresses (as mentioned earlier, there are roughly 17 million of such addresses available). Here the concern is to safely remove unused bindings to ensure that the mapping table does not require too much memory and that address lookup performance does not deteriorate. Removing a binding too early should never occur, as it would effectively terminate any ongoing communication that relied on the binding.

Protocol Translation
Protocol translation consists of a simple mapping between the two IP protocols, with some special rules for handling fragments and path MTU discovery. The basic operation is to remove the original IP header and replace it with a new header from the other IP version. The rest of this section provides a high-level overview of the protocol translation process and the issues involved.

IP Translation
The IPv6 and IPv4 headers have some similarity, but there are a number of fields that are either missing or have different sizes or meaning. The translator either directly copies, translates, ignores, or sets fields in the IP header to a default value when translating from one version of IP to the other. Figure 13 illustrates the actions taken by the translator for each header field. Many of the fields require a simple adjustment. The IPv4 *checksum* field is computed when translating from IPv6-to-IPv4, and ignored when translating from IPv4- to-IPv6. The IPv4 *total-length* field includes the IPv4 header size whereas the IPv6 *payload-length* field does not.

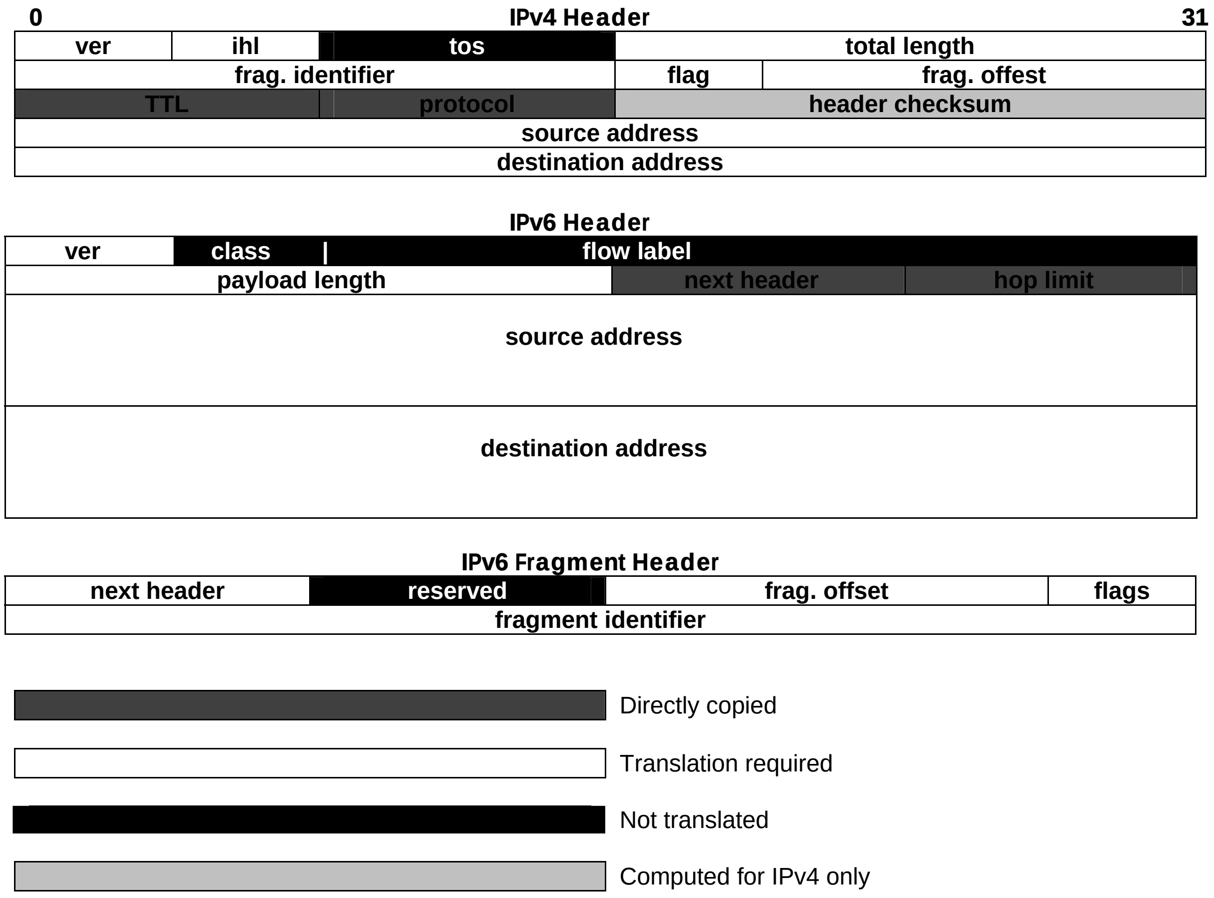


Figure 13 Illustrates which fields of the IPv6/Ipv4 header, are directly copied require translation or are ignored. In contrast to IPv4, IPv6 does not have explicit fields to support fragmentation; it user separate Fragment header for its information.

The translation needs to account for this difference. The *hop-limit/time-to-live* fields are copied and decreased by one. Finally, the *protocol* field can be directly copied from one version of IP to the other, with ICMPv4 and ICMPv6 protocol numbers being the only exception.

With the exception of the IPv6 Fragment header, the translator silently ignores all other IPv6 extension headers and IPv4 options. The translator also ignores the IPv4 type-of service and IPv6 traffic-class and flow-label fields, as there does not exist a semantic mapping between them (specifically, the use of the IPv6 flow-label field has not been specified yet). We discuss this loss of information in Section 4.1 further. When the translator receives a fragmented packet, the translation is straightforward since there is a direct mapping between the IPv4 and IPv6 fragmentation fields. The only caveat is the size difference of the fragment identifier field between the two protocols. In IPv6, this field is 32-bits wide and twice as large as its IPv4 counterpart. To account for this, we currently just copy the lower 16 bits of the IPv6 fragmentation identifier when translating from IPv6 to IPv4. Whenever the translator encounters a non-fragment IPv4 packet with the *Don't Fragment* flag set to false (i.e., fragmentation is allowed for that packet), it notes that by adding an IPv6 Fragment header and copying the IP header of the error-causing packet, which must be translated as well. The IPv4 fragmentation fields to it, which indicates the following:

- The sender allows fragmentation and that the fragmentation information is carried end-to-end to ensure that packets are correctly reassembled.
- The sender is not using path MTU discovery and the *Don't Fragment* bit must be set to false should the packet be translated back to IPv4. The translation from IPv4 to IPv6 increases the packet size by at least 20 bytes due to the header length difference between the two protocols (28 bytes if it needs to add a Fragment header). If the *Don't Fragment* flag is set to true and the resulting packet is greater than the next-hop MTU, then the translator will return an ICMP error message (Packet Too Big). Otherwise, the translator will fragment the resulting packet into next-hop MTU-sized packets. Note that this fragmentation results in an inefficient packet stream in the case where the IPv4 host is sending MTU sized packets (e.g., a network file system, such as NFS). For this situation, we are experimenting with returning ICMPv4 "Packet Too Big" error message to the IPv4 host that contains a next-hop MTU that accounts for the size difference in the IP header size, giving the host the opportunity to re-adjust its path MTU value. If the host continues to send large packets (i.e., it does not support path MTU discovery), then the translator will stop sending the ICMP error message and continue fragmenting the packet.

ICMP Translation

The translator silently drops single hop ICMP messages as well as ICMP messages with unknown Type fields. For the remaining ICMP messages the header format is nearly identical for ICMPv4 and ICMPv6. The only exception is the ICMP Parameter Problem message, which an 8-bit pointer value in ICMPv4 and a 32-bit pointer value in ICMPv6. The following ICMP messages and errors have a counterpart in each version: Echo Request, Echo Reply, Time Exceeded, Destination Unreachable, Packet Too Big, and Parameter Problem. For most cases there is a simple translation of the ICMP Type and Code fields. When a Packet Too Big error message reaches the translator, it needs to adjust the Maximum Transmission Unit (MTU) field during the translation to account for the difference between IPv4 and IPv6 header sizes. Also, for a Parameter Problem error message the Pointer field needs to be adjusted to point to the corresponding field in the error causing IP header. ICMP error messages contain as much of the error invoking packet's IP header and data as can fit, and needs to be translated just like a normal IP header that delivered the message.

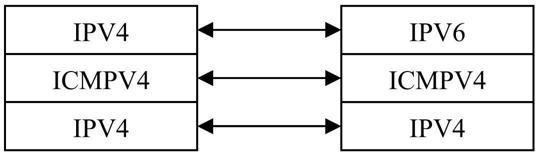


Figure 14, ICMP error message Include the IP header of the error causing packet, which must be translated as well

That is, it requires a recursive translation of the IP packet contained in the ICMP error message, as illustrated in Figure 14. The caveat is that the translation of the IP header is likely to change the length of the datagram, in which case the IPv6 Payload length and IPv4 Total-length fields need to be adjusted as well. Finally, the translator silently drops all IGMP messages.

