

Internetworking with TCP/IP

SOI ASIA Operators Workshop

Brawijaya University
28 August - 1 September 2005

Contents

1	Introduction to TCP/IP	2
1.1	TCP/IP Architecture	2
1.2	Addressing	5
1.2.1	IPv4 Addressing	5
1.2.2	IPv6 addressing	7
1.3	Address Resolution	10
1.4	Routing	11
1.4.1	Routing Architecture	11
1.4.2	Routing Table	12
1.4.3	Populating routing table	13
1.5	ICMP	14
1.6	Internet Server	15
1.7	Exercise	16
2	Routing with Zebra	20
2.1	Overview	20
2.2	OSPF Routing Protocol	22
2.3	Zebra Routing Daemon	23
2.4	Routing Sample	27
2.4.1	Configuration	28
2.4.2	Operation	30
2.5	Troubleshooting	32
2.6	Exercise	33
3	PIM-SM Multicast Routing with XORP	42
3.1	Overview	42
3.2	PIM-SM	44
3.3	Multicast Routing on FreeBSD	46
3.4	XORP for PIM-SM Multicast Routing	47
3.5	Exercise	64

Chapter 1

Introduction to TCP/IP

1.1 TCP/IP Architecture

The Internet as we know today was originated in 1969 when Advanced Research Projects Agency (ARPA) built an experimental packet switching network called *ARPAnet*. ARPAnet was then converted into an operational network in 1975, and basic TCP/IP protocols were developed after ARPAnet became operational. The term *Internet* became a common name when TCP/IP was adopted as the network protocol standard. In 1985, National Science Foundation (NSF) created NFSnet and connected it to the Internet. ARPAnet ceased to operate in 1990, and in 1995 NFSnet stopped playing a role as the Internet primary backbone network. Since then, the Internet evolved into a large collection of networks independent from the American government.

The TCP/IP protocol suite has several features that contribute to its popularity: open protocols standard, independence from specific physical network hardware, and a common addressing scheme. Protocols in data communication determine the rules of communication between nodes. TCP/IP is an open protocol standard, where the standards are developed via open meetings, and the standard documents are publicly available. Internet Engineering Task Force (IETF) is the organization responsible for developing Internet standards. Independency from physical network interface allows TCP/IP to run on various network technologies, even as these technologies evolve. TCP/IP has a common addressing scheme that allows any nodes connected to the network to communicate.

International Standard Organization developed the Open Systems Interconnect (OSI) Reference Model as the architecture reference for data communications. The OSI Reference Model consists of seven layers, numbered from 1 to 7, and each layer provides a certain functionality (Figure 1.1). When a node sends data to another node, the data is passed from Layer 7 down to Layer 1, and the receiving node passes the data from Layer 1 up to Layer 7.

The TCP/IP architecture is generally viewed as having four layers according to how TCP/IP passes data between nodes (Figure 1.2). The four layers from the top to bottom are: Application, Transport, Internet, and Network Access Layers. When a node sends data, the TCP/IP adds a header each time it passes data to the lower layer in a process called encapsulation (Figure 1.2). The reverse process is called decapsulation, i.e. the header is stripped and data is sent to the upper layer, and it happens at the receiving node. Each

Figure 1.1: OSI Reference Model

layer has protocols that are independent from protocols at other layers, and encapsulation-decapsulation processes merely prepend and strip headers without considering the data passed between layers.

Figure 1.2: TCP/IP Architecture

The Network Access Layer provides the protocols to transmit data on a network medium, and the data structure is called frame. These includes Ethernet, HDLC (High Level Data Link Control), and ATM (Asynchronous Transfer Mode). The Internet Layer defines the Internet Protocol that provides the addressing for internet hosts, and handles datagram transmission and routing between hosts. At this layer, data is transmitted in a best effort manner, i.e. a datagram is sent to another host but the Internet Layer doesn't check whether the datagram arrives at that host. The Transport Layer has two main protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP provides a reliable data delivery between two communicating end nodes, in which each node sends an

acknowledgement for the data it received. UDP only provides an unreliable data delivery, since it does not verify whether data is received at the other end of the communication. These two protocols introduce the notion of *port number* in order to correctly pass data to and from the Application Layer.

On top of the TCP/IP layer lies the Application Layer which includes all processes that use Transport Layer for network communication. There are many protocols at the Application Layer, such as Telnet for remote access, HTTP (Hypertext Transport Protocol) for World Wide Web, and SMTP (Simple Mail Transfer Protocol) for emails. An application protocol may or may not be standardized. When a protocol becomes a standard, the Internet Assigned Numbers Authority (IANA) may assign a unique port for that protocol to be used for the server processes. These ports are called "well-known ports", in the range 0-1023. For example, HTTP's port number is 80, and SMTP is 25. IANA may register application protocols port numbers for the convenience of the Internet community. The Registered Ports are in the range 1024-49151.

Problems of IPv4

The Internet has been using its protocol, IPv4, for more than a quarter of a century. The Internet saw its deployment found the tipping point in early 1990s with the popularity of World Wide Web. This fast pace development, however, creates problems for IPv4:

- Exhaustion of IPv4 addresses.
- Routing table explosion.
- Proliferation of NAT.

Exhaustion of IPv4 addresses. IPv4 address is 32 bits long, hence it can handle $2^{32} = 4.3$ billion hosts, which is less than the human population. With the current deployment pace, IPv4 address is thought to will be exhausted in 2008. Internet Registries today enforce a rather strict address allocation policy, and this policy actually extends the lifetime of IPv4.

Routing table explosion. IPv4 address allocation scheme does not allow effective routing information aggregation at the core of the Internet. As of July 2004, the number of prefixes in the Internet routing table has more than 130 thousand prefixes before aggregation and more than 95 thousand entries after aggregation. Routing table explosion burdens core routers, and may create instability problems and routing accidents.

Proliferation of NAT. New networks resort to use private IP addresses and Network Address Translation (NAT) mechanism because they cannot get enough IP address space. NAT breaks the end-to-end connectivity between hosts behind a NAT router and hosts on the Internet, and limits the use of some applications.

IPv6 Features

IPv6 fixes the IPv4 address exhaustion problem and several other problems related to IPv4. It also adds some improvements and features to the current IPv4 protocol, such as zero configuration and better security. Briefly, the features of IPv6 are:

- Larger address space
- New header format
- Efficient and hierarchical addressing and routing infrastructure
- Built-in security
- Better support for quality of service
- Extensibility

Larger address space. IPv6 has 128 bit address, supporting up to 3.4×10^{38} possible combinations. IPv6 will not have this many possible addresses, since it is designed for hierarchical subnetting and address allocation; however the total possible addresses in IPv6 are very large.

New header format. IPv6 header is only twice that of IPv4, even though it has four times the address size. This is achieved by streamlining the header, removing nonessential and optional fields in IPv4 header. Furthermore, IPv6 headers have boundaries in the multiples of 32 bits for faster processing.

Efficient and hierarchical addressing and routing infrastructure. The IPv6 address has multiple subnetting hierarchy, that allows aggregation at the core of the Internet. Address aggregation will result in an efficient routing at the Internet core, where routing tables will consist of only several thousand entries.

Built-in security. IPsec is included in the IPv6 protocol requirements. Therefore, every hosts have a standard mechanism to ensure secure communications.

Better support for quality of service. IPv6 has a Traffic Class and a Flow Label field to support QoS. Intermediate routers give traffic priority based on the content of Traffic Class field, while Flow Label allows router to identify and give a special handling to the packet.

Extensibility. Each IPv6 header has a Next Header field. This allows an IPv6 packet to have many headers.

1.2 Addressing

1.2.1 IPv4 Addressing

Notation

An IPv4 address is a 32-bit value that uniquely identifies every node connected to a TCP/IP network. An IPv4 address is usually written as four decimal numbers representing an 8-bit value separated by periods, and called the *dotted decimal notation*. Examples of IPv4 addresses are 10.39.234.121, 155.12.56.212, and 202.249.25.1.

An IPv4 address contains a *network part* and a *host part*. The network and host parts are determined by the *network mask* of the address. A network mask is a 32-bit value whose a contiguous series of MSBs are 1 and the rests are 0. The contiguous series of 1 defines the network part of the address. Examples of network masks are 255.0.0.0, 255.128.0.0, and 255.255.192.0. The network part of an address is derived by masking the address with

the network mask. For example, an IPv4 address 10.39.234.121 whose network mask is 255.255.255.0. This address is the host 121 on network 10.39.234.0.

Writing an IPv4 address with its network mask is cumbersome, thus a shorthand form is introduced. The format is *address/prefix-length*, where *prefix-length* is the number of bits in the network part of the address. The shorthand form of the above example is 10.39.234.121/24, since there are 24 bits are set to 1 in network mask 255.255.255.0.

Address class and subnet

The IPv4 address space was originally divided into several *address classes*, where an address space with a certain prefix will have a certain network mask. Table 1.1 shows the IPv4 address space and its classes. We can see from the table that there are big differences between the number of hosts that can be accommodated by class A, B, and C.

Table 1.1: IP Address Classes

Class	Prefix bits	Net. number	Rest	Net. size (host)
Class A	0	7	24	16,777,214
Class B	10	14	16	65,534
Class C	110	21	8	254
Class D (multicast)	1110			
Class E (reserved)	1111			

IPv4 address space was traditionally distributed to organizations based these classes. However, some organizations needed address space more than a class C address can provide, but much less than provided by a class B address. Also, a class A address is too much for an organization but a class B address is not enough for it. To overcome these problems, IP address space is not distributed based on the original address class, but as a block of contiguous IP addresses. This IP address assignment method increases the usable IP address space and enables route aggregation. Routing entries on the Internet now use address with address mask, and this method is called *Classless Inter-Domain Routing (CIDR)*.

An organization may distribute the IP address space within its organization with a method called *subnetting*. An organization creates several subnets by modifying the network mask of its address space. For example, 10.39.234.0/24 may be divided into four smaller subnets: 10.39.234.0/26, 10.39.234.64/26, 10.39.234.128/26, and 10.39.234.192/26. The administrator of 10.39.234.0/24 then may delegate the subnets to other administrators within the organization or to customers.

