receive responses from echo servers.

Below shows how a "hello world" message is sent to `172.16.30.255` network. There are four responses from hosts on the same segment of the network that run `echod` server.

```
$ ./broadcast 172.16.30.255 "hello world"
Sent "hello world" to 172.16.30.255
Received "hello world" from 172.16.30.208
Received "hello world" from 172.16.30.226
Received "hello world" from 172.16.30.223
Received "hello world" from 172.16.30.5
```

2.3 Multicasting

Multicasting is used when a group of hosts participate in sending or receiving the same messages among the hosts. When a host sends messages, such as video and sound broadcasts, other host that are interested in receiving the messages join the group and receive the messages.

A range of IP addresses are allocated for multicasting, which is Class D of the class-full IP address between 224.0.0.0 and 239.255.255.255 or 224.0.0.0/4. The lower order 23 bits of IP address is mapped to the lower order 23 bits MAC address, and the left three octets of MAC address are always 01:00:5e. For example, an IP multicast address is 224.0.1.5 then the MAC multicast address would be 01:00:5E:00:01:05. Another example for IP multicast address 239.192.10.25 and MAC multicast address would be 01:00:5E:40:0A:19, and below shows how the bits are mapped:

```
   192       10        25
1
2631
8426 8421
1100 0000|0000 1010|0001 1001
X100 0000|0000 1010|0001 1001 Use only the lower 23 bits
 4    0  | 0    A  | 1   9
```

When multicast is applied in audio and video streaming, client programs may choose to receive audio only, or audio and video at the same time. Groupings of IP multicast addresses and transport layer port numbers called *multicast sessions* are used for such purpose.

All hosts that wish to receive multicast packets need to join a group or IP multicast address before they can receive multicast packets. The `setsockopt()` function is used together with `IPPROTO_IP`, `IP_ADD_MEMBERSHIP`, and `struct ip_mreq`. These values are passed as the second, third, and fourth arguments for joining an IP multicasting group. `struct ip_mreq` is for holding the multicast address and the interface to listen on. Appendix J shows a sample program for receiving multicast packets.

Sending multicast packets is similar to sending a UDP packet, and the sending host does not need to join any groups. The sample program in Appendix I shows how to create a socket for sending multicast packets. It is similar to creating UDP socket; however, the time-to-live for multicasting is default to 1, so this value need to be changed if multicast packets are send to remote networks. The packets outgoing interface also need to be set, as the operating system may chose a different interface each time a packet is sent if there are more than one interfaces.

Figure 2.6 shows the output generated from some senders and a receiver.

```
$mcastsend 224.0.0.3 172.16.30.5 1234      $mcastreceive 224.0.0.3 1234
172.16.30.5>Aloha                          Bound to address 0.0.0.0
$mcastsend 224.0.0.3 172.16.30.28 1234     Joined multicast address 224.0.0.3 on port 1234
172.16.30.28>Hello                         Received from 172.16.30.5 on port 1234. 12 bytes message: Aloha
$mcastsend 224.0.0.3 172.16.30.110 1234    Received from 172.16.30.28 on port 1234. 12 bytes message: Hello
172.16.30.110>Hi                           Received from 172.16.30.110 on port 1234. 12 bytes message: Hi
```

Figure 2.6: Sending and receiving multicast packets

2.4 IPv4 and IPv6 interoperability

IPv6 was implemented to replace IPv4; however, the migration from IPv4 to IPv6 is not an easy task. One way to do so is using a *dual stacks* approach, where the network nodes support both IPv4 and IPv6 are released in new installations. Gradually all the nodes could support IPv6, and eventually IPv4 is phased out. Figure 2.7 shows how the implementation takes place, `Node A` supports IPv4, `Node B` supports IPv4 and IPv6 or dual stacks, and `Node C` supports IPv6.

When the dual-stacks node receives packets from `Node C`, which sends packets using IPv6 address. `Node B` would process it as IPv6 and communicate with `Node C` using IPv6 address.

`Node A` and `Node C` cannot communicate because they are not using the same network layer protocol. However, when `Node A` sends packets to `Node B`, `Node B` accepts it and converts the IPv4 address to IPv4-mapped IPv6 address, and passes the packet to the `Transport` layer then to the `Application` layer. The layers above `Network` layer on `Node B` are not aware of the network layer implementation on the remote hosts, and both nodes communicate using IPv4.

The data link layer of the dual-stacks node differeciates the two versions of IP using its `Type` field, where the value for IPv4 is 0x0800, and IPv6 is 0x86dd.