Adjusting Checksum Values

Several higher-layer protocols (e.g., TCP, UDP) compute their checksum values on a pseudo-header that consists of fields from the IP header. The checksum value needs to be adjusted with the difference between the original IP addresses and the translated IP addresses. The checksum adjustment for ICMP is slightly more complex. ICMPv6 uses a pseudo-header checksum similar to UDP and TCP, whereas ICMPv4 does not. For ICMP Echo Reply and Echo Request informational messages we calculate the incremental checksum adjustment, as only the Type value changes. When translating from ICMPv6 to ICMPv4 we need to subtract the pseudo-header checksum. Conversely, when translation from ICMPv4 to ICMPv6 we need to add the pseudo-header checksum. Note that these informational messages may be fragmented either by the sending host or intermediate routers if their size exceeds the path MTU. For this case, the translator cannot calculate the correct checksum value for ICMP Echo and Echo Request messages, because it does not know the total size of the packet, which it requires to add/subtract the pseudo-header checksum value when translating between the ICMP versions. Finally, since ICMP error messages are never fragmented, our approach is to recalculate the checksum value from scratch rather than incrementally, because most of the ICMP header and data values have changed.

Protocol Translation Details

This appendix describes the protocol translation for both IP and ICMP headers in detail.

Translating IPv4 to IPv6 Headers

If the Don't Fragment flag is true and the IPv4 packet is not a fragment (i.e., the More Fragments flag is false and the Fragment Offset is zero) then the IPv6 header fields are set as follows:

Version: 6
Traffic-Class: 0 (all zero bits)
Flow ID: 0 (all zero bits)
Payload Length: Total Length value from IPv4 header, minus the Internet Header Length (multiplied by 4) value from the IPv4 header
Next Header: Protocol field copied from IPv4 header. If the value of the Protocol field is 1 (ICMPv4), then substitute it with 58 (ICMPv6)
Hop Limit: Time To Live value from IPv4 header decreased by one.
Source and Destination Addresses: Depends on address translation mechanism.
If there is need to add a Fragment header (i.e., the Don't Fragment flag is false or the More Fragments flag is true or the Fragment Offset is non-zero) the IPv6 header fields are set as above with the following exceptions:

Payload Length: Total Length minus the Internet Header Length (multiplied by 4) from the IPv4 header, plus 8 for the Fragment header.
Next Header: 44 (Fragment Header)
The Fragment header fields are set as follows:
Next Header: Protocol field copied from IPv4 header. If the value of the Protocol field is 1 (ICMPv4), then substitute it with 58 (ICMPv6). Reserved: 0 (all zero bits)
Fragment Offset: Fragment Offset copied from the IPv4 header.
M flag: More Fragments flag copied from the IPv4 header.
Identification: The low-order 16 bits copied from the Identification field in the IPv4 header. The high order 16 bits set to zero.

Translating IPv6 to IPv4 Headers
With exception of the IPv6 Fragment header, all other IPv6 extension headers are ignored (i.e., there is no attempt made to translate them). For each IPv6 extension header that is ignored the Payload Length needs to be adjusted by the size of these headers before the IPv4 Total Length field is calculated. If there is no IPv6 Fragment header the IPv4 header fields are set as follows:
Version: 4
Internet Header Length: 5 (no IPv4 options)
Type of Service: 0 (all zero bits)
Total Length: Payload length value from IPv6 header, plus the size of the IPv4 header.
Identification: 0 (all zero bits)
Flags: Don't Fragment flag is set to true (1), and all other flags set to false (0)
Fragment Offset: 0 (all zero bits)
Time To Live: Hop Limit value from IPv6 header decreased by one
Protocol: Next Header copied from IPv6 header or last extension header; and, if the value of the Next Header field is 58 (ICMPv6), then substitute it with 1 (ICMPv4)
Header Checksum: Computed once the IPv4 header has been created.
Source and Destination Address: Depends on address translation mechanism

If the IPv6 packet contains a Fragment header the header fields are set as above with the following exceptions:
Total Length: Payload length value from IPv6 header, minus 8 for the Fragment header, plus the size of the IPv4 header.

Identification: Copied from the low-order 16-bits in the Identification field in the Fragment header.
Flags: The More Fragments flag is copied from the Fragment header and the Don't Fragments flag is set to false.
Fragment Offset: Copied from the Fragment Offset field in the Fragment Header.

Translating ICMPv4 to ICMPv6
Echo Request and Echo Reply (Type 8 and Type 0): set the Type to 128 and 129, respectively.

Destination Unreachable (Type 3): for most Code values set the Type to 1, unless specified otherwise below. Translate the Code field as follows:
Code 0, 1, 6, 7, 8, 11, and 12: set Code to 0 (no route to destination)
Code 2: translate to an ICMPv6 Parameter Problem (Type 4, Code 1) and set the Pointer to 6, which is the IPv6 Next Header field
Code 3: set Code to 4 (port unreachable)
Code 4: translate to an ICMPv6 Packet Too Big message (Type 2, Code 0) and the MTU field needs to be adjusted for the difference between the IPv4 and IPv6 header sizes.
Code 5: set Code to 2 (not a neighbor).
Code 9, 10: set Code to 1 (communication with destination administratively prohibited).
Time Exceeded (Type 11): set the Type field to 3. The Code field is unchanged.
Parameter Problem (Type 12): set the Type field to 4 and translate the Pointer values as follows: 0-to-0, 2- to-4, 8-to-7, 9-to-6, 12-to-8, 16-to-24, and for all other ICMPv4 Pointer values set the ICMPv6 Pointer value to –1.

Translating ICMPv6 to ICMPv4
Echo Request and Echo Reply (Type 128 and 129): set the Type to 0 and 8, respectively.
Destination Unreachable (Type 1): set the Type field to 3. Translate the code field as follows:
Code 0: Set Code to 1 (host unreachable)
Code 1: set Code to 10 (communication with destination host administratively prohibited)
Code 2: set Code to 5 (source route failed)
Code 3: set Code to 1 (host unreachable)
Code 4: set Code to 3 (port unreachable).
Packet Too Big (Type 2): translate to an ICMPv4
Destination Unreachable with code 4. The MTU field needs to be adjusted for the difference between the IPv4 and IPv6 header sizes taking into account whether or not the packet in error includes a Fragment header.
Time Exceeded (Type 3): set the Type to 11. The Code field is unchanged.
Parameter Problem (Type 4): If the Code is 2 then set Type to 12, Code to 0, and Pointer to –1. If the Code is 1 translate this to an ICMPv4 protocol unreachable (Type 3, Code 2) message. If the Code is 0 then set the Type to 12, the Code to 0, and translate the Pointer values as follows: 0-to-0, 4-to-2, 7-to-8, 6-to-9, 8-to-12, 24-to-16, and for all other ICMPv6 Pointer values set the ICMPv4 Pointer value to –1.

4.2 Implementation of the Translators

The Translator was implemented in C for the Linux (Fedora Core 3) kernel 2.x.x. The implantation of the translator for Microsoft Windows Xp is possible by the usage and insertion of a proper .dll library, but difficult due to the non open source nature of the protocol stack implementation.

Major Libraries used

- icmp.h - An implementation of the TCP/IP protocol suite for the LINUX operating system. INET is implemented using the BSD Socket interface as the means of communication with the user level definitions for the ICMP protocol.

Version:	icmp.h 1.0.3 04/28/93
Author:	Fred N. van Kempen, waltje@uWalt.NL.Mugnet.org

Functional modules of the translator

- Convert header: - Converts the header from IPv4 format to IPv6 or vice-versa. The header includes the source and destination address. The address translation takes place here.
- Convert icmp: - converts the icmp packet from one format to other (works both ways).
- Calculate checksum: - calculates the checksum with the newly converted header.
- Send frame: -Send the converted frame to the correct network port.

The control flow of the translator

The working of the translator is described in the flowchart in Figure 15.The translator waits till it receive any IP frame, from the lower layers of the network stack. It at first tries to identify the packets source and destination address with the NAPT protocol format set for the network. If it finds that it was a v4 to v4 or v6 to v6 communication and no conversion is required then it sends it to the lower layer stating that is from a network layer of the destination address type. If v4 to v6 or v6 to v4 packet is found then it is sent for the conversion. A IP packet conversion from IPv4 to IPv6 consist of the three steps explained above .The header conversion this is when the network address translation takes place. The checksum has to be calculated in case the destination header is of IPv4 type (if the destination address is of the IPv6 type the n checksum is not need since, the Ipv6 main header contains no check sum).The calculated checksum is then appended to the header. Finally a new ICMP message of the destination address type is created if required by the destination address. The packet with new header and the new ICMP packet is sent to the lower layer from the port stating it is from the network layer of the destination address type.