Address type

There are three types of IPv4 address: unicast, multicast, and broadcast. A unicast IPv4 address is used to directly address a node. A group of nodes may be addressed using a multicast address (Class D), and all nodes on a subnet may be addressed using the broadcast address of the subnet. Multicast and broadcast addresses may only be used as

the destination address of an IP datagram, and may not be used to address a node. Address Class A, B, and C are the unicast address spaces, and Class D is the address space dedicated for multicast.

Broadcast addresses are 255.255.255.255 and the address in a subnet whose bits in the host part are all 1. Another important address in a subnet is the *network address*, which is the address in the subnet whose bits in the host part are all 0. These two addresses are reserved on a subnet and should not be used as a host address. For example, on subnet 10.39.234.0/24, the broadcast address is 10.39.234.255 and the network address is 10.39.234.0.

1.2.2 IPv6 addressing

IPv6 addresses are 128-bit identifiers of interfaces and sets of interfaces. There are three types of IPv6 addresses:

- **Unicast** An identifier for a single interface. A packet sent to a unicast address is delivered to the interface identified by that address.
- **Anycast** An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to an anycast address is delivered to one of the interfaces identified by that address.
- **Multicast** An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to a multicast address is delivered to all interfaces identified by that address.

Notation

An IPv6 address is written using 8 groups of 16-bit block separated by a colon. For example, 2001:1D80:0000:3FC6:0000:0000:4AB7:5E91. 16-bit blocks whose value are 0 can be compressed using a double colon (::) to simplify the address notation with a limitation that there can be no more than one double colon in an address. Table 1.2 shows the correct and incorrect IPv6 address notations of the previous address example. Notation number 4 in the table is incorrect because it includes the zero in 1D80 to the double colon, therefore changes the address into 2001:01D8:0000:3FC6:0000:0000:4AB7:5E91.

Table 1.2: Simplifying the notation of 2001:1D80:0000:3FC6:0000:0000:4AB7:5E91

No.	Notation	Correct
1	2001:1D80:0:3FC6::4AB7:5E91	Yes
2	2001:1D80::3FC6:0:0:4AB7:5E91	Yes
3	2001:1D80::3FC6::4AB7:5E91	No
4	2001:1D8::3FC6:0:0:4AB7:5E91	No

IPv6 also uses prefixes to identify subnets and routes, as in IPv4 CIDR (Classless Interdomain Routing). An IPv6 prefix address is written as *ipv6-address/prefix-length*. If

the address in the previous example has a route prefix with length of 48 bits, the prefix is 2001:1D80::/48.

Address type identification

The type of an IPv6 address is identified by the high-order bits of the address, as in Table 1.3.

Table 1.3: Address type identification

Address type	Binary prefix	IPv6 notation
Unspecified	000..0 (128bits)	::/128
Loopback	000..1 (128bits)	::1/128
Multicast	11111111	FF00::/8
Link-local unicast	1111111010	FE80::/10
Site-local unicast	1111111011	FEC0::/10
Global unicast	(everything else)	

Unicast address

There are several types of unicast addresses in IPv6; for example global unicast, site-local unicast, and link-local unicast addresses. New address types may also be allocated in the future. The global unicast address of an interface may be an aggregatable global unicast address, whose format is as shown in Figure 1.3.

Figure 1.3: Aggregatable global unicast address format

Interface identifier. For all unicast addresses, except those that start with binary value 000, the IPv6 address structure consists of a 64-bit subnet prefix and a 64-bit interface identifier constructed by a Modified EUI-64 format. Interface identifiers must be unique for each interface on a subnet. Interface identifiers may be configured manually by network administrators. They may also be configured automatically using the Address Autoconfiguration mechanism of IPv6. The interface identifiers are usually taken from the interface hardware tokens, such as MAC addresses. The Modified EUI-64 format of a MAC address is constructed by complementing the second LSB of the first byte of MAC address and inserting 0xfffe between the third and fourth bytes of the MAC address. If such tokens

are not available, system administrators may configure these manually. For example, an Ethernet network interface has a MAC address 0:e0:81:20:af:c2, thus the interface identifier is 2e0:81ff:fe20:afc2.

Local-use unicast addresses. There are two types of local-use unicast addresses. Link-local addresses are for use on a single link, and the prefix identifier is FE80::/10 and the next 54-bits are all zeros. Site-local addresses are for use in a single site. They serve as private addresses to networks that do not connect to the Internet. The prefix for site local addresses is FEC0::/10 with the next 54-bits are for subnet identifier assignments.

Unspecified address. The address 0:0:0:0:0:0:0 is called the unspecified address. This address indicates the absence of an address and may not be assigned to any node.

Loopback address. The unicast address 0:0:0:0:0:0:1 is called the loopback address. It is used by a node to send packets only to itself, and must never be assigned to any physical interface.

IPv6 addresses with embedded IPv4 addresses. IPv6 nodes uses these addresses for transition from IPv4. These addresses have IPv4 address in its low-order 32-bits and have prefix 000..0 (80bits). There are two types of such address: 1. IPv4-compatible IPv6 address; and 2. IPv4-mapped IPv6 address.

Multicast address

An IPv6 multicast address is an identifier for a group of interfaces. IPv6 does not recognize broadcast addresses, and all broadcast address functionalities in IPv4 have been replaced by multicast addresses in IPv6. An interface may belong to any number of multicast groups. There are pre-defined multicast addresses, for example reserved addresses, All Node addresses, and Solicited-Node-Addresses, that serve for particular purposes. Figure 1.4 shows the format of an IPv6 multicast address.

Figure 1.4: IPv6 multicast address format

Anycast address

An IPv6 anycast address is an address that is assigned to more than one interface. Packets destined to an anycast address are routed to the nearest interface having the anycast address. At this moment, anycast addresses may only be assigned to IPv6 routers. An IPv6 router must recognize a subnet-router anycast address for each subnet to which they have interfaces.

A node's addresses

An IPv6 node is required to recognize the following addresses in identifying itself:

- Its required Link-Local Address for each interface.
- Any additional Unicast and Anycast Addresses that have been configured for the node's interfaces (manually or automatically).
- The loopback address.
- The All-Nodes Multicast Addresses.
- The Solicited-Node Multicast Address for each of its unicast and anycast addresses.
- Multicast Addresses of all other groups to which the node belongs.

An IPv6 router must recognize the below addresses in addition to the above addresses:

- The Subnet-Router Anycast Addresses for all interfaces for which it is configured to act as a router.
- All other Anycast Addresses with which the router has been configured.
- The All-Routers Multicast Addresses.

1.3 Address Resolution

When a host sends an IP datagram to a destination, it has to know the physical (or Layer 2) address of the destination or the gateway to the destination to be used as the destination address of the frame containing the IP datagram. Each network interface has its own address, usually preset from the factory, and there has to be a mechanism to map an IP address to the physical address of the network interface. This mechanism is called *address resolution*.

Address resolution for IPv4 uses Address Resolution Protocol (ARP), and it works as follows. Suppose a host *A* with IP address 10.39.234.121 is going to send an IP datagram to *B* (10.39.234.1) on the local network, but *A* doesn't know the physical address of *B*. First *A* sends an ARP Request to the local network using the Ethernet broadcast address as the destination address of the Ethernet frame, saying **arp who is 10.39.234.1 tell 10.39.234.121**. This ARP Request is received by all nodes on the local network. Receiving this message, *B* sends an ARP Reply message to *A* using the physical address of *A* as destination of the Ethernet frame, saying **arp 10.39.234.1 is 00:02:b3:ec:6c:d4**. *A* then stores the IP address – physical address mapping of *B* in its *ARP cache* table until the entry for *B* expires in a certain amount of time.

IPv6 uses Neighbor Discovery Protocol (NDP) for address resolution. NDP is used not only to determine the Layer 2 addresses of nodes on the same link, but also to find the neighboring routers and to keep track of which neighbors are reachable and which are not. An IPv6 node sends an Neighbor Solicitation message to all-nodes multicast address FF02::1 to request the physical address of the node in question. All nodes on the local link

receive this message, and the solicited node replies with a Neighbor Advertisement message to the soliciting node. The soliciting node stores the mapping in an NDP cache for some time.

Below is an example of ARP and NDP caches.

```
> arp -an
? (10.39.234.1) at 00:02:b3:ec:6c:d4 on fxp0 [ethernet]
? (10.39.234.121) at 00:0a:79:33:98:59 on fxp0 [ethernet]

> ndp -an
Neighbor                               Linklayer Address  Netif Expire      St Flgs Prbs
3ffe:1:2:3:202:b3ff:feec:6cd4          0:2:b3:ec:6c:d4   fxp0 16h56m31s S  R
3ffe:1:2:3:2d0:b7ff:fe9e:e5d2          0:d0:b7:9e:e5:d2   fxp0 permanent R
fe80::202:b3ff:feec:6cd4%fxp0         0:2:b3:ec:6c:d4   fxp0 16h56m26s S  R
fe80::2d0:b7ff:fe9e:e5d2%fxp0         0:d0:b7:9e:e5:d2   fxp0 permanent R
fe80::1%lo0                             (incomplete)      lo0 permanent R
```

1.4 Routing

1.4.1 Routing Architecture

TCP/IP has the routing feature that enables IP datagrams to be sent across many links to reach the destination. The routing functionality is called *IP Forwarding*, and it is performed in the Internet Layer of the nodes acting as routers. Figure 1.5 illustrates IP Forwarding. When host *A* sends IP packets to *C*, *A* sends the packets to the interface B_1 of *B*, which is the local router on the network. Host *B* receives the packets and consults its *routing table*, and based on the routing table, *B* knows that it has to forward the packets from *A* via its B_2 interface to reach *C*. A router reduces the *Hop-Limit* (or *Time-to-Live* for IPv4) value of an IPv6 (or IPv4) packet when performing IP Forwarding. A packet may be forwarded as long as the resulting Hop Limit, or TTL, value of the packet is not 0.