There are two ways an IPv6 address can carry IPv4 address. The first is IPv4-compatible IPv6 address, where the higher order 32 bits of the 128 bits IPv6 address are allocated for IPv4 address. For example, an IPv4 address 10.1.1.2 is incorporated in an IPv4-compatible IPv6 address as 0:0:0:0:0:0:10.1.1.2 or ::10.1.1.2.

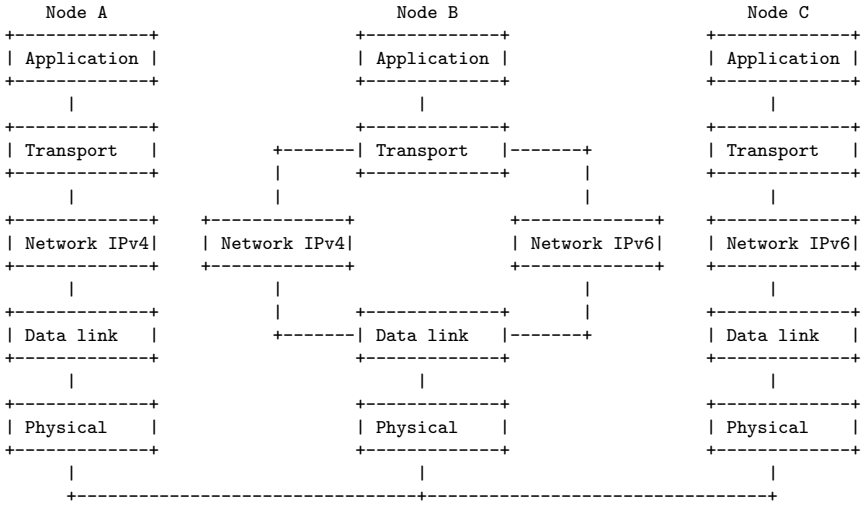

Figure 2.7: Dual stacks

The other way is IPv4-mapped IPv6 address, where the 128 bits IPv6 address are divided into three parts as shown in Figure 2.8. The first part consists of 80 bits all 0's, the second part is 16 bits of all 1's, and the last part is 32 bits for holding IPv4 address. For example, an IPv4 address 10.1.1.3 is mapped to IPv6 address using an IPv4-mapped IPv6 address would look like 0:0:0:0:0:FFFF:10.1.1.3 or ::FFFF:10.1.1.3.

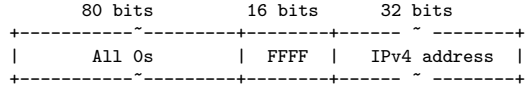

Figure 2.8: IPv4-mapped IPv6 address

2.4.1 IP Next Generation(IPv6)

IP Next Generation IPv6 is the new version of the Internet Protocol. IPv6 was implemented to enhance the capability of IPv4, and for overcoming some of the weaknesses in IPv4. Below are the changes in IPv6. [RFC 1883] p. 3.

Expanded addressing capabilities IPv4 has 32 bits of address size which is hitting its limits, so IPv6 incorporated 128 bits of address size.

Header format simplification reduce the header size to 40 bytes fixed-length. This eliminates unnecessary parsing of fields, such as checksum, and the fixed-length header improves processing time and reduces bandwidth cost.

Improved support for extensions and options to improve efficiency in handling option fields, and for introducing new options. These options cannot be supported by IPv4.

Flow labeling capability for handling special requests, such as real-time service. The flow label includes bits for differentiated service specification.

Authentication and privacy capabilities for supporting authentication, data integrity, and data confidentiality.

Provision for protocol extension where the IPv6 header supports extension to adapt new network hardware and applications without specifically declares static fields in the header.

2.4.2 IPv6 improves forwarding speed

IPv4 consists of some fields that are redundant and expensive in terms of processing time. The checksum field in IPv4 is kind of redundant, because both the upper and lower layer protocols have data checking. Checksum is an expansive process, because it needs to be recomputed on every packet due to changes in the TTL field.

Fragmentation and reassembly support in IPv4 is a very expensive process. IPv6 does not perform fragmentation and reassembly, so packets are dropped if exceed the MTU and a "packet too big" ICMP message is sent to the sender.

IPv6 does not have an `option` field, and it is taken care of by the `next header` field if additional headers are needed. Removing the option field results to a fixed 40 bytes header, and processing packets can be improved when the header size is fixed.

2.4.3 IPv6 Header Format

Version

Version is the same as IPv4. It is the first field and is a 4-bit field, for identifying the version of the IP packet.