The Working of the translator is shown in the following flowchart Figure 15:

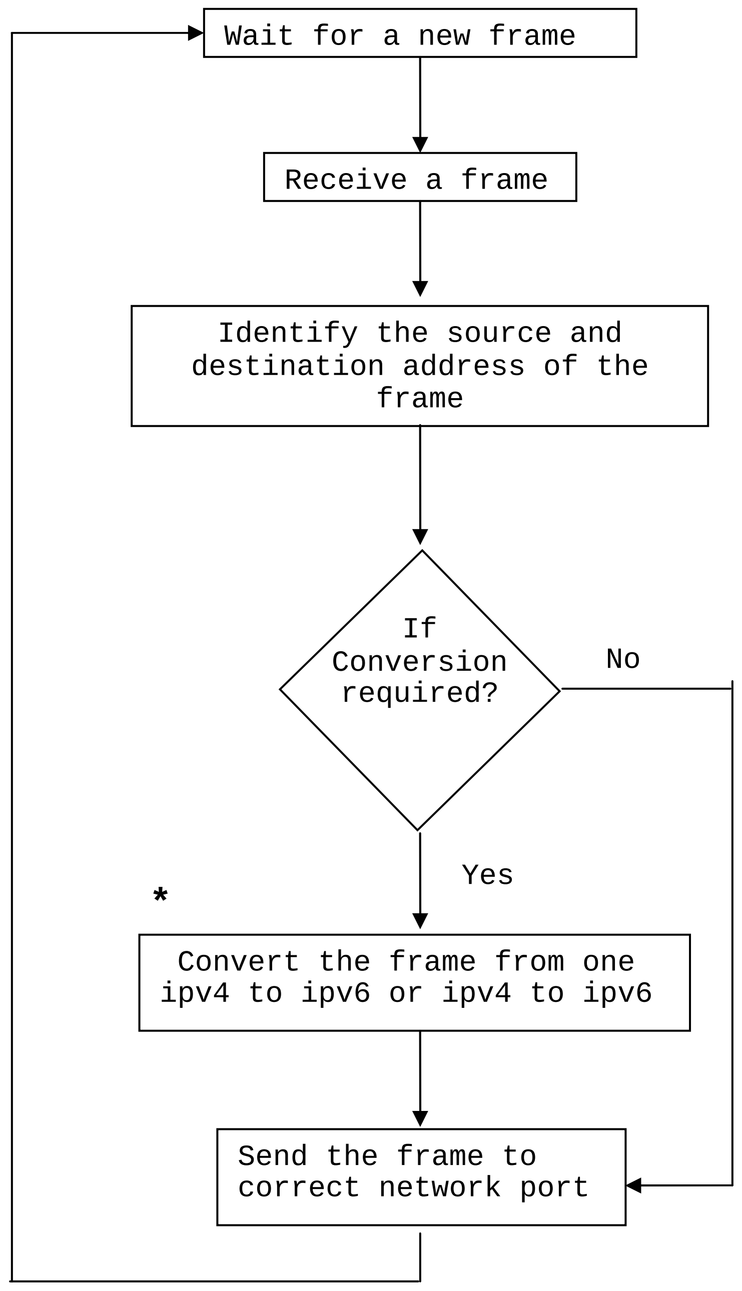

Wait for a new frame
Receive a frame
Identify the source and destination address of the frame
If Conversion required?
No
Yes
*
Convert the frame from one ipv4 to ipv6 or ipv4 to ipv6
Send the frame to correct network port

Figure15 Flow chart showing the working of the translator

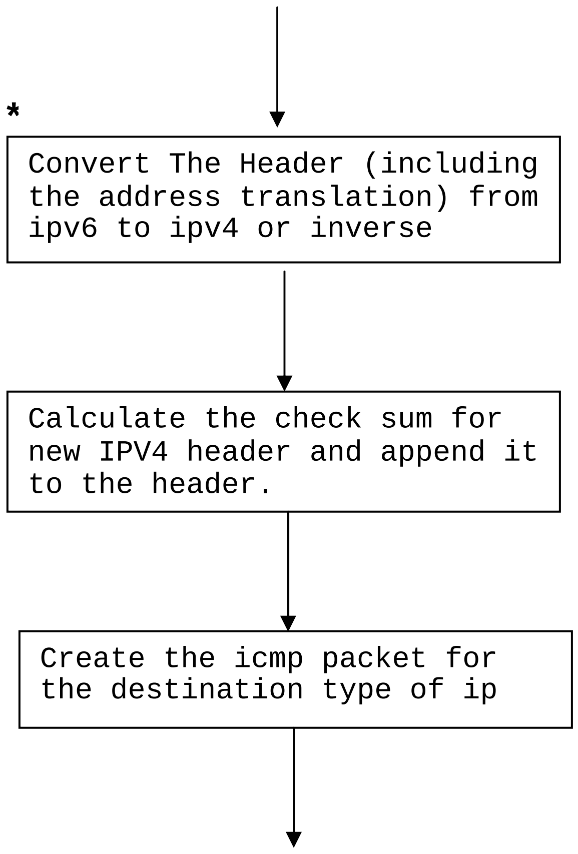

*
Convert The Header (including the address translation) from ipv6 to ipv4 or inverse
Calculate the check sum for new IPV4 header and append it to the header.
Create the icmp packet for the destination type of ip

Figure16 Steps for conversion the frame from one IPv4 to IPpv6 or ipv4 to ipv6

The translator was installed on a machine with equipped with two Ethernet cards and acts as a gateway between the IPv6 and IPv4 Ethernet segments.

The translator can be separately run every time its desired or we can start it as a daemon every time the operating system boots by:
Making an entry if the binary file with its location in rc.d or rc.local

Running the translator
The IPv6/IPv4 translator works for a representative set of "real world"(most of the applications particularly used for in the academic environment) applications that exercise TCP, UDP, and ICMP protocols. Our test applications consist of IPv6 versions of tcp, finger, telnet, ping, traceroute, and ftp, which were able to communicate with their IPv4 counterparts via the translator. Additionally, we were able to view WWW pages served by an IPv6 version of the Apache web-server using IPv4 versions of Netscape and Internet Explorer.

5 Testing the Configured Network and the Translator

To establish the performance of the above configured network the setup was tested and the performance logged. The three scenarios for testing and comparative study were:

- IPv4-IPv4 network: - The purpose of IPv4-IPv4 performance logging was only for a comparative study, with other scenario.
- IPv6-IPv6 network: - The newly configured network with its mentioned better support for some services needed to prove its worth against the older more established one.
- IPv4-IPv6 network: - The performance translator implemented above and its limiting factor needed to be identified by testing.

5.1Tools Used for testing

TTCP: - Test TCP
Test TCP (TTCP) is a command-line sockets-based benchmarking tool for measuring TCP and UDP performance between two systems. It was originally developed for the BSD operating system starting in 1984.

PCATTCP - Port Of TTCP to Windows Sockets
Porting TTCP to Windows Sockets is fairly straightforward.
PCATTCP is a Win32 Console Application. We must run it from the Command Prompt or from a Batch File.
PCATTCP Usage
We must copy PCATTCP to two Windows platforms. One platform will be used as a *receiver* or data sink and the other will be used as a *transmitter* or data source.

Though primarily used for test of TCP and UDP, the round trip latency can be used check the performance of upper layer and the translator.

Ethereal network Analyzer: A high eng GUI based user friendly
Open source tool for network analysis, and observation. The tool is designed for mainly Linux environment but the windows version is also available.

Ethereal network Analyzer is GUI based auto configurable very user friendly tool the configurations for taking the test can be seen at the [2]

5.2 Testing

5.2.1 Ping Test

We measured the roundtrip latency of ping packets ranging in size from 64 bytes to 1440 bytes on 100Mbps Ethernet links. In Table 3, the columns labeled *v4-v4* and *v6-v6* show the latency between two machines communicating directly using the same protocol. The columns labeled *NAPT* show the roundtrip latency our translator.

Msg. Size in bytes	**v4-v4**	**v6-v6**	**NAPT**
64	246	244	424
128	262	261	463
256	297	295	540
512	364	360	658
1024	487	482	918
1440	603	596	1104

Table4. The roundtrip latency of ping packets

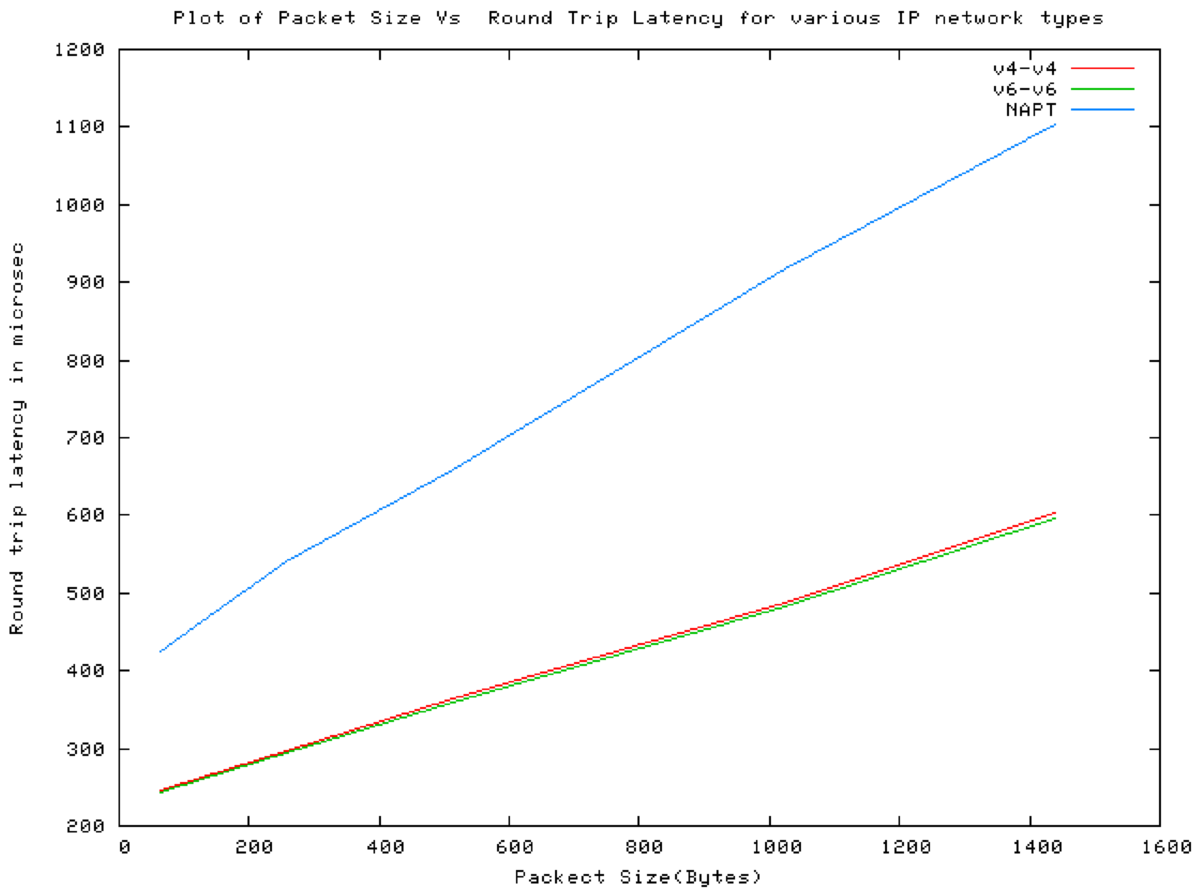


Figure 17: Plot between Packet size and round trip ping latency for various networks