Figure 1.5: Illustration IP Forwarding

Routers have to have the correct routing information in order to forward packets so the packets reach their destinations in the most efficient way. Routers on the Internet exchange routing information using routing protocols in a hierarchical manner. On the top hierarchy, the Internet consists of Autonomous Systems that exchange routing information called *reachability information*. An Autonomous System (AS) is a collection of networks and routers with a single routing policy, and it is usually controlled by a single administrative organization. The routing protocol used by Autonomous Systems is Border Gateway

Protocol (BGP). Within an Autonomous System, routers usually use only a single Interior Gateway Protocol, such as Open Shortest Path First (OSPF), even though there are ASes that use more than one IGPs.

1.4.2 Routing Table

Each node uses its routing table to decide where to send IP packets. For mosts hosts, this decision is simple:

- if the destination is on the local network, deliver the IP packets directly to the destination hosts.
- otherwise, send the IP packets to a local router.

Routers usually have more complete routing tables compared to those of hosts that they build based on the routing information exchange.

A routing table is a list of routing entries, where each routing entry contains:

- destination address, and
- next-hop gateway to the destination.

A node performs a table lookup on the routing table to find out where to send a packet to reach the destination address of the packet.

On a FreeBSD system, the routing table can be displayed by issuing `netstat -nr` command. Below is an example of the minimum routing table of a host with the default route.

```
> netstat -nr
Routing tables
```

```
Internet:
```

Destination	Gateway	Flags	Refs	Use	Netif	Expire
default	10.39.234.1	UGSc	11	1791	fxp0	
127.0.0.1	127.0.0.1	UH	1	970	lo0	
10.39.234	link#1	UC	2	0	fxp0	
10.39.234.1	00:02:b3:ec:6c:d4	UHLW	12	0	fxp0	1199
10.39.234.121	127.0.0.1	UGHS	0	1463	lo0	

```
Internet6:
```

Destination	Gateway	Flags	Netif	Expire
::/96	::1	UGRSc	lo0	=>
default	fe80::202:b3ff:feec:6cd4%fxp0	UGc	fxp0	
::1	::1	UH	lo0	
::ffff:0.0.0.0/96	::1	UGRSc	lo0	
3ffe:1:2:3::/64	link#1	UC	fxp0	
3ffe:1:2:3:2d0:b7ff:fe9e:e5d2	00:d0:b7:9e:e5:d2	UHL	lo0	
fe80::/10	::1	UGRSc	lo0	
fe80::%fxp0/64	link#1	UC	fxp0	
fe80::202:b3ff:feec:6cd4%fxp0	00:02:b3:ec:6c:d4	UHLW	fxp0	
fe80::2d0:b7ff:fe9e:e5d2%fxp0	00:d0:b7:9e:e5:d2	UHL	lo0	

fe80::%1o0/64	fe80::1%1o0	Uc	1o0
fe80::1%1o0	link#4	UHL	1o0
ff01::/32	::1	U	1o0
ff02::/16	::1	UGRS	1o0
ff02::%fxp0/32	link#1	UC	fxp0
ff02::%1o0/32	::1	UC	1o0

The routing table above shows the entries for both IPv4 (Internet), and IPv6 (Internet6). Notice that the gateway address of default route is the IP address of the default gateway, while the gateway address of the default gateway is the physical address of the default gateway. This is because a host needs to know the physical address of nodes on the same link; as will be explained in the next section.

Besides the destination address and the gateway to the destination, the routing table also includes other information for the routing entries. More information can be found from the manual page of netstat.

The above routing table for IPv4 consists of entries to:

- localhost address 127.0.0.1
- IP addresses of each interface
- Network IP addresses of each interface
- default route
- IP address of default router

The entries for IPv6 routing table are as above, plus route entries to multicast addresses.

1.4.3 Populating routing table

Routing table for a host may be as simple as above, but a router should have a routing table that allows packet to be forwarded toward their destinations. A routing table can be populated statically and dynamically. Network administrators may add or delete routing entries on routers manually after considering the network topology. This is called *static routing*, and it is prone to errors and not scalable. Furthermore, static routing doesn't respond well to network changes. When a link goes down, for example, network administrators should change the routing tables of all routers on the network.

The command to manually manipulate routing table on FreeBSD is the `route` command. Network administrators can do the following:

1. add a routing entry


```
route add -inet6 2002:e000:: -prefixlen 48 fe80::202:1ff:fe02:34:56%fxp0
route add 10.20.30.0/24 10.2.3.4
```
2. delete a routing entry


```
route delete -inet6 2002:e000:: -prefixlen 48
route delete 10.20.30.0/24
```
3. change a routing entry


```
route change -inet6 2002:e000:: -prefixlen 48 fe80::202:1ff:fe02:34:56%fxp0
route change 10.20.30.0/24 10.2.3.4
```

4. delete all routing entries
`route -n flush`

Dynamic routing remove the burden of populating routing tables from network administrators to routing protocols. Network administrators only have to configure routers to run routing protocols. Routers on network use routing protocols to exchange routing information among them, and each router calculates the best next-hop gateway toward each destination based on the exchanged routing information. When the network changes, e.g. a link or a router goes down, the information about this change is propagated throughout the network and all routers make necessary changes to their routing tables. Examples of routing protocols are: OSPF, RIP, and BGP. We will discuss OSPF in details in the next chapter.

1.5 ICMP

TCP/IP uses Internet Control Message Protocol (ICMP) to provide information and errors about the network and hosts. For example, ICMP gives reachability information of a host, or whether there is a firewall to a host. ICMPv6, which is the ICMP for IPv6, includes new functions that are not incorporated in ICMPv4. For example, ICMPv6 includes the multicast group membership protocol, named Multicast Listener Discovery (MLD). This function is handled by Internet Group Membership Protocol (IGMP) in IPv4. NDP, which handles address resolutions in IPv6, is a type of ICMPv6 message, instead of a separate message type such as ARP.

Two well-known applications that use ICMP are ping and traceroute. Ping uses informational ICMP messages: ICMP Echo Request and ICMP Echo Reply. A host *A* pings another host *B* by sending ICMP Echo Request messages to *B*. For each ICMP Echo Request message received, *B* sends a ICMP Echo Reply message back to *A*.

Traceroute is an application that shows the route taken to reach a destination by making use of the ICMP Time Exceeded and ICMP Destination Unreachable error messages. An example of traceroute results is

```
> traceroute6 -n www.kame.net
traceroute6 to www.kame.net (2001:200:0:8002:203:47ff:fea5:3085)
from 2001:200:0:8801:2d0:b7ff:fe9e:e5d2, 64 hops max, 12 byte packets
 1 2001:200:0:8801:202:b3ff:feec:6cd4 0.259 ms 0.169 ms 0.164 ms
 2 2001:200:0:1001:201:64ff:fea3:ec55 0.297 ms * 0.330 ms
 3 2001:200:0:1c04::1000:2000 0.919 ms 0.850 ms 0.893 ms
 4 2001:200:0:4819::2000:1 2.614 ms 2.320 ms 2.304 ms
 5 2001:200:0:8002:203:47ff:fea5:3085 2.324 ms 2.281 ms 1.891 ms
```

Traceroute sends UDP packets with increasing Hop Limit (HL) and destination port number values until the packets reach the destination. A router issues an ICMP Time Exceeded message when the HL of the packet being forwarded reaches 0 before arriving at the destination. Lines 1 to 4 in the above example show the IP addresses of routers en route to the destination. When a packet arrives at the destination, the destination host issues an ICMP Destination Unreachable message because the destination host doesn't open the port, thus line 5.

1.6 Internet Server

An application on a host exchanges data with an application on another host using TCP or UDP as the transport protocol. Data exchange between two hosts uses the server-client model. In this model, an application acts as a server, i.e. listening on a port, and a client application initiates data exchange by creating a connection to that port. On a FreeBSD system, active Internet connections, including server applications can be displayed using `netstat -na` command.

```
> netstat -na -f inet
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        (state)
tcp4   0      0 *.3306                 *.*                    LISTEN
tcp46  0      0 *.80                   *.*                    LISTEN
tcp4   0      0 *.587                  *.*                    LISTEN
tcp46  0      0 *.25                   *.*                    LISTEN
tcp4   0      0 *.25                   *.*                    LISTEN
tcp4   0      0 *.22                   *.*                    LISTEN
tcp46  0      0 *.22                   *.*                    LISTEN
udp4   0      0 *.514                  *.*                    LISTEN
udp4   0      0 *.68                   *.*                    LISTEN

> netstat -na -f inet6
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        (state)
tcp46  0      0 *.80                   *.*                    LISTEN
tcp46  0      0 *.25                   *.*                    LISTEN
tcp46  0      0 *.22                   *.*                    LISTEN
udp6   0      0 *.514                  *.*                    LISTEN
```

Each line in the output `netstat -na` has the following fields:

1. Protocol. For example: `tcp4` is TCP IPv4.
2. Amount of data in receiving queue.
3. Amount of data in sending queue.
4. Local address and port in format *ipaddress.portnumber*. For example: `*.*` means any IP address and any port.
5. Other hosts' address and port.
6. State of the connection, for TCP.

The above results show several Internet servers running on the FreeBSD system, such as SSH, SMTP, and HTTP. TCP is a stateful, connection oriented protocol, and the above results show the state of the TCP connection. TCP servers are in the LISTEN state, which means that these servers are ready to receive connections from clients. There is no state displayed for UDP servers because UDP is a connectionless transport protocol.