Priority

This field is for specifying the priority of the packet. The priority is divided into two ranges, from 0 to 7 and from 8 to 15, for congestion control purposes. The first range

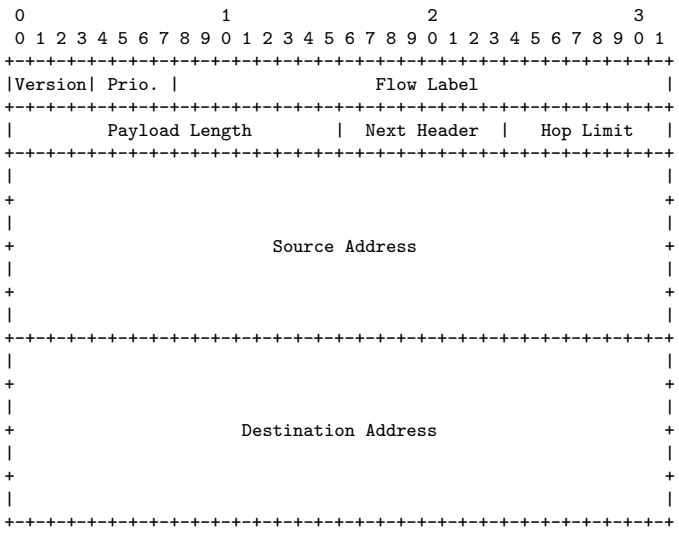

Figure 2.9: IPv6 Header Format

is for packets that can be "hold on" whenever there is congestion on the Internet, so that not too much traffics are injected into the network. The second range is for specifying that the packet is "important", and allow the packet to go through the network even though there is network congestion. The value 8 has lower priority or the packet can be discarded, and 15 has the highest priority and the packet should not be discarded.

Flow Label

This field is used by the source host to specify the type of data or the flow of data is being transmitted. This is to inform the routers that this flow of data needs special handling. For instance, a stream of audio or video data can be considered as a flow.

Payload Length

It is a 16-bit unsigned integer for specifying the length of the payload in octets.

Next Header

This field identifies the type of protocol this packet is carrying, such as TCP or UDP.

Hop Limit

This is a 8-bit unsigned integer set by the sender and it is decremented by 1 every time the packet passes through a node that forward the packet. The packet is discarded if the `Hop Limit` is decremented to 0. This is to avoid packets from circulating in the network indefinitely.

Source Address

The source address is a 128-bit field specifying the originator of the packet. Examples:

```
FEDC:BA98:7654:3210:FEDC:BA98:7654:3210

1080:0:0:0:8:800:200C:417A
```

Destination Address

The destination address is a 128-bit field specifying the address of the intended packet recipient.

Packet Size Issues

"IPv6 requires that every link in the internet have an MTU of 576 octets or greater. On any link that cannot convey a 576-octet packet in one piece, link-specific fragmentation and reassembly must be provided at a layer below IPv6." [RFC 1883] p. 26.

2.4.4 IPv4 to IPv6 transition

Replacing IPv4 by IPv6 is not an easy task, because majority of the nodes on the Internet are using IPv4. Several options are possible for the transition from IPv4 to IPv6, one of the options is to have a "flag day" or an Internet Protocol migration day. This is rather impossible, because there are too many nodes on the Internet, this involves millions of users, and network administrators.

An alternative as sugested in RFC 2893 is to deploy a *dual-stack* approach, and *tunneling*. The dual-stack approach is when releasing new Operating Systems and network equipment, these equipment are capable of running both IPv4 and IPv6. Eventually, all the nodes on the Internet are able to handle both IPv4 and IPv6, and at the end IPv6 takes over.

Tunneling is encapsulating IPv6 packets as the payload of IPv4 in routers between the tunnel. Network equipment that are IPv6 capable communicate with each other using IPv6, and a tunnel is setup in between routers that are not IPv6 aware. Figure 2.10 shows an example of tunneling.

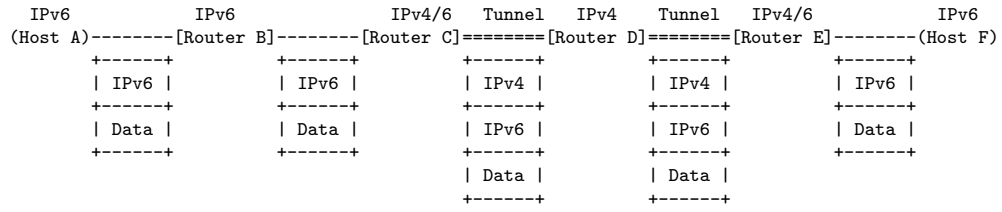

Figure 2.10: Tunneling