Interpretations

- The roundtrip latency for an IPv6-v6 ping packet is lesser than that of the IPv4-v4.
- The difference of the latency is more or less constant with varying size of the packet.
- The latency shows a marked increase with the translator (almost double).This can be attributed to the overheads of the translator.

Table5 shows the bandwidth of sending 64 Mbytes using TCP for both 10Mbps and 100Mbps Ethernet.

Link Speed	**v4-v4(Kbytes/s)**	**v6-v6 (Kbytes/s)**	**NAPT (Kbytes/s)**
Ethernet	1095	1092	1089
Fast Ether	11003	9076	7210

Table5. TCP bandwidth in Ethernet and Fast Ethernet

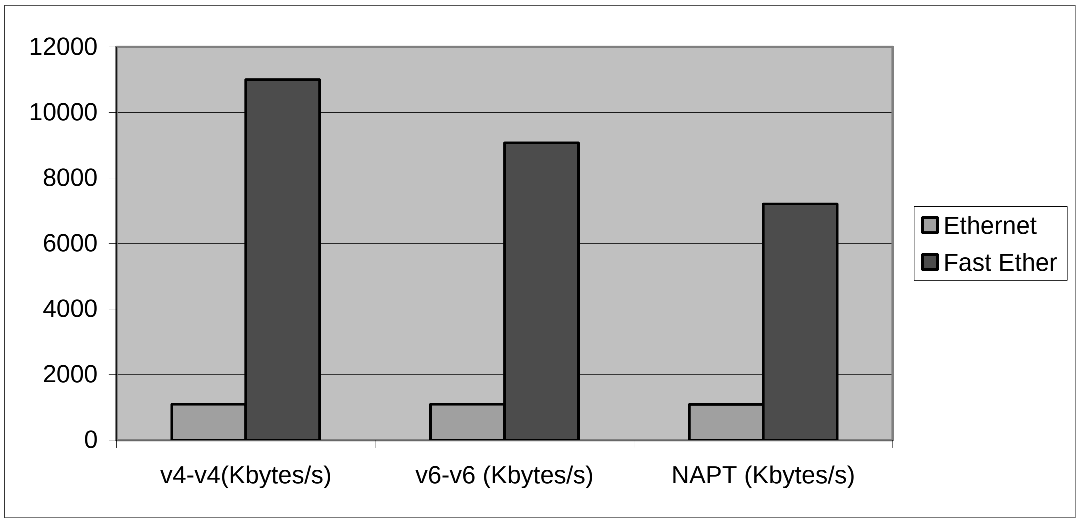


Figure18: TCP bandwidth in Ethernet and Fast Ethernet

Interpretations

- The bandwidth measured for IPv6-IPv6 network shows slight increase over the IPv4-v4 in both the cases.
- For 10Mbps Ethernet the overhead of the translator is unnoticeable. However, the bandwidth for the translator on fast Ethernet is much lower compared to two machines communicating directly using either IPv4 or IPv6. We attribute this performance degradation partly to the IPv6 prototype, which is roughly 1.9Mbytes/second slower than the production IPv4 stack. We expect the end-to-end TCP bandwidth to improve as the IPv6 implementation.

5.2.2 FTP Test

We measured the bandwidth utilization during a file transfer with the ftp protocol with the file sizes ranging from 100 Bytes to 1 Mega byte.

File size	v4-v4(Mbytes/s)	v6-v6(Mbytes/s)	NAPT(Mbytes/s)
100 B	0.168	0.182	0.350
1 KB	5.485	5.603	10.999
10 KB	4.478	4.539	8.965
100 KB	43.691	44.102	87.401
1MB	446.326	450.459	892.652

Table6. Network bandwidth usage for a file transfer protocol

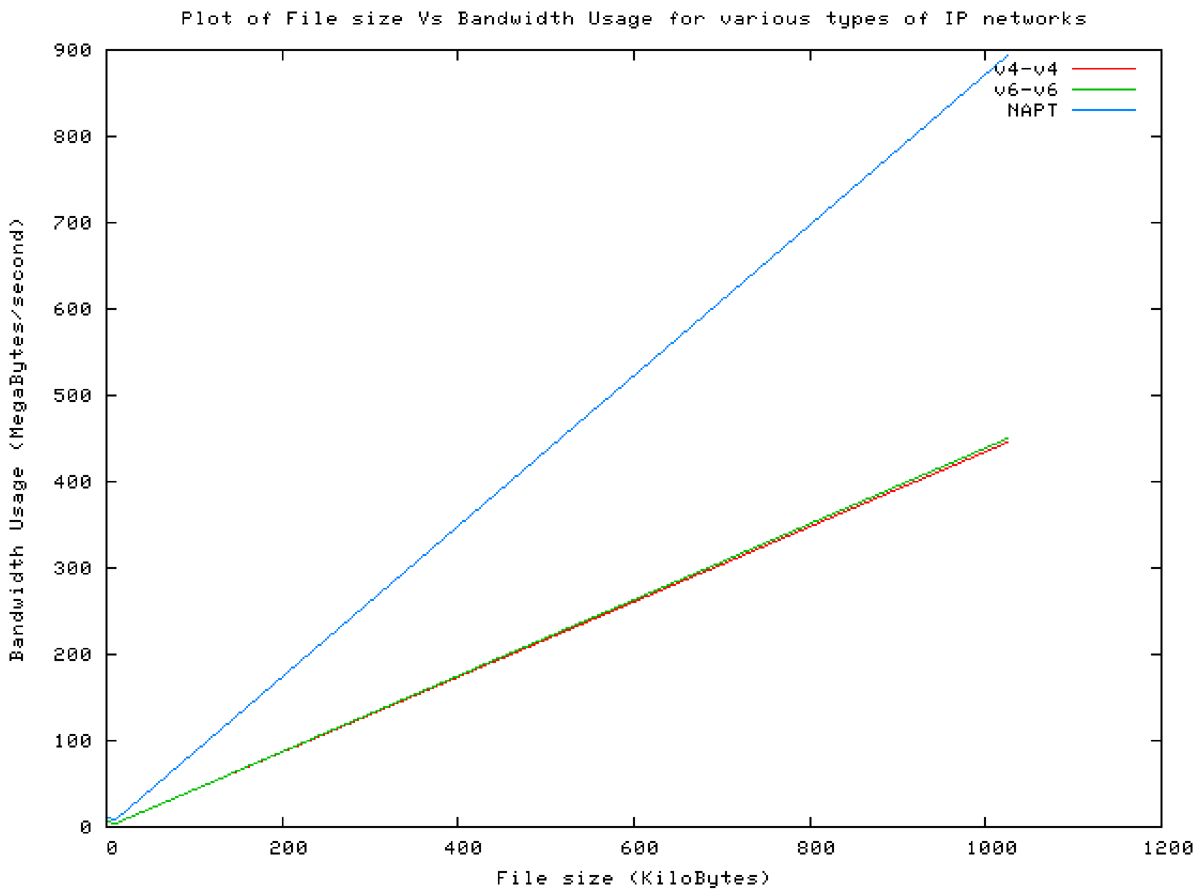


Figure19: Plot between file size and bandwidth usage for various networks

Interpretations

- The bandwidth usage measured for IPv6-IPv6 file transfer shows a slight increase over the IPv4-v4 in both the cases.

- The classical observation of lesser bandwidth utilization of an optimal file size (here 10 kb) over smaller file sizes is shown bi IPv6 also.
- The ratio of the NAPT bandwidth usage to pure network bandwidth usage (any of the two) is more for smaller file sizes.