1.7 Exercise

Ex. 1: IP Addressing

1. Write IPv6 addresses in long and shorthand notation.

No.	Short notation	Long notation
1	3ffe::	
2	2001:d30::1234:abcd	
3		2001:d1:00a2:0:0:698a:fc22:563b
4		2001:0:40c0:0:0:0:00a0:fa32

2. Write IP addresses in IP-address/netmask and IP-address/prefix-length notation.

No.	IP-address/prefix-length	IP-address/netmask
1	10.1.1.1/28	
2		202.249.193.1/255.255.224.0
3		172.16.2.5/255.255.255.248
4	114.5.89.5/18	

3. Write MAC address and the corresponding interface identifier based on modified-EUI format.

No.	MAC address	Interface identifier
1	00:60:3e:46:e8:d9	
2		2d0:b7ff:fe2c:6914
3		230:48ff:fe71:f58e
4	00:e0:b7:2c:22:4f	

4. Write the lowest and the highest IP addresses in the address space

No.	Prefix	Lowest	Highest
1	192.168.0.0/18		
2	2001:d10:00a2::/48		
3	3ffe:2c0::/35		

Ex. 2: Enabling IPv6

1. Log on as `root`, edit `/etc/rc.conf`, and add the below line to the file `ipv6_enable="YES"`
2. Reboot your machine.
3. Log on as `root`.
4. At the command prompt, type:
`ifconfig -a`
You should see that your machines' interfaces have IPv6 addresses, e.g. the ones starting with `inet6`.
5. Write the interfaces addresses below

Ethernet iface	MAC address	IPv6 address

Ex. 3: Local neighbors

1. At the command prompt, type:
`ndp -an`
You should see a NDP cache table with 7 columns:

Column	Description
Neighbor	IPv6 address of neighbor
Linklayer Address	link layer address of neighbor
Netif	the network interface toward neighbor
Expire	expire time for cache entry
St	neighbor cache state; the possible states are N : no state W : wait to delete I : incomplete R : reachable S : stale D : delay P : probe
Flgs	neighbor flags; R: router; P: proxy
Prbs	num. of sent Neighbor Solicitation messages

2. Write the displayed NDP cache below.

Neighbor	Linklayer Addr.	Netif	Expire	St	Flgs	Prbs

Ex. 4: Ping

- At the command prompt, type:
`ndp -cn`
- followed by:
`ndp -an`
You should see a NDP cache table containing only your host entries.
- Run `ping6` to know your neighbor. For example, to know the neighbors of your host on the Ethernet interface named `r10`, type:
`ping6 -c 5 ff02::1%r10`
You should see ICMPv6 echo replies from neighbors, if there are any.
- Check the NDP cache again. You should see the entries of your neighbors, if there are any.
- Get to know routers on `r10` by typing:
`ping6 -c 5 ff02::2%r10`
- Ping to several hosts on your neighbor.

Ex. 5: Routing table and traceroute

- At the command prompt, type:
`netstat -nr -f inet6`
You should see the IPv6 routing table.
- Check the path to another (random) host, for example to `2001:d30::`
`traceroute6 -n 2001:d30::`
What are the results? Why such results appear?

Ex. 6: tcpdump

In this exercise you will practice how to watch packets seen by a network interface.

- The instructor sends IPv6 traffic to the network.
- Log on as `root`.

3. At the command prompt, type:
`tcpdump -n 'ip6'`
What are the packets you see on your console?
4. Stop tcpdump using `Ctrl+C`.
5. Now run
`tcpdump -vvn 'ip6'`
What are the differences between the previous command?

Chapter 2

Routing with Zebra

2.1 Overview

Routing in IPv6 and IPv4 works basically the same. Two differences between them are:

1. An IPv6 router must not fragment IP packets, while an IPv4 router may fragment packets.
2. An IPv6 router must advertise itself to its attached links that it can route IPv6 packets, while in IPv4, a router doesn't have to.

An IPv6 router advertises itself using Router Advertisement messages that contain several information, such as Default Router Preference, Router Lifetime, etc. It also may advertise optional information, e.g. MTU (Maximum Transmission Unit), Prefix Information.

In this chapter we discuss the how to install and operate a FreeBSD-based IPv4 and IPv6 routers.

The steps to build an IPv4 router are:

1. Enable IP forwarding.
2. Assign addresses to the interfaces.
3. Populate routing table statically and/or using routing protocols.

For IPv6, the steps are:

1. Enable IPv6 forwarding.
2. Assign site-local and/or global addresses to the interfaces.
3. Activate Router Advertisement.
4. Populate routing table statically and/or using routing protocols.

The basic configurations are added to the `/etc/rc.conf` file.

Enabling packet forwarding is the first step to build a router. The configuration lines are:

```
gateway_enable="YES"
ipv6_enable="YES"
ipv6_gateway_enable="YES"
```

The next step is assigning IPv4 addresses and site-local and/or global IPv6 addresses to a router interfaces. Use this command to assign an IPv4 address to an interface:

```
ifconfig_fxp0="inet 10.20.30.40 netmask 255.255.255.192"
```

You can assign an IPv6 address using one of these methods:

1. Assign the first 64 bits of an IPv6 address. The interface ID part of the address will be calculated automatically. For example for interface fxp0.

```
ipv6_prefix_fxp0="fec0:0000:0000:0001 fec0:0000:0000:0002"
```

2. Assign the whole address. If you assign the whole address, it is better to assign a unique interface identifier for each router interface in a site.

```
ipv6_ifconfig_fxp0="fec0:0:0:5::1 prefixlen 64"
```

An IPv6 router must send Router Advertisement messages to its link. You can enable this by activating the Router Advertisement daemon using the below configuration line.

```
rtadvd_enable="YES"
```

You can limit the Router Advertisement to certain links, e.g. only to the downstream links, using the following configuration.

```
rtadvd_interfaces="fxp1"
```

You may want to populate the routing table statically. Usually you should add a default route to the routing table. The following configuration sets the default IPv4 and IPv6 routes to a router.

```
defaultrouter="10.20.30.1"
ipv6_defaultrouter="fe80::207:e9ff:fe05:ba6f%fxp0"
```

A site is better to use routing protocols to populate the routing table. The simplest routing protocol for IPv4 is RIP, and for IPv6 is RIPng, which can be activated using the following configurations.

```
router_enable="YES"
router="routed"
ipv6_router_enable="YES"
ipv6_router="/usr/sbin/route6d"
```

These are the basic steps to build an router. Next, we will explain a better routing protocol, i.e. OSPF, and how to operate the protocol using zebra routing protocol package.

2.2 OSPF Routing Protocol

OSPF is a link-state routing protocol that operates between routers in a single Autonomous System. This routing protocol was designed to address the limitations of RIP in the supported network size. Several advantages of OSPF are: scalability, full subnetting support, and TOS routing.

OSPF works as follows. First, each OSPF router is given a unique 32-bit identifier for sending OSPF messages and for creating OSPF network topology for calculations. When an OSPF router goes up, it sends a Hello packet to each network interface that is part of the OSPF network. Routers send and receive Hello packets on a link to discover and maintain neighbor relationship with other routers on the link. A router sends a Hello packet periodically every HelloInterval. If a router doesn't hear a Hello packet from another router for RouterDeadInterval period, the router considers that the other router is dead.

A router will attempt to form adjacencies with some of the neighbors. On broadcast and Non-broadcast Multiple Access (NBMA) link, routers elect a Designated Router (DR) and a Backup Designated Router (BR). Routers on a broadcast and NBMA link should form adjacencies with these routers. If a link is a point-to-point link, the two routers on the link always form adjacencies with each others. Each pair of adjacent routers synchronize their Link-state databases, where each entry in the database basically states who are connected to who. Each router builds a picture of the network using its Link-state database, and calculates the shortest path to reach all subnets using the Dijkstra Shortest Path First algorithm. The results will create a forwarding table for the router. Zebra OSPF routers uses Database Description and Link State Request Packets in forming adjacencies. First, a router describes its Link-state database by sending a series of Database Description packets containing Link State Advertisement (LSA) to its neighbor. When a router sees that its neighbor has a more recent LSA, it sends a Link State Request packet to that neighbor. The neighbor will give the requested LSA using Link State Update packets, and a router will acknowledge the update by sending Link State Ack packets. Neighboring routers are fully adjacent after their databases are synchronized. LSAs are exchanged between routers within a network hop-by-hop until all routers have the same LSAs. This process is called database flooding.

Here we summarize the five types of OSPF packets:

1. Hello To discover/maintain neighbors
2. Database Description To summarize database contents
3. Link State Request To download database
4. Link State Update To update database
5. Link State Ack To acknowledge database flooding

OSPF allows contiguous networks to be grouped together to form areas. Splitting an AS into areas is useful when there are many routers in the AS. A rule of thumb is to limit the number of routers in an area to be no more than around 50 routers. When an AS is splitted into areas, each area has its own separate link-state database. LSAs are flooded only within an area, therefore routers in an area do not know the detailed network topology of other areas. A router may be connected to multiple areas. In this case, the router must have the same number of link-state database as the areas it is connected to. These routers are called Area Border Routers. For example, a router has an interface is in Area 0, while

another is in Area 1. This router has two link-state databases. OSPF backbone is the special OSPF Area. It must exist in a network, and other areas must be connected to the OSPF backbone.

2.3 Zebra Routing Daemon

Zebra (<http://www.zebra.org>) is a free routing software distributed under GNU General Public License. Zebra runs on several platforms, including FreeBSD. It supports IPv4 and IPv6, and several routing protocols: RIP, OSPFv2, BGP4+, RIPng, and OSPFv3. Zebra consists of routing daemons specific for each protocol and Zebra the kernel routing manager. Zebra the kernel routing manager must be running for the operation of a router. Each Zebra routing daemon (called Zebra beast) runs independently from other daemons, so when we want to run OSPFv3, for example, we only need to run `zebra` daemon and `ospf6d` daemon.

Zebra user interface is a command line interface (CLI). The commands are similar to those of Cisco, so people who are familiar with Cisco can easily configure Zebra. We access a Zebra beast CLI by accessing a certain TCP port of Zebra interface: `telnet localhost <port>`. The ports used by Zebra beast are shown in Table 2.1. These ports have been added to the Well Known Port Numbers, thus we can access the port not only by the port number, but also by the port name.

Table 2.1: Ports of Zebra beast CLI

Port name	Port number
zebrasrv	2600
zebra	2601
ripd	2602
ripngd	2603
ospfd	2604
bgpd	2605
ospf6d	2606

Each Zebra beast stores its configuration in a file named according to the beast filename. For example, `ospfd.conf` is the configuration file for `ospfd`. The default directory for the configuration files is `/usr/local/etc`.

Configuring Zebra

Below is a sample of `zebra.conf` file with line numbers. This configuration is basic, but it is enough for configuring Zebra to work for IPv4 and IPv6.

```

1 !
2 hostname Router
3 password 8 bJ0xh87QLiLbI
4 log syslog
5 service password-encryption
6 !