5.2.3 Media Streaming Test

To conduct the media streaming test we streamed a continuous media (audio stream) through both the IPv4 and the IPv6 stack using the VLC media player. The continuous stream was recorded on the Ethereal Network Monitor. The test was conducted for separate IPV6 and IPv4 networks. Below is the snap-short of the Ethereal Network Monitor at certain` time instant. It shows both the traffics of IPv6 and Ipv4 separately at a time, this could be done by simultaneously broadcasting two streams of the same media from the gateway one into the IPv6 network other into the IPv4 network

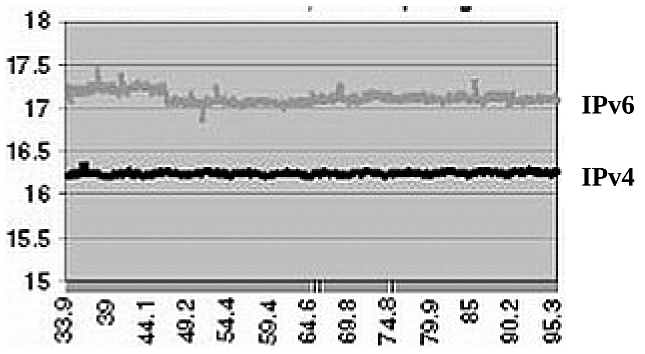


Figure 20, Snap-Short of the Ethereal Network Analyzer showing IPv6 and IPv4 bandwidth usage separately. **The Horizontal axis shows Seconds and the Vertical axis shows Kbytes/second.**

The Interpretations

- Again we can see in Figure 15 that in the absence of any competing traffic and with a link of much higher capacity than the audio codec could ever want, we obtain a steady transmission rate in IPV6 of around 17 KBytes=s; around 7% larger than the IPv4 transmission rate. This difference is due to the fact that the Data-Link layer was carrying 294-byte packets in the case of IPv4, and 314-byte packets in the case of IPv6. The standard IPv6 header is 20 bytes larger than the standard IPv4 header, which produces the 7% overhead. This is in fact an expected and known result, since the larger IPv6 header introduces some overhead, especially in relatively low-rate transmissions.

5.2.4 Testing the translator

We knew from our experiments with ttcp that the TCP protocol translation works, but wanted to verify this with common TCP applications. We were able to use telnet and finger to connect between IPv6 and IPv4 hosts through the translator. Additionally, a web browser on an IPv4 host retrieving documents from an IPv6 Apache web-server was equally successful.

The ping program uses ICMP messages to determine whether a particular host is alive. We also used ping to measure basic roundtrip latency between hosts.

The traceroute program tracks the flow of a packet from router to router. When tracking routes from an IPv6 node through the translator along an IPv4 network, the addresses of the IPv4 routers are translated into IPv4-mapped IPv6 addresses. For the other direction, the translator establishes bindings for the IPv6 router addresses to private network addresses.

Although ping and traceroute use ICMP, they do not adequately test whether the recursive ICMP translation was working properly. Table 3 lists how we caused various ICMP error messages to verify their correct translation.

ICMP Error Message	**Error causing action**
Destination unreachable	UDP packet to unreachable port
Packet Too Big	packet exceeding path MTU size
Time Exceeded	single incomplete IP fragment
Parameter Problem	packet with invalid field

Table 7 ICMP error messages and corresponding actions

As shown in the ftp test ftp, which is an application that embeds an ASCII IP address and sends it to its peer. For it to work correctly via the translator, the IPv6 implementation of the ftp client needs to detect whether the connection is with an IPv6 or IPv4 version of the ftp daemon. When communicating with an IPv4 ftp daemon it needs to use as an ASCII IP address of its host's IPv4-compatible IPv6 address instead of the host's native IPv6 address. Conversely, when an IPv4 ftp client contacts an IPv6 ftp daemon, the daemon must treat the ASCII IP address as an IPv4-mapped IPv6 address. With this approach it is not necessary for the translator to update the ASCII IP address.

CPU utilization of the translator

- Windows Xp: Using NT's performance monitor we noticed that processor utilization reaches nearly 100% on our forwarder/translator machine when running the ttcp bandwidth benchmark over fast Ethernet. The reason for the high CPU utilization is NT's packet receive architecture, which assumes the device driver owns the packet buffer rather than passing buffer ownership to the module receiving the packet (as is the case in most UNIX systems). Consequently, we believe that bandwidth through the translator and the forwarder are CPU limited, as they incur significant overhead due to NT's packet receive architecture; they must allocate buffer space for the IP packet's payload and copy the data in its entirety before being able to forward it.

- Fedora Core 3(GNU/Linux):- The CPU utilization in UNIX was varying from 40 to 60%, this advantage in CPU performance is due to the fact the daemon handling the translation owns the packet buffer.

We are pleased with the current latency and bandwidth measurements, as they indicate that translation does not inherently have a significant impact on performance.

5.3 Major Observations, IPv6 compared to IPv4

- We saw that which ever type of data transmission took place the loose packets, files or media stream the IPv6 tends to use larger bandwidth than IPv4.The value in percentage increase in bandwidth utilization of IPv6 over IPv4 for the File transfer is shown below in the Table

.

File size	**Increase v4 to v6 in percent**
100 B	8.333
1 KB	2.151
10 KB	1.362
100 KB	0.941
1MB	0.962

Table8,Percentage increase in the bandwidth usage of IPv6 to IPv4 for various file sizes.

- This increase in the bandwidth usage of IPv6 to IPv4 is reduces with increase in the amount of data. For protocols that use very small packets such as Voice over IP (VoIP) or Simple Network Management Protocol (SNMP), the increase will be more pronounced.
- The round trip latency or the time spend in the network travel was lesser for IPv6 than for IPv4.
- Faster the network media greater the bandwidth usage increase of IPv6 to Ipv4.

Hence, we see IPv6 if available uses a larger amount of bandwidth for faster network travel (round trip latency).

6. Conclusion

The Migration of a network setup of an Academic/Technical Institution of the size of Delhi College Of Engineering from IPv4 to IPv6, can be successfully completed within a period of one to two years. A completed migration is not suggested (in a way not possible) until all or most of the applications and software come up with a new IPv6 compatible versions. Dual layer support is more appropriate for the time being. Though an IPv6 only network can also be maintained with all basic applications available such as browsers, databases, web servers, media players, application servers etc

But an IPv6 infrastructure is very necessary in technical institutions either in a Dual stack form or in a separate Ipv6 setup. All new applications, specifically network algorithms and programs should have IPv6 compatibility.Ipv6 protocol stack is transparent and easy to program with.

Ipv6 is future, lot of its strengths and characteristics are yet to be discovered and utilized to the fullest. The standard itself may see few additions (for example new packet type for secure or media streams.), and the protocol stack implementation of IPv6 will definitely improve in the future.

Once a complete shift or the Dual Stack arrangement is set, the need of the translator will be eliminated. The translator only work for a period when the dual stack arrangement is not ready or when specific pure IPv4 and IPv6 networks are required.

7. Follow UP and Future Scope

The project of migration of the network protocol for large institution is always large and time taking. New ideas and challenges come up along the way.

- In the implementation of various IPv6 compatible applications during the end days of this thesis the Apache Web Browser was also configured to run IPv6 network hosting Ipv6 sites. But a lot of other applications such as the databases, application servers are yet to be tried.
- The translator needs to be improved specifically the Windows version. This could be done by gathering more information of the IPv6 protocol stack of Windows, and hence writing an efficient .dll file.
- More testing with variable continuous network traffic such a Web server needs to conduct.
- The Whole institution (or a larger part of it) can be given dual layer support with both IPv6 and Ipv4.
- With the above step done in other word when we have large multi-networked variable traffic network in IPv6, we can also study the various routing techniques and their resulting with IPv6 can field tested.
- Once The ERNET global IPv6 address is available a global static IPv6 gateway will be present connect to the ERNET IPv6 backbone in India. This would allow new avenues for networking.
- Finally the last step in the migration process earlier mentioned “The Gradual Shift” is to be implemented will take another full fledged project of this size.

7.1New Ideas

The setting up of the IPv6 network laboratory in the college has brought about a new burst of ideas in relation with the new IPv6 Protocol or the Network layer in general.

7.1.2 The Generic IP

IP version 4 then IP version 6, after that what? Though it has been theoretically proved that the IPv6 address will cater to a network population large enough for very long time, but other changes in the header or the ICMP format may develop from future developments etc.

An IP packet with variable header format may be the solution. The real header will be preceded by information about the header format a **pre-header**. This pre-header will contain the number of bits in the source and destination address fields, the next headers or any other fields in the header and their formats. The network layer will only be able to actually read the real header after it has got the information from the pre-header what are the fields present in the header and their length.

The overhead of repeatedly processing the pre-headers for similar kind of packets can be reduced by using **template-headers**. The general purpose header widely used can be set out as standard templates with template numbers contained in their pre headers. This format of these templates-headers will be either already known to the network layer or be updated by the first incoming IP packet. Once the network layer sees the pre-header with the template number whose format information already known (either by a preceding packet or by a standard template) hen it will ignore new the rest of the pre-header.
The straightforward advantages of this method are

- The number or bits used for addressing may grow shrink or vary any ways without requirement of any major protocol overhaul.
- Multiple addressing systems may coexist and intercommunicate without any need of separate translator.
- The fields of the header may vary giving flexibility to send whatever information to be sent, in their desired length and format.

To implement the above idea the whole way the network layer now works may needed to be changed, the network protocol implementation would require efficient and intelligent algorithms to extract the format in from the pre-headers and to maintain and update templates. The network layer will require a larger processing power and time than now to complete the above tasks, but with greater processing power we may think of it.