```

```

7 interface fxp0
8   description Ethernet to Upstream
9   ipv6 nd suppress-ra
10 !
11 interface fxp1
12   description Another Ethernet
13   ipv6 nd suppress-ra
14 !
15 ip route 0.0.0.0/0 1.2.3.4
16 ipv6 route ::/0 fe80::212:34ff:fe56:789a fxp0
17 !
18 access-list vty-access permit 127.0.0.1/32
19 access-list vty-access deny any
20 !
21 ipv6 access-list vty-access permit ::1/128
22 ipv6 access-list vty-access deny any
23 !
24 line vty
25   access-class vty-access
26   ipv6 access-class vty-access
27   exec-timeout 0 0
28 !

```

We now explain the above configuration file. Line 1 starts with "!", which is just a comment line. Line 2 determines the hostname displayed when we access Zebra CLI. Line 3 is the password to access Zebra CLI. This line shows the encrypted password caused by Line 5. The password must not be encrypted when creating the configuration file for the first time. You can do this by removing Line 5. Line 4 means that the log of Zebra will be send to syslog.

Lines 7–13 are for configuring interfaces. Here we have two interfaces, fxp0 and fxp1. An interface should have a description for clarity. The Router Advertisement is suppressed by `ipv6 nd suppress-ra` because we use `rtadvd` for this purpose.

Lines 15–16 are static route commands for the default prefix (0.0.0.0/0 for IPv4 and ::/0 for IPv6). The routing table entry for the default prefix is usually configured in the `/etc/rc.conf` file. The next-hop for IPv4 default route is 1.2.3.4 For IPv6, the next-hop for default route is fe80::212:34ff:fe56:789a on the fxp0 interface. Remember that the next-hop for IPv6 should be a link-local address.

Lines 18–27 are to limit access to Zebra CLI only from the router itself. This is an approach to secure access to the CLI.

Configuring OSPF

A simple OSPF configuration using `Ospfd` is shown below. This is a configuration of an OSPF router having two interfaces and located in the backbone area. By default, this file name is `/usr/local/etc/ospfd.conf`.

```
1 !
2 hostname ospfrouter
3 password zebra
4 enable password zebra
5 log syslog
6 !
7 interface fxp0
8 ip ospf cost 50
9 ip ospf hello-interval 10
10 ip ospf dead-interval 40
11 ip ospf priority 100
12 !
13 interface fxp1
14 !
15 router ospf
16 router-id 1.2.3.4
17 network 10.1.0.0/16 area 0.0.0.0
18 network 10.2.1.0/24 area 0.0.0.0
19 !
20 access-list vty-access permit 127.0.0.1/32
21 access-list vty-access deny any
22 !
23 ipv6 access-list vty-access permit ::1/128
24 ipv6 access-list vty-access deny any
25 !
26 line vty
27 access-class vty-access
28 ipv6 access-class vty-access
29 exec-timeout 0 0
30 !
```

Lines 2, 3, and 5 contain commands that are already shown in the sample Zebra configuration. Line 4 shows the password to enter the privilege mode, which was left out in the sample Zebra configuration. Both passwords on lines 3 and 4 are shown in plain text because the password encryption is not active.

Lines 7–13 show the configuration for two interfaces. Interface `fxp0` has several configuration lines: `cost`, `hello-interval`, `dead-interval`, and `priority`. The cost to use this interface is 50. The time interval between sending Hello packets is 10 seconds, and if other routers do not hear a Hello packet from this router in 40 seconds, this router is assumed dead. These intervals must be same for all routers on a link. The priority for `fxp0` to become a Designated Router (DR) is 100; router with the highest priority will be elected as the DR. A router with priority 0 will never be a DR. Interface `fxp1` doesn't have configuration lines, thus this interface will use the default configuration for an interface. The configuration commands for an OSPF interface are shown in Table 2.2.

Lines 15–18 define the OSPF configuration for the router. Line 16 defines the router ID for the OSPF process. The OSPF process will use the largest IP address on its interface if `router-id` is not defined. Lines 17 and 18 enable OSPF routing protocol on all interfaces that fall within the defined network prefixes, and the interfaces are in the backbone area. For example, if this router has an interface whose IP address is 10.3.1.1, then the interface will not run OSPF routing protocol.

Table 2.2: OSPF Interface Configuration

Configuration command	Description
authentication	Enable authentication on this interface
authentication-key	Authentication password (key)
cost	Interface cost
dead-interval	Interval after which a neighbor is declared dead
hello-interval	Time between HELLO packets
message-digest-key	Message digest authentication password (key)
network	Network type
priority	Router priority to be DR
retransmit-interval	Time between retransmitting lost link state advertisements
transmit-delay	Link state transmit delay

Lines 20–29 are used to limit access to Ospf6d CLI only from this router.

Configuring OSPFv3

A basic OSPFv3 configuration using Ospf6d is shown below. This is a configuration of an OSPF router having two interfaces and located in the backbone area. By default, this file name is `/usr/local/etc/ospf6d.conf`.

```

1 !
2 hostname ospf6router
3 password zebra
4 enable password zebra
5 log syslog
6 !
7 interface fxp0
8  ipv6 ospf6 cost 50
9  ipv6 ospf6 hello-interval 10
10 ipv6 ospf6 dead-interval 40
11 ipv6 ospf6 priority 10
12 !
13 interface fxp1
14 !
15 router ospf6
16  router-id 0.1.2.3
17  interface fxp0 area 0.0.0.0
18  interface fxp1 area 0.0.0.0
19 !
20 access-list vty-access permit 127.0.0.1/32
21 access-list vty-access deny any
22 !
23 ipv6 access-list vty-access permit ::1/128
24 ipv6 access-list vty-access deny any
25 !

```

```
26 line vty
27  access-class vty-access
28  ipv6 access-class vty-access
29  exec-timeout 0 0
30 !
```

Line 2, 3, and 5 contain commands that are already shown in the sample Zebra configuration. Line 4 shows the password to enter the configuration mode, which was left out in the sample Zebra configuration. Both passwords on Line 3 and 4 are shown in plain text because the password encryption is not active.

Line 7–11 are the `fxp0` interface configuration, which is same as the OSPF example. Line 13 shows the the simplest form of interface configuration for OSPFv3. Without any other parameter, `fxp1` uses the default configuration. Some of the defaults are: `hello-interval 10`, `dead-interval 40`, `cost 50`, and `priority 1`. Other parameters and the default values are available on the `Ospf6d` documentation.

Line 15–18 are the OSPv3 routing configuration. Line 15 states that this router runs OSPFv3. The router ID in line 16 is a 32 bit number written in dotted-decimal notation. The router ID must be defined in the configuration, and the value must be unique within an AS. Line 17–18 state that interface `fxp0` and `fxp1` are active, and they are in the backbone area (area ID 0.0.0.0).

The rest of the lines are access control lines, which is similar to the ones in Zebra, and `Ospf6d` configurations.

2.4 Routing Sample

This section gives a sample routing design, configuration, and operation using Zebra routing daemon package for the network topology shown in Figure 2.1. This network has five routers, *R1–R4* and *RGW*, running FreeBSD; and each shown with the network interface names. The gateway to the Internet is *RGW*. Table 2.3 shows the addresses for each interface.

Figure 2.1: Sample Network Topology

Table 2.3: Interface Addresses for Figure 2.1

Interface	MAC address	IPv6 address	IPv4 address
RGW-fxp0	0:e0:81:9:9:90	3ffe:1:2:a::9	10.1.1.9
R1-fxp0	0:e0:81:1:1:10	3ffe:1:2:a::1	10.1.1.1
R1-fxp1	0:e0:81:1:1:20	3ffe:1:2:b::1	10.1.2.1
R2-fxp0	0:e0:81:2:2:10	3ffe:1:2:b::2	10.1.2.2
R2-fxp1	0:e0:81:2:2:20	3ffe:1:2:c::2	10.1.3.2
R3-fxp0	0:e0:81:3:3:10	3ffe:1:2:c::3	10.1.3.3
R3-fxp1	0:e0:81:3:3:20	3ffe:1:2:d::3	10.1.4.3
R4-fxp0	0:e0:81:4:4:10	3ffe:1:2:c::4	10.1.3.4
R4-fxp1	0:e0:81:4:4:20	3ffe:1:2:e::4	10.1.5.4

We design the routing as follows:

1. Router advertisement is handled by `rtadvd`.
2. Default gateway is configured using Zebra.
3. OSPF and OSPFv3 networks only consist of backbone areas.
4. Router IDs for OSPF and OSPFv3 are $N.N.N.N$ for RN , and $N = 9$ for RGW .

2.4.1 Configuration

In this section we give examples of Zebra, `Ospfd`, and `Ospf6d` configuration files on several routers to show a general idea on how to configure them based on a network design.

zebra.conf on R1

```
!
hostname Router1
password zebra-R1
enable password zebra-enable
log syslog
!
interface fxp0
  description Ethernet to upstream
  ipv6 nd suppress-ra
!
interface fxp1
  description Downstream
  ipv6 nd suppress-ra
!
ip route 0.0.0.0/0 10.1.1.9
ipv6 route ::/0 fe80::2e0:81ff:fe09:0990 fxp0
!
access-list vty-access permit 127.0.0.1/32
```

```

access-list vty-access deny any
!
ipv6 access-list vty-access permit ::1/128
ipv6 access-list vty-access deny any
!
line vty
  access-class vty-access
  ipv6 access-class vty-access
  exec-timeout 0 0
!

```

The above configuration still doesn't enable encryption; we use no encryption when creating a configuration file for the first time. Remember that for IPv6, the next hop address for a route entry must use link-local address. Thus we see that the next hop address for default route is fe80::2e0:81ff:fe09:0990, which is the EUI-64 format for *RGW*'s MAC address.

ospfd.conf on R2

```

!
hostname router2-ospf
password ospf-r2
enable password ospf-r2
log syslog
!
interface fxp0
!
interface fxp1
!
router ospf
  router-id 2.2.2.2
  network 10.1.2.0/23 area 0.0.0.0
!
access-list vty-access permit 127.0.0.1/32
access-list vty-access deny any
!
ipv6 access-list vty-access permit ::1/128
ipv6 access-list vty-access deny any
!
line vty
  access-class vty-access
  ipv6 access-class vty-access
  exec-timeout 0 0
!

```

The above configuration has a line of network configuration. We can do this because both interfaces are within a contiguous block of IP address space. If we are informed that our network's IP address space is 10.1.0.0/21, we can use this network definition on the OSPF configurations of all routers.

ospf6d.conf on R4

```
!  
hostname router4-ospfv3  
password ospfv3-r4  
enable password ospfv3-r4  
log syslog  
!  
interface fxp0  
!  
interface fxp1  
 ip ospf priority 100  
!  
router ospf  
 router-id 4.4.4.4  
 interface fxp0 area 0.0.0.0  
 interface fxp1 area 0.0.0.0  
!  
access-list vty-access permit 127.0.0.1/32  
access-list vty-access deny any  
!  
ipv6 access-list vty-access permit ::1/128  
ipv6 access-list vty-access deny any  
!  
line vty  
 access-class vty-access  
 ipv6 access-class vty-access  
 exec-timeout 0 0  
!
```

2.4.2 Operation

The most common task in operating a network is to monitor the network's health. We can use ping and traceroute to check the reachability and routing. Besides that, we also need to check our routers. We show some routers operations of the sample network in this section. The usual procedure in operating routers are checking the routing table of the router and the routing protocol's state.

Routing table in Zebra

Zebra's routing table can be viewed using these procedures. First, logon to the Zebra CLI. We show the example for *R2*.

```
> telnet localhost zebra
Trying ::1...
Connected to localhost.
Escape character is '^]'.

Hello, this is zebra (version 0.95-pre2).
Copyright 1996-2004 Kunihiro Ishiguro.
```

User Access Verification

```
Password:
zebra@Router2#
```

Below is the IPv6 routing table as the result of `show ipv6 route` command.

```
zebra@Router2# show ipv6 route
Codes: K - kernel route, C - connected, S - static, R - RIPng, O - OSPFv3,
       B - BGP, * - FIB route.

S>* ::/0 [1/0] via fe80::2e0:81ff:fe01:120, fxp0, 01w0d12h
K>* ::/96 via ::1, lo0
C>* ::1/128 is directly connected, lo0
K>* ::ffff:0.0.0.0/96 via ::1, lo0
O>* 3ffe:1:2:a::/64 [110/0] via fe80::2e0:81ff:fe01:120, fxp0, 01w0d12h
C>* 3ffe:1:2:b::/64 is directly connected, fxp0
C>* 3ffe:1:2:c::/64 is directly connected, fxp1
O>* 3ffe:1:2:d::/64 [110/0] via fe80::2e0:81ff:fe03:310, fxp1, 01w0d12h
O>* 3ffe:1:2:e::/64 [110/0] via fe80::2e0:81ff:fe04:410, fxp1, 01w0d12h
K>* fe80::/10 via ::1, lo0
C * fe80::/64 is directly connected, fxp1
C>* fe80::/64 is directly connected, fxp0
K>* ff02::/16 via ::1, lo0
```

The above results show the routing entries, including where the routes come from (ex: O for OSPFv3), the route entry cost, and which entries are installed in the FreeBSD's routing table (marked with *).

State of OSPFv3

Two main items that must be checked at an OSPFv3 (an OSPF) router are OSPFv3 routing table and adjacencies with neighbors. Here we show the neighbors state of R2.

```
ospf6d@Router2# show ipv6 ospf6 neighbor
Neighbor ID      Pri   DeadTime  State/IfState      Duration I/F[State]
1.1.1.1          1     00:00:37  Full/DR            15d12:58:02 fxp0[BDR]
3.3.3.3          1     00:00:34  Full/DROther       15d12:52:13 fxp1[BDR]
4.4.4.4          1     00:00:35  Full/DR            15d12:57:57 fxp1[BDR]
```

The items to be checked on neighbor states are:

1. Number of neighbors.
2. Interface's state: DR, BDR, or DROther

3. Neighbor's DeadTime.

If the DeadTime is less than RouterDeadInterval – HelloInterval, there might be problems.

4. Adjacencies state with neighbors.

The adjacency state with a neighbor depends on the router's interface state. If it is DR or BDR, then it must be Full with other routers. If it is DROther, it must be Full with DR and BDR, and two-way with others.

You can check the OSPFv3 routing table using `show ipv6 ospf6 route` command. If the neighbor states do not show any problem but the routing table is not correct, the problem comes from other routers.

2.5 Troubleshooting

You will certainly face troubles in operating networks. This section provides a guide to basic routing troubleshooting for routers that use Zebra routing package. The general procedure to troubleshoot routing problems with Zebra is:

1. Direct Zebra beast log to a file.

The command is: `log file <filename>`

2. Turn on related debugging messages.

For example, you want to debug packets received by Zebra from Zebra beasts. The command is:

```
debug zebra packet recv detail
```

3. Watch the log file.

Below are the outline of several symptoms and possible solution to the problems.

Routes are not installed in the kernel routing table.

1. Are the routes present in Zebra routing table?

Yes: May be Zebra problem; try to debug Zebra or restart Zebra. If the problem still persists, it may be a bug, find information from the Internet.

No: Zebra doesn't have the routes. If the routes are static, check Zebra configuration. If from a routing protocol, check the corresponding Zebra beast (step 2).

2. Are the routes present in Zebra beast routing table?

Yes: May be an inter-process communication problem between Zebra beasts. Restart the Zebra beast. If fails, restart all Zebra beasts.

No: Check the routing protocol states.

Problems in OSPF neighbor states.

1. No output.

Check the interfaces; the links may be disconnected or the interfaces are not enabled.

2. state = init
This state means that the router has seen Hello message from the neighbor, but the neighbor has not seen this router.
Check the firewall and the authentication type and key.
3. state = exstart or exchange
Neighbors in this state get stuck when trying to initiate database synchronization.
Check the MTU and try to ping the neighbor with large packets.
4. state = loading
Router is exchanging LSA, but the packets may be corrupted.
Debug the LSA packets.
5. state = two-way
Two routers that are not DR or BDR are in two-way state. If the router has Full state with the DR and BDR, then there is no problem. If there are no DR and BDR on the link, check the priority.

Missing routes in OSPF routing table.

1. Are all OSPF routes missing?
Check whether the router forms Full Adjacency with DR and BDR.
2. Are only External routes missing? Check the OSPF as-external database whether the advertising router is an AS border router.

2.6 Exercise

Ex. 1: IPv4 static routing

In this exercise you will enable an IPv4 router and populate the IPv4 routing table manually.

1. Check the network topology you received from the instructor.
2. Log on as `root`, edit `/etc/rc.conf`.

```
vi /etc/rc.conf
```
3. Configure the network interfaces and enable your host as a router. For example:

```
ifconfig_r10="inet 10.1.1.1 netmask 255.255.255.0"  
ifconfig_r10="inet 10.2.2.2 netmask 255.255.255.0"  
gateway_enable="YES"
```
4. Reboot your machine.
5. Log on as `root`.
6. Determine and write the nexthop gateway from your FreeBSD to reach each network.

7. Add five routing entries to the routing table of your FreeBSD using `route` command. Example: `route add 10.10.10.0/24 10.10.1.1` is to add routing entry to reach network 10.10.10.0/24 via 10.10.1.1.

No.	IP-address/prefix-length	next-hop gateway
1		
2		
3		
4		
5		

8. Confirm the route to each network using `traceroute` command. What are the results? Why such results appear?

Ex. 2: IPv6 router

In this exercise you will enable an IPv6 router.

1. Log on as `root`, edit `/etc/rc.conf`, and add the below line to the file.
`ipv6_gateway_enable="YES"`
2. Check the network topology you received from the instructor, and set the global unicast addresses to your network interfaces by configuring `/etc/rc.conf`.
A sample configuration line is
`ipv6_ifconfig_r10="2001:d30:105:201::1 prefixlen 64"`
3. Enable Router Advertisement to the downstream interface.
For example, suppose the downstream interface is `r11`.
`rtadvd_enable="YES"`
`rtadvd_interfaces="r11"`
4. Save the configuration.
5. Reboot your machine.
6. Log on as `root`.
7. Check the network interfaces.
`ifconfig -a`
8. Check the IPv6 routing table.
`netstat -nrf inet6`
9. Discuss your findings.

Ex. 3: Installing Zebra

1. Download Zebra source code.
You can download from the Zebra website <http://www.zebra.org/>
We provide a local copy in this workshop; the instructor will give the details.
2. Untar the package. At the command prompt, type:

```
tar xzpvf zebra-0.95pre2.tar.gz
```
3. Run the commands below:

```
cd zebra
./configure
make
```
4. As `root`, install Zebra by typing:

```
make install
```

Ex. 4: Configuring Zebra

In this exercise you will configure Zebra with the following requirements.

- plain text password
 - log is using syslog
 - suppress RA
 - no static route
 - limit access from localhost only
1. As `root`, copy the sample configuration to the Zebra configuration.

```
cd /usr/local/etc
cp zebra.conf.sample zebra.conf
```
 2. Edit Zebra configuration.

```
vi zebra.conf
```
 3. Remove all configuration lines except these lines.

```
hostname Router
password zebra
enable password zebra
```
 4. Configure Zebra based on the above requirements.
 5. Check the correctness of your configuration.

Ex. 5: Configuring Ospf6d

In this exercise you will configure Ospf6d with the following requirements.

- plain text password
- log is using syslog
- cost 50 for all interfaces
- priority 1 for interface to upstream, 100 to downstream
- router ID is the IPv4 address of the upstream interface
- area is backbone
- limit access from localhost only

1. Create Ospf6d configuration.
`vi ospf6d.conf`
2. Configure Ospf6d based on the above requirements.
3. Check the correctness of your configuration.

Ex. 6: Configuring Ospfd

In this exercise you will configure Ospfd with the following requirements.

- plain text password
- log is using syslog
- cost 50 for all interfaces
- priority 1 for interface to upstream, 100 to downstream
- area is backbone
- limit access from localhost only

1. Create Ospfd configuration.
`vi ospfd.conf`
2. Configure Ospfd based on the above requirements.
3. Check the correctness of your configuration.