Appendix A

Outputs of the Commands and Contents of files

- **[root@pc08 dhcp-0.10]#./dhcp6s -dDf eth0 /*Command Snapshort6*/**

```
Jun/15/2006 00:46:39 found an interface eth0 hardware fee369a0
Jun/15/2006 00:46:39 generated a new DUID:
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e
Jun/15/2006 00:46:39 saved generated DUID to /var/lib/dhcpv6/dhcp6s_duid
Jun/15/2006 00:46:39 set timer for syncing file ...
Jun/15/2006 00:46:39 configure duid is
00:01:00:01:05:c8:8c:7e:00:10:a4:8d:30:7f
Jun/15/2006 00:46:39 interface definition for eth0 is ok
Jun/15/2006 00:47:32 received message packet info addr is ff02::1:2, scope
id (2)
Jun/15/2006 00:47:32 received solicit from fe80::2d0:b7ff:fee3:d13e%eth0
Jun/15/2006 00:47:32 get DHCP option client ID, len 14
Jun/15/2006 00:47:32   DUID: 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e
Jun/15/2006 00:47:32 get DHCP option opt_8, len 2
Jun/15/2006 00:47:32 this message elapsed time is: 0
Jun/15/2006 00:47:32 get DHCP option opt_3, len 12
Jun/15/2006 00:47:32 get option iaid is 3820474368, renewtime 0, rebindtime 0
Jun/15/2006 00:47:32 get DHCP option option request, len 2
Jun/15/2006 00:47:32   requested option: DNS_RESOLVERS
Jun/15/2006 00:47:32 client ID 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e
Jun/15/2006 00:47:32 server preference is ff
Jun/15/2006 00:47:32 option type is 0
Jun/15/2006 00:47:32 iaid 3820474368 iaidaddr for client duid
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e doesn't exists
Jun/15/2006 00:47:32 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:47:32 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)
Jun/15/2006 00:47:32 new address 2001:e30:1402:1::4 is got
Jun/15/2006 00:47:32  preferlifetime 130, validlifetime 200
Jun/15/2006 00:47:32  renewtime 60, rebindtime 90
Jun/15/2006 00:47:32  status code: success
Jun/15/2006 00:47:32 set client ID
Jun/15/2006 00:47:32 set server ID
Jun/15/2006 00:47:32 set IA_NA iaidinfo: iaid 3820474368 renewtime 60
rebindtime 90
Jun/15/2006 00:47:32 set IADDR address option len 30: 2001:e30:1402:1::4
preferlifetime 130 validlifetime 200
Jun/15/2006 00:47:32   this address status code: success
Jun/15/2006 00:47:32 set opt_3
Jun/15/2006 00:47:32 server preference ff
Jun/15/2006 00:47:32 set preference
Jun/15/2006 00:47:32 set status code
Jun/15/2006 00:47:32 send destination address is
fe80::2d0:b7ff:fee3:d13e%eth0, scope id is 2
Jun/15/2006 00:47:32 transmit advertise to fe80::2d0:b7ff:fee3:d13e%eth0
Jun/15/2006 00:47:32 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:47:32 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)
Jun/15/2006 00:47:32 DUID is 00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e,
DUID_LEN is 14
Jun/15/2006 00:47:32 removing ID (ID:
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e)
Jun/15/2006 00:47:32 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:47:32 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)
Jun/15/2006 00:47:32 DUID is , DUID_LEN is 0
Jun/15/2006 00:47:32 received message packet info addr is ff02::1:2, scope
id (2)
Jun/15/2006 00:47:32 received request from fe80::2d0:b7ff:fee3:d13e%eth0
Jun/15/2006 00:47:32 get DHCP option client ID, len 14
Jun/15/2006 00:47:32   DUID: 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e
Jun/15/2006 00:47:32 get DHCP option server ID, len 14
Jun/15/2006 00:47:32   DUID: 00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e
Jun/15/2006 00:47:32 get DHCP option opt_8, len 2
Jun/15/2006 00:47:32  this message elapsed time is: 0
Jun/15/2006 00:47:32 get DHCP option opt_3, len 12
Jun/15/2006 00:47:32 get option iaid is 3820474368, renewtime 0, rebindtime 0
Jun/15/2006 00:47:32 get DHCP option option request, len 2
```

```
Jun/15/2006 00:47:32   requested option: DNS_RESOLVERS
Jun/15/2006 00:47:32 client ID 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e
Jun/15/2006 00:47:32 server preference is ff
Jun/15/2006 00:47:32 iaid 3820474368 iaidaddr for client duid
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e doesn't exists
Jun/15/2006 00:47:32 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:47:32 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)
Jun/15/2006 00:47:32 new address 2001:e30:1402:1::5 is got
Jun/15/2006 00:47:32  preferlifetime 130, validlifetime 200
Jun/15/2006 00:47:32  renewtime 60, rebindtime 90
Jun/15/2006 00:47:32 start date is 1118776652
Jun/15/2006 00:47:32 write lease 2001:e30:1402:1::5/64 to lease file
Jun/15/2006 00:47:32 add lease for 2001:e30:1402:1::5/64 iaid 3820474368
with preferlifetime 130 with validlifetime 200
Jun/15/2006 00:47:32 hash_add an iaidaddr 3820474368 for client duid
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e
Jun/15/2006 00:47:32  status code: success
Jun/15/2006 00:47:32 set client ID
Jun/15/2006 00:47:32 set server ID
Jun/15/2006 00:47:32 set IA_NA iaidinfo: iaid 3820474368 renewtime 60
rebindtime 90
Jun/15/2006 00:47:32 set IADDR address option len 30: 2001:e30:1402:1::5
preferlifetime 130 validlifetime 200
Jun/15/2006 00:47:32   this address status code: success
Jun/15/2006 00:47:32 set opt_3
Jun/15/2006 00:47:32 server preference ff
Jun/15/2006 00:47:32 set preference
Jun/15/2006 00:47:32 set status code
Jun/15/2006 00:47:32 send destination address is
fe80::2d0:b7ff:fee3:d13e%eth0, scope id is 2
Jun/15/2006 00:47:32 transmit reply to fe80::2d0:b7ff:fee3:d13e%eth0
Jun/15/2006 00:47:32 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:47:32 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)
Jun/15/2006 00:47:32 DUID is 00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e,
DUID_LEN is 14
Jun/15/2006 00:47:32 removing ID (ID:
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e)
Jun/15/2006 00:47:32 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:47:32 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)
Jun/15/2006 00:47:32 DUID is 00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e,
DUID_LEN is 14
Jun/15/2006 00:47:32 removing ID (ID:
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e)
```