Ex. 7: Operating Zebra

1. See the routing table
`netstat -nr`
2. As root, run Zebra the kernel routing manager.
`/usr/local/sbin/zebra -d`
If you don't see any error messages, Zebra should be running.
3. Confirm that the Zebra process is running.
`ps -xa | grep zebra`
If your configuration is syntactically correct, Zebra should be running. If it doesn't, check the syslog for errors.
4. Access the Zebra CLI, log on using the configured password.
`telnet localhost zebra`
5. At the zebra CLI, type a question mark (?).
Typing a question mark is the way to get information and help in completing your command in any Zebra beast CLI.
6. See the interfaces.
`show interface`
7. See the IP and IPv6 routing tables.
`show ip route`
`show ipv6 route`
8. Enter to the privilege mode.
`enable`
If you configure the enable password, you will be asked for a password.
9. See the start-up configuration.
`show startup-config`
10. See the running configuration.
`show running-config`
If you had changed the configuration without saving it, the running and start-up configuration would be different.
11. Enter the configuration mode.
`configure terminal`
12. Set enable password and activate password encryption.
`enable password <your password here>`
`service password-encryption`
13. Save your configuration.
`write memory`

14. Quit from Zebra CLI.
`exit`
15. Read the Zebra configuration, see the change.
`more /usr/local/etc/zebra.conf`
16. See the routing table again
`netstat -nr`
17. Learn the differences before and after running zebra.

Ex. 8: Operating OSPFv3 with Zebra package

1. See kernel's IPv6 routing table
`netstat -nr -f inet6`
2. Run Ospf6d.
`/usr/local/sbin/ospf6d -d`
If you don't see any error messages, Ospf6d should be running.
3. Confirm that the Ospf6d process is running.
`ps -xa | grep ospf6d`
If your configuration is syntactically correct, Ospf6d should be running. If it doesn't, check the syslog for errors.
4. Access the Ospf6d CLI, log on using the configured password.
`telnet localhost ospf6d`
5. Confirm that your configuration is OK.
`enable`
`show running-config`
6. Enter the configuration mode.
`configure terminal`
7. Set enable password and activate password encryption.
`enable password <your password here>`
`service password-encryption`
8. Save your configuration.
`write memory`
9. When finished, turn off privileged mode command.
`disable`
10. See the OSPFv3 interface information.
`show ipv6 ospf6 interface`
11. See the OSPFv3 neighbor list.
`show ipv6 ospf6 neighbor`

5. Confirm that your configuration is OK.

```
enable
show running-config
```
6. Enter the configuration mode.

```
configure terminal
```
7. Set enable password and activate password encryption.

```
enable password <your password here>
service password-encryption
```
8. Save your configuration.

```
write memory
```
9. When finished, turn off privileged mode command.

```
disable
```
10. See the OSPF interface information.

```
show ip ospf interface
```
11. See the OSPF neighbor list.

```
show ip ospf neighbor
```

You should see a list of neighbors with their status. Write down the neighbor list.

RouterID	State/Duration	DR	BDR	I/F[State]

Analyze and discuss the neighbor list.
12. See the OSPF routing table.

```
show ip ospf route
```

Analyze and discuss the routing table.
13. Quit Ospf CLI.

```
exit
```
14. Access Zebra CLI.

```
telnet localhost zebra
```
15. See the IP routing table.

```
show ip route
```

What has been changed? Analyze and discuss.
16. Quit Zebra CLI.

```
exit
```

17. See the kernel's IPv4 routing table.
`netstat -nrf inet`

18. Learn the differences before and after running ospfd.

Ex. 10: Installing Zebra to the FreeBSD start-up configuration

1. Create the start-up file for Zebra beasts.
Create a file `/usr/local/etc/rc.d/zebra.sh`, whose content is as below.

```
#!/bin/sh

case "$1" in
start)
    /usr/local/sbin/zebra -d && echo -n ' zebra'
    /usr/local/sbin/ospfd -d && echo -n ' ospfd'
    /usr/local/sbin/ospf6d -d && echo -n ' ospf6d'
    ;;
stop)
    killall ospf6d && echo -n ' ospf6d'
    killall ospfd && echo -n ' ospfd'
    killall zebra && echo -n ' zebra'
    ;;
*)
    echo "Usage: 'basename $0' {start|stop}" >&2
    ;;
esac

exit 0
```

2. Change the start-up file mode into executable.
`chmod +x /usr/local/etc/rc.d/zebra.sh`
3. Confirm all your changes.
4. Reboot the FreeBSD.
5. After the FreeBSD has restarted, confirm that Zebra, Ospfd, and Ospf6d are working correctly.

Chapter 3

PIM-SM Multicast Routing with XORP

3.1 Overview

Multicast is a many-to-many communication model in the Internet where a host send packets to a group of receivers. The intermediate routers between the sending host and receivers are responsible for forwarding packets and duplicating them when necessary, to reach all member of the group. The main advantage of multicast over unicast is a host only needs to send a single copy of data to the network. On the other hand, multicast communication uses UDP therefore it is a best effort delivery. Reliability, congestion control, and such functions have to be handled by multicast applications.

A host joins to a multicast address in order to receive multicast traffic being sent to the address. Meanwhile, a host sending multicast traffic does not have to join to the multicast address. It just sends packets to the multicast address, and routers on the local network with the host are responsible to forward the traffic toward the receivers.

Multicast is assigned special address prefix in IPv4 and IPv6 networks. The prefix in IPv4 is 224.0.0.0/4, and in IPv6 is ff00::/8. These prefixes can only be used as the destination address in IP packets.

Multicast works using two types of control messages: 1. between routers and hosts on local network; and 2. between routers. IGMP and MLD messages are used to manage multicast group membership on a local network. A host joining an IPv6 multicast group sends an MLD packet (IGMP for IPv4), and routers on the local network receive the packet. One of the routers then communicates with other routers on its upstream network toward the source using a multicast routing protocol in order to enable multicast traffic flowing to the receiver host.

Communications between routers result in a multicast distribution tree for the multicast traffic to reach all receivers. Multicast traffic flows from a source to receivers according to the multicast distribution tree. There are two types of distribution trees:

- Source or Shortest Path Tree
The root of the tree is the multicast source. This tree is abbreviated as SPT.
- Shared Tree
Distribution trees share a root, which is called the Rendezvous Point (RP). This tree is also called Rendezvous Point Tree (RPT).

The SPT is the most efficient tree because the traffic flows from the source to receivers using the shortest path. If RPT is used, multicast traffic from a source first goes to the RP then flows from the RP to the receivers using the shortest path.

The process to forward multicast traffic along the distribution tree is called multicast forwarding. Multicast forwarding is backward from unicast routing, i.e. it concerns about where a packet comes from. When a router receives a multicast packet, first it checks its routing table whether the packet arrives at the correct interface toward the source address of the packet. This process is called Reverse Path Forwarding (RPF) Check. If the packet fails the RPF check, it is dropped. If the packet passes this check, the router forwards the packet to the outgoing interfaces, i.e. interfaces that have downstreams, not including the interface where the packet is received.

Routers use multicast routing protocols to build multicast distribution trees. The available multicast routing protocols are:

- Distance Vector Multicast Routing Protocol (DVMRP)
exchanges routing information to build its own routing table and uses flood-prune mechanism.
- Multicast Open Shortest Path First (MOSPF)
extends OSPF for multicast trees.
- Core Based Tree (CBT)
uses the existing unicast routing protocol to determine RPF and uses join-prune mechanism to build tree.
- Protocol Independent Multicast (PIM)
uses the existing unicast routing protocol to determine RPF and uses join-prune mechanism to build tree.

DVMRP is the first standard multicast routing protocol. It is a dense-mode protocol, which uses the flood-and-prune mechanism to build multicast trees. With this mechanism, all routers periodically forward all multicast packets to all interfaces, except the interface where a router receives the packet, regardless of whether the router has a downstream or not. This process is called *flooding*, because multicast packets reach all parts of the network. After flooding, a router on the leaf network sends a *Prune* message toward the source when it does not find any receivers on its local network that want to receive the forwarded multicast traffic. The prune message is forwarded hop-by-hop toward the multicast source until every routers that do not have downstreams stop forwarding multicast traffic. The result is a multicast distribution tree for each multicast group.

As we can see, flood-and-prune mechanism is not efficient for a network with a small number of receivers relative to the network size. Therefore, sparse-mode multicast routing

protocols are proposed to eliminate the problems of dense-mode multicast routing protocol. A sparse-mode multicast routing protocol uses explicit *Join* and *Prune* messages to build multicast distribution trees. When a host joins a multicast group, a router on the local network sends a Join message to upstream in order to build a multicast distribution tree to reach the host. A router sends a Prune message to tear the multicast tree when there is no receiver or downstream router for the multicast group. Thus, sparse-mode multicast routing protocols do not waste bandwidth as dense-mode routing protocols.

Two sparse-mode multicast routing protocols are CBT and PIM-SM (PIM Sparse Mode). PIM-SM (PIM Sparse Mode) is de-facto standard multicast routing protocol on the Internet. This chapter focuses on how to run PIM-SM on FreeBSD using XORP routing daemon.

3.2 PIM-SM

PIM relies on the underlying system to populate its routing table, called Multicast Routing Information Base (MRIB). Routes from unicast routing table such as OSPF or from multicast address family of MBGP (Multiprotocol Border Gateway Protocol) are typically used to populate the MRIB. MRIB is used to perform RPF check and to send Join/Prune messages to build/destroy multicast distribution tree. PIM-SM builds a multicast distribution tree in three phases:

1. RP tree
2. Register stop
3. Shortest path tree

Phase one: RP tree

When PIM-SM routing protocol starts the operation, PIM-SM routers on the network elect a router's interface to be the Rendezvous Point (RP) of a certain multicast group. On the link level, PIM-SM routers on a link elects a router to be the Designated Router (DR) that will act on behalf of hosts on the link. These routers must be elected before PIM-SM can build multicast distribution trees. When a host wants to receive multicast traffic to group G, the DR on the same link with the host issues a (*,G) Join message to the next hop toward the RP. Each PIM routers along the path from the receiver to the RP send a (*,G) Join message to the next-hop toward RP. The RP tree is complete when (*,G) Join message reaches the RP or reaches a router that already has (*,G) Join state. This shared tree is rooted at the RP. Join messages have to be resent periodically to keep the state in the upstream routers.