- **[root@pc212 ~]# dhcp6c -dDf eth0/*Command Snapshort8*/**

```
Jun/15/2006 00:43:58 <3>[interface] (9)
Jun/15/2006 00:43:58 <5>[eth0] (4)
Jun/15/2006 00:43:58 <3>begin of closure [{] (1)
Jun/15/2006 00:43:58 <3>comment [# send rapid-commit;] (20)
Jun/15/2006 00:43:58 <3>comment [# request prefix-delegation;] (28)
Jun/15/2006 00:43:58 <3>[request] (7)
Jun/15/2006 00:43:58 <3>[domain-name-servers] (19)
Jun/15/2006 00:43:58 <3>end of sentence [;] (1)
Jun/15/2006 00:43:58 <3>comment [# request temp-address;] (23)
Jun/15/2006 00:43:58 <3>comment [#  iaid 11111;] (14)
Jun/15/2006 00:43:58 <3>comment [#  address {] (12)
Jun/15/2006 00:43:58 <3>comment [#       2001:e30:1402:1::16/64;] (31)
Jun/15/2006 00:43:58 <3>comment [#prefer-life-time 6000;] (23)
Jun/15/2006 00:43:58 <3>comment [#valid-life-time 8000;] (22)
Jun/15/2006 00:43:58 <3>comment [# };] (4)
Jun/15/2006 00:43:58 <3>comment [#renew-time 11000;] (18)
Jun/15/2006 00:43:58 <3>comment [#rebind-time 21000;] (19)
Jun/15/2006 00:43:58 <3>end of closure [}] (1)
Jun/15/2006 00:43:58 <3>end of sentence [;] (1)
Jun/15/2006 00:43:58 extracted an existing DUID from
/var/lib/dhcpv6/dhcp6c_duid: 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e
Jun/15/2006 00:43:58 link local addr is fe80::2d0:b7ff:fee3:d13e
Jun/15/2006 00:43:58 res addr is fe80::2d0:b7ff:fee3:d13e%eth0/28
Jun/15/2006 00:43:58 found an interface eth0 hardware
00:ffffffd0:ffffffb7:ffffffe3:ffffffd1:3e
Jun/15/2006 00:43:58  create iaid 3820474368 for interface eth0
Jun/15/2006 00:43:58 found an interface eth0 hardware
```

```
00:ffffffd0:ffffffb7:ffffffe3:ffffffd1:3e
Jun/15/2006 00:43:58  found interface eth0 iaid 3820474368
Jun/15/2006 00:43:58 interface eth0 iaid is 3820474368
Jun/15/2006 00:43:58 open_netlink_socket called
Jun/15/2006 00:43:58 netlink_send_rtmsg called
Jun/15/2006 00:43:58 netlink_recv_rtgenmsg called
Jun/15/2006 00:43:58 netlink_recv_rtgenmsg error
Jun/15/2006 00:43:58 netlink_send_rtgenmsg called
Jun/15/2006 00:43:58 netlink_recv_rtgenmsg called
Jun/15/2006 00:43:58 get_if_flags called
Jun/15/2006 00:43:58 get_if_flags called
Jun/15/2006 00:43:58 netlink_recv_rtgenmsg error
Jun/15/2006 00:43:58 create an event 0x8442340 xid 0 for state 0
Jun/15/2006 00:43:58 reset a timer on eth0, state=INIT, timeo=0, retrans=831
Jun/15/2006 00:43:59 ifp 0x843e008 event 0x8442340 a new XID (c9428a) is
generated
Jun/15/2006 00:43:59 set client ID
Jun/15/2006 00:43:59 set opt_8
Jun/15/2006 00:43:59 set IA_NA iaidinfo: iaid 3820474368 renewtime 0
rebindtime 0
Jun/15/2006 00:43:59 set opt_3
Jun/15/2006 00:43:59 set option request
Jun/15/2006 00:43:59 send dst if eth0 addr is ff02::1:2%eth0 scope id is 2
Jun/15/2006 00:43:59 send solicit to ff02::1:2%eth0
Jun/15/2006 00:43:59 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:43:59 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)Jun/15/2006 00:43:59 DUID is ,
DUID_LEN is 0
Jun/15/2006 00:43:59 reset a timer on eth0, state=SOLICIT, timeo=0,
retrans=1068Jun/15/2006 00:43:59 receive packet info ifname eth0, addr is
fe80::2d0:b7ff:fee3:d13e scope id is 2
Jun/15/2006 00:43:59 receive advertise from fe80::2d0:b7ff:fee3:d10e%eth0
scope id 2 eth0
Jun/15/2006 00:43:59 get DHCP option client ID, len 14
Jun/15/2006 00:43:59   DUID: 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e
Jun/15/2006 00:43:59 get DHCP option server ID, len 14
Jun/15/2006 00:43:59   DUID: 00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e
Jun/15/2006 00:43:59 get DHCP option opt_3, len 46
Jun/15/2006 00:43:59 get option iaid is 3820474368, renewtime 60,
rebindtime 90
Jun/15/2006 00:43:59   IA address option: opt_5, len 30
Jun/15/2006 00:43:59   get IAADR address information: 2001:e30:1402:1::4
preferlifetime 130 validlifetime 200
Jun/15/2006 00:43:59 status code for this address is: success
Jun/15/2006 00:43:59 get DHCP option preference, len 1
Jun/15/2006 00:43:59 get option preferrence is ff
Jun/15/2006 00:43:59 get DHCP option status code, len 2
Jun/15/2006 00:43:59   this message status code: success
Jun/15/2006 00:43:59 ifp 0x843e008 event 0x8442340 id is c9428a
Jun/15/2006 00:43:59 server ID: 00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e,
pref=ff
Jun/15/2006 00:43:59 status code: success
Jun/15/2006 00:43:59 new server DUID
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e, len 14
Jun/15/2006 00:43:59 reset a timer on eth0, state=REQUEST, timeo=0,
retrans=1067Jun/15/2006 00:43:59 ifp 0x843e008 event 0x8442340 a new XID
(1b4daa) is generated
Jun/15/2006 00:43:59 current server ID
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0eJun/15/2006 00:43:59  IAID is
3820474368
Jun/15/2006 00:43:59 set client ID
Jun/15/2006 00:43:59 set server ID
Jun/15/2006 00:43:59 set opt_8
Jun/15/2006 00:43:59 set IA_NA iaidinfo: iaid 3820474368 renewtime 0
rebindtime 0
Jun/15/2006 00:43:59 set opt_3
Jun/15/2006 00:43:59 set option request
Jun/15/2006 00:43:59 send dst if eth0 addr is ff02::1:2%eth0 scope id is 2
Jun/15/2006 00:43:59 send request to ff02::1:2%eth0
Jun/15/2006 00:43:59 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:43:59 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)Jun/15/2006 00:43:59 DUID is
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e, DUID_LEN is 14
Jun/15/2006 00:43:59 removing ID (ID:
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e)Jun/15/2006 00:43:59 DUID is
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e, DUID_LEN is 14
Jun/15/2006 00:43:59 removing ID (ID:
```

```
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)Jun/15/2006 00:43:59 DUID is
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e, DUID_LEN is 14
Jun/15/2006 00:43:59 removing ID (ID:
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e)Jun/15/2006 00:44:00 receive
packet info ifname eth0, addr is fe80::2d0:b7ff:fee3:d13e scope id is 2
Jun/15/2006 00:44:00 receive reply from fe80::2d0:b7ff:fee3:d10e%eth0
scope id 2 eth0
Jun/15/2006 00:44:00 get DHCP option client ID, len 14
Jun/15/2006 00:44:00   DUID: 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e
Jun/15/2006 00:44:00 get DHCP option server ID, len 14
Jun/15/2006 00:44:00   DUID: 00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e
Jun/15/2006 00:44:00 get DHCP option opt_3, len 46
Jun/15/2006 00:44:00 get option iaid is 3820474368, renewtime 60,
rebindtime 90
Jun/15/2006 00:44:00   IA address option: opt_5, len 30
Jun/15/2006 00:44:00   get IAADR address information: 2001:e30:1402:1::5
preferlifetime 130 validlifetime 200
Jun/15/2006 00:44:00 status code for this address is: success
Jun/15/2006 00:44:00 get DHCP option preference, len 1
Jun/15/2006 00:44:00 get option preferrence is ff
Jun/15/2006 00:44:00 get DHCP option status code, len 2
Jun/15/2006 00:44:00   this message status code: success
Jun/15/2006 00:44:00 reply message XID is (1b4daa)
Jun/15/2006 00:44:00 ifp 0x843e008 event 0x8442340 id is 1b4daa
Jun/15/2006 00:44:00 serverID is 00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e
len is 14
Jun/15/2006 00:44:00 status code: success
Jun/15/2006 00:44:00 open_netlink_socket called
Jun/15/2006 00:44:00 netlink_send_rtmsg called
Jun/15/2006 00:44:00 netlink_recv_rtgenmsg called
Jun/15/2006 00:44:00 netlink_recv_rtgenmsg error
Jun/15/2006 00:44:00 netlink_send_rtgenmsg called
Jun/15/2006 00:44:00 netlink_recv_rtgenmsg called
Jun/15/2006 00:44:00 get_if_flags called
Jun/15/2006 00:44:00 get_if_flags called
Jun/15/2006 00:44:00 netlink_recv_rtgenmsg error
Jun/15/2006 00:44:00 assigned address 2001:e30:1402:1::5 prefix len is not
in any RAs prefix length using 64 bit instead
Jun/15/2006 00:44:00 try to add address 2001:e30:1402:1::5
Jun/15/2006 00:44:00 add an address 2001:e30:1402:1::5 on eth0
Jun/15/2006 00:44:00 renew time 60, rebind time 90
Jun/15/2006 00:44:00 set timer for checking link ...
Jun/15/2006 00:44:00 set timer for checking DAD ...
Jun/15/2006 00:44:00 set timer for syncing file ...
Jun/15/2006 00:44:00 removing an event 0x8442340 on eth0, state=3, xid=1b4daa
Jun/15/2006 00:44:00 got an expected reply, sleeping.
Jun/15/2006 00:44:00 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:44:00 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)Jun/15/2006 00:44:00 DUID is
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e, DUID_LEN is 14
Jun/15/2006 00:44:00 removing ID (ID:
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e)Jun/15/2006 00:44:05 enter
checking dad ...
Jun/15/2006 00:44:05 enter checking link ...
Jun/15/2006 00:44:10 enter checking link ...
Jun/15/2006 00:44:15 enter checking link ...
Jun/15/2006 00:44:16 received a signal (2)
Jun/15/2006 00:44:16 remove an address 2001:e30:1402:1::5 on eth0
Jun/15/2006 00:44:16  remove all events on interface
Jun/15/2006 00:44:16 removing server (ID:
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e)
Jun/15/2006 00:44:16 DUID is 00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e,
DUID_LEN is 14
Jun/15/2006 00:44:16 removing ID (ID:
00:01:00:01:0a:2e:39:49:00:d0:b7:e3:d1:3e)Jun/15/2006 00:44:16 DUID is
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e, DUID_LEN is 14
Jun/15/2006 00:44:16 removing ID (ID:
00:01:00:01:0a:41:e9:97:00:d0:b7:e3:d1:0e)
```

- **[root@pc08 dibbler]# ./dibbler-server run /*Command Snapshort12*/**

```
| Dibbler - a portable DHCPv6, version 0.4.0(SERVER)
| Authors : Tomasz Mrugalski<thomson(at)klub.com.pl>,Marek
Senderski<msend(at)o2 .pl>
| Licence : GNU GPL v2 or later. Developed at Gdansk University of
Technology.
| Homepage: http://klub.com.pl/dhcpv6/
```

```
2006.06.15 01:19:54 Server Notice    My pid (5059) is stored in
/var/lib/dibbler /server.pid
2006.06.15 01:19:54 Server Notice    Detected iface sit0/3, flags=128,
MAC=00:00 :00:00.
2006.06.15 01:19:54 Server Notice    Detected iface eth0/2, flags=4163,
MAC=00:d 0:b7:e3:d1:0e.
2006.06.15 01:19:54 Server Notice    Detected iface lo/1, flags=73,
MAC=00:00:00 :00:00:00.
2006.06.15 01:19:54 Server Debug     Parsing config file...
19:54 Info      Interface eth0/2 configuration has been loaded.
19:54 Notice    Running in stateful mode.
19:54 Info      My duid is 00:01:00:00:42:9c:2a:10:00:d0:b7:e3:d1:0e.
19:54 Notice    Creating multicast (ff02::1:2) socket on eth0/2 interface.
19:54 Notice    Accepting connections. Next event in 2147483647 second(s).
20:01 Notice    Received SOLICIT on eth0/2,TransID=0x321a48, 3 opts: 1 3
8, 0 re lay(s).
20:01 Info      Client has 0 addrs, asks for 1, 65532 is available, limit
for cl ient is 4294967295, 1 will be assigned.
20:01 Info      Client requested ::, got 2001:e30:1402:1::e835 (IAID=2,
pref=180 0,valid=3600).
20:01 Notice    Sending ADVERTISE on eth0/2,transID=0x321a48, opts: 1 3 2
7, 0 r elays.
20:01 Notice    Accepting connections. Next event in 2147483647 second(s).
20:03 Notice    Received REQUEST on eth0/2,TransID=0x273fdd, 4 opts: 1 3 8
2, 0 relay(s).
20:03 Info      Client has 0 addrs, asks for 1, 65532 is available, limit
for cl ient is 4294967295, 1 will be assigned.
20:03 Info      Client requested ::, got 2001:e30:1402:1::d17d (IAID=2,
pref=180 0,valid=3600).
20:03 Notice    Sending REPLY on eth0/2,transID=0x273fdd, opts: 1 3 2, 0
relays.
20:03 Notice    Accepting connections. Next event in 60 second(s).
```

- **[root@pc212 ~]# ./dibbler-client run/*Command Snapshort12*/**

```
| Dibbler - a portable DHCPv6, version 0.4.0(CLIENT)
| Authors : Tomasz Mrugalski<thomson(at)klub.com.pl>,Marek
Senderski<msend(at)o2.pl>
| Licence : GNU GPL v2 or later. Developed at Gdansk University of
Technology.
| Homepage: http://klub.com.pl/dhcpv6/
2006.06.15 01:16:28 Client Notice    My pid (3755) is stored in
/var/lib/dibbler/client.pid
2006.06.15 01:16:28 Client Notice    Detected iface sit0/3, flags=128,
MAC=00:00:00:00.
2006.06.15 01:16:28 Client Notice    Detected iface eth0/2, flags=4163,
MAC=00:d0:b7:e3:d1:3e.
2006.06.15 01:16:28 Client Notice    Detected iface lo/1, flags=73,
MAC=00:00:00:00:00:00.
2006.06.15 01:16:28 Client Debug     Parsing /etc/dibbler/client.conf...
16:28 Info      Interface eth0/2 configuation has been loaded.
16:28 Info      Bind reuse enabled.
16:28 Notice    Creating control (::) socket on the lo/1 interface.
16:28 Notice    Creating socket (addr=fe80::2d0:b7ff:fee3:d13e) on the
eth0/2 interface.
16:28 Info      Socket bound to fe80::2d0:b7ff:fee3:d13e/port=546
16:28 Info      Creating SOLICIT message  on eth0 interface.
16:28 Notice    Sleeping for 1 second(s).
16:29 Info      Processing msg (SOLICIT,transID=0x321a48,opts: 1 3 8)
16:29 Notice    Sleeping for 1 second(s).
16:29 Notice    Received ADVERTISE on eth0/2,TransID=0x321a48, 4 opts: 1 3
2 7
16:29 Notice    Sleeping for 1 second(s).
16:30 Info      Processing msg (SOLICIT,transID=0x321a48,opts: 1 3 8)
16:30 Info      Creating REQUEST. Backup server list contains 1 server(s).
16:30 Notice    Sleeping for 1 second(s).
16:31 Info      Processing msg (REQUEST,transID=0x273fdd,opts: 1 3 8 2)
16:31 Notice    Sleeping for 1 second(s).
16:31 Notice    Received REPLY on eth0/2,TransID=0x273fdd, 3 opts: 1 3 2
16:31 Notice    Address 2001:e30:1402:1::d17d added to eth0/2 interface.
16:31 Notice    Sleeping for 1 second(s).
16:32 Notice    Sleeping for 1 second(s).
16:33 Notice    Sleeping for 998 second(s).
```

- Content of the file **/var/named/chroot/var/named/named.ca :/*File Snapshort 18***

```
;       This file holds the information on root name servers needed to
;       initialize cache of Internet domain name servers
;       (e.g. reference this file in the "cache  .  <file>"
;       configuration file of BIND domain name servers).
```

```
;
;       This file is made available by InterNIC
;       under anonymous FTP as
;           file                /domain/named.cache
;           on server           FTP.INTERNIC.NET
;       -OR-                    RS.INTERNIC.NET
;
;       last update:    Jan 29, 2004
;       related version of root zone:   2004012900
;
;
; formerly NS.INTERNIC.NET
;
.                        3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.      3600000      A     198.41.0.4
;
; formerly NS1.ISI.EDU
;
.                        3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.      3600000      A     192.228.79.201
;
; formerly C.PSI.NET
;
.                        3600000      NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.      3600000      A     192.33.4.12
;
; formerly TERP.UMD.EDU
;
.                        3600000      NS    D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.      3600000      A     128.8.10.90
;
; formerly NS.NASA.GOV
;
.                        3600000      NS    E.ROOT-SERVERS.NET.
E.ROOT-SERVERS.NET.      3600000      A     192.203.230.10
;
; formerly NS.ISC.ORG
;
.                        3600000      NS    F.ROOT-SERVERS.NET.
F.ROOT-SERVERS.NET.      3600000      A     192.5.5.241
;
; formerly NS.NIC.DDN.MIL
;
.                        3600000      NS    G.ROOT-SERVERS.NET.
G.ROOT-SERVERS.NET.      3600000      A     192.112.36.4
;
; formerly AOS.ARL.ARMY.MIL
;
.                        3600000      NS    H.ROOT-SERVERS.NET.
H.ROOT-SERVERS.NET.      3600000      A     128.63.2.53
;
; formerly NIC.NORDU.NET
;
.                        3600000      NS    I.ROOT-SERVERS.NET.
I.ROOT-SERVERS.NET.      3600000      A     192.36.148.17
;
; operated by VeriSign, Inc.
;
.                        3600000      NS    J.ROOT-SERVERS.NET.
J.ROOT-SERVERS.NET.      3600000      A     192.58.128.30
;
; operated by RIPE NCC
;
.                        3600000      NS    K.ROOT-SERVERS.NET.
K.ROOT-SERVERS.NET.      3600000      A     193.0.14.129
;
; operated by ICANN
;
.                        3600000      NS    L.ROOT-SERVERS.NET.
L.ROOT-SERVERS.NET.      3600000      A     198.32.64.12
;
; operated by WIDE
;
.                        3600000      NS    M.ROOT-SERVERS.NET.
M.ROOT-SERVERS.NET.      3600000      A     202.12.27.33
; End of File
```

Resources

[1].Tortonesi, M.: An overview of the IPv6 protocol.2004. [cit 2004-5-15]. Accessible from: http://www.deepspace6.net/docs/overview.html

[2].Luckie, M.: Measurement & Network Analysis. NLANR/AMP IPv6. 07 Jun 2004. [cit.2004-5-7]. Accessible from: http://amp.nlanr.net//IPv6

[3].Migration and Co-existence of IPv4 and IPv6 in Residential Networks Pekka Savola CSC/FUNET .Accessible from:http://pekka.savola.funet.fi

[4]. Discover Internet Protocol, version 6 (IPv6) (Makham V. Kumar, June 2006) Accessiblefrom:http://www.ibm.com/introductoryworks/web/library/dis-ipv6.html

[5].Writing a simple IPv6 program (Senthil Sundaram, developerWorks, September 2001): Configure an IPv6 address and port an IPv4 application to IPv6 Accessible from: http://www.ibm.com/developerworks/web/library/wa-ipv6.html

[6].The IPv6 Forum: Browse this Web site to get an idea of the latest discussion topics. Accessible from: http://www.ipv6forum.com/

[7].IPv6 home page: Visit this IPv6 site with various topics pertaining to IPv6 Accessible from: http://www.ipv6.org/

[8].IPv6 Related Specifications: Learn more about the Internet Engineering Task Force's (IETF's) Requests for Comments (RFCs) and IPv6 specifications. Accessible from: http://www.ipv6.org/specs.html

[9].Linux IPv6 HOWTO: Find information on how to configure IPv6 on Linux. Accessible from: http://www.tldp.org/HOWTO/Linux+IPv6-HOWTO/

[10].IPv6 for Microsoft Windows: Frequently Asked Questions: Find answers commonly asked questions in this FAQ about the IPv6 protocol for the Microsoft Windows family of operating systems.
Accessible from:
http://www.microsoft.com/technet/itsolutions/network/ipv6/ipv6faq.mspx

[11]The Dibbler project Accessible from: http://klub.com.pl/dhcpv6/

RFCs consulted:-4213, 2893, 2119, 2462, 3493, 3484, 3315, 19182460, 2466.