When hosts on a leaf network no longer want to receive traffic to the group, the DR will issue a Prune message to the upstream router. If the upstream router no longer has any downstream, it leaves the Join state and issues a Prune message to its upstream router. Prune messages are sent upward to the root of the tree, and are stopped by the first router that still has a downstream for the corresponding group.

When a source S sends traffic to a multicast group G, the Designated Router (DR) on the same link with S encapsulates the multicast packets from S, and sends the encapsulated packets to the RP as unicast packets. The encapsulated packet is called PIM Register

message. The RP decapsulates PIM Register messages back to the native multicast packets and forward the packets onto the shared tree.

Phase two: Register stop

Sending PIM Register packet is inefficient because packets must be encapsulated at DR and decapsulated at RP. An RP may choose to stop receiving Register messages and switch to native multicast forwarding. If it does so, it initiates a (S,G) Join toward S. Routers along the path between RP and S send (S,G) Join to their RPF next hop. When (S,G) Join arrives at the DR of S, the DR starts to forward packets from S using both native multicast and Register-encapsulated messages. RP then starts receiving two copies of packets from S; and when that happens it sends a Register-Stop message to S's DR. The DR stops sending Register-encapsulated packets to RP, completing the phase two. After the phase two is complete, only native multicast packets are flowing from S to RP, removing the inefficiencies of Register packets.

Phase three: Shortest path tree

The path delay from S to a receiver may be reduced if the receiver doesn't use shared tree. The router on the same subnet with a receiver, typically DR, may optionally switch to the source tree. If it switches to the source tree, it issues an (S,G) Join to the RPF next hop toward S. The (S,G) Join will eventually arrives at the subnet of S or at a router that already has (S,G) Join state. The routers then starts forwarding packets onto the source tree to the receiver. When the first traffic begin arriving at S's subnet, the DR or the upstream router sends a (S,G) Prune toward the RP to the upstream router. This message is known as (S,G,rpt) Prune, which says that RP should not forward traffic coming from S to G. (S,G,rpt) Prune message travels hop-by-hop until it reaches RP or a router that still needs traffic from S.

PIM Assert

Multiple copies of a multicast traffic may arrive on a transit LAN from different routers. PIM-SM doesn't prevent this from happening, instead it provides a mechanism to elect a single forwarder, using PIM Assert messages. Routers on a link elect a single forwarder in favor of the router that has the best metric toward the source, then toward RP.

RP discovery

PIM-SM uses a Rendezvous Point (RP) for each multicast group. An RP serves two purposes:

- as the root of shared trees; and
- for senders and receivers to learn the existence of each others.

Routers on the network should know the RP for a group, otherwise it cannot form the multicast distribution tree for the group. The RP for each group may be statically configured at each router. PIM-SM provides a mechanism to discover the RP dynamically using Bootstrap Router mechanism. A router in a PIM domain is elected as the Bootstrap

Router. Routers that are configured to become RP candidate inform their candidacy to the Bootstrap Router (BSR). The BSR gathers the candidacy information and creates an RP-set then send the RP-set in Bootstrap messages. The Bootstrap messages are flooded throughout the PIM domain until all routers know the RP-set.

3.3 Multicast Routing on FreeBSD

Multicast-routing enabled kernel

The FreeBSD's generic kernel does not support IPv4 multicast routing, but IPv6 multicast routing support is already enabled. To enable multicast routing, the `MROUTING` kernel option must be activated. Also, if you want to run PIM-SM multicast routing protocol, the `PIM` kernel option must be activated. The kernel configuration lines to enable PIM-SM multicast routing are:

```
options MROUTING
options PIM
```

Multicast routing cache table

Multicast routing cache table and the virtual interface table can be displayed using `netstat -ng` command. Below is an example of the result for IPv4.

Virtual Interface Table

Vif	Thresh	Rate	Local-Address	Remote-Address	Pkts-In	Pkts-Out
0	1	0	10.10.20.1		18871	5255
1	1	0	10.50.20.1		5255	12740
2	1	0	10.10.20.1		0	2532

IPv4 Multicast Forwarding Cache

Origin	Group	Packets	In-Vif	Out-Vifs:Ttls
10.1.1.10	239.18.100.100	208	0	2:1

In this example, the FreeBSD router has three virtual interfaces (Vif), each is identified by a number. Each Vif may forward multicast packets whose TTL is at least same as **Thresh** value. Each Vif has a rate limit, where 0 represents no limit. If the Vif is a tunnel, then the remote address is also displayed. The number of incoming and outgoing packets are also displayed. In the IPv4 multicast forwarding cache you can see there is a multicast source sending packets from the Vif 0 and the packets are forwarded to Vif 2 if the TTL is more than 1.

The Vif 2 is the interface to send PIM Register packets to PIM Rendezvous Point. This is not clearly displayed in the netstat results, however you can see it using the PIM-SM multicast routing protocol daemon.

An example for the IPv6 is displayed below. You can see the difference between the interface tables of IPv4 and IPv6. The IPv6 interface table uses the physical interface name instead of the IP address of the interface. The `reg0` interface in the table is the interface to send PIM Register packets. The IPv6 multicast forwarding cache is similar to that of IPv4.

IPv6 Multicast Interface Table

Mif	Rate	PhyIF	Pkts-In	Pkts-Out
0	0	fxp0	23834	10097
1	0	fxp1	10097	12066
2	0	reg0	0	9833

IPv6 Multicast Forwarding Cache

Origin	Group	Packets	Waits	In-Mif	Out-Mifs
3ffe:1:2:a:2d0:b7ff:fe9e:e5d2	ff0e::100	209	0	0	2

3.4 XORP for PIM-SM Multicast Routing

In this section we will explain the use of XORP routing daemon for PIM-SM multicast routing. The eXtensible Open Router Platform (XORP) is an open source router platform that supports unicast and multicast routing protocols for IPv4 and IPv6. It runs on several operating systems, including FreeBSD. The current version of XORP (<http://www.xorp.org>) is release 1.1.

The XORP process model is displayed in Figure 3.1. XORP can be divided into two subsystems: the "user-space", which handles management processes and routing protocols, and the "kernel-space", which handles the forwarding path and provides API to the user-space.

Figure 3.1: XORP process model

Installing XORP

To compile XORP release 1.1, you must have GNU make (gmake) installed. XORP also needs Net-SNMP package for installation and operation. GNU make and Net-SNMP may

be installed using the package or port installation. The details on how to install XORP is available in the XORP tarball. By default, XORP will be installed in `/usr/local/xorp`. Besides this, you also have to create `xorp` group.

Configuring XORP for PIM-SM

XORP uses a configuration file whose name is `/usr/local/xorp/config.boot` by default. A sample configuration is available in `rtrmgr` directory inside the XORP source tree.

A basic configuration of XORP to enable PIM-SM for IPv4 and IPv6 is quite long, therefore we explain the configuration part-by-part.

```
/* PART I: network interface definition */
interfaces {
    interface fxp0 {
        description: "upstream interface"
        disable: false
        default-system-config
    }
    interface fxp1 {
        description: "downstream interface"
        disable: false
        default-system-config
    }
}
```

Above is the network interfaces part. This configuration enables two physical interfaces. An interface must be explicitly enabled for XORP operation. An interface statement refers to a physical interface of the host. XORP may have several virtual interface for an interface, and in majority of cases, there is only a virtual interface for a physical interface, and the names are identical. The `default-system-config` statement tells XORP to use the IP addresses etc. that is set by the system, i.e. the kernel on FreeBSD.

Below is the Multicast Forwarding Engine Abstraction (MFEA) for both IPv4 and IPv6. The `mfea4` and `mfea6` statements are MFEA statements for IPv4 and IPv6, respectively. The virtual interface `register_vif` is for PIM-SM Register messages, and should always be enabled for PIM-SM operation. The `traceoptions` statement is for debugging purposes.

```
/* PART II: Multicast Forwarding Engine Abstraction definition */
plumbing {
    mfea4 {
        disable: false
        interface fxp0 {
            vif fxp0 {
                disable: false
            }
        }
        interface fxp1 {
            vif fxp1 {
                disable: false
            }
        }
        interface register_vif {
            vif register_vif {
                /* Note: this vif should be always enabled */
            }
        }
    }
}
```

```

        disable: false
    }
}
traceoptions {
    flag all {
        disable: false
    }
}
}
}

plumbing {
    mfea6 {
        disable: false
        interface fxp0 {
            vif fxp0 {
                disable: false
            }
        }
        interface fxp1 {
            vif fxp1 {
                disable: false
            }
        }
        interface register_vif {
            vif register_vif {
                disable: false
            }
        }
        traceoptions {
            flag all {
                disable: false
            }
        }
    }
}
}
}

```

The next part is the protocol part for IGMP and MLD.

```

/* PART IIIa: MLD and IGMP definitions */
protocols {
    igmp {
        disable: false
        interface fxp0 {
            vif fxp0 {
                disable: false
            }
        }
        interface fxp1 {
            vif fxp1 {
                disable: false
            }
        }
        traceoptions {
            flag all {
                disable: false
            }
        }
    }
}
}

```

```

    }
  }
  mld {
    disable: false
    interface fxp0 {
      vif fxp0 {
        disable: false
      }
    }
    interface fxp1 {
      vif fxp1 {
        disable: false
      }
    }
    traceoptions {
      flag all {
        disable: false
      }
    }
  }
}

```

The next part is for PIM-SM configuration. You can see the three interfaces are enabled. Interface `fxp0` is given priority 1, and `fxp1` is given priority 255, which is the highest, to be the Designated Router (DR), using `dr-priority` command. The default DR priority value is 1. This router has a static RP set, and also it is set to become a BSR and an RP candidate for all multicast groups (224.0.0.0/4 and ff00::/8). Which RP will be used by this router, whether the static RP or the RP from Bootstrap mechanism, depends on the priority. An RP candidate with the highest priority becomes the RP.

Bootstrap mechanism is used to elect RP dynamically. A set of routers may be configured as candidate Bootstrap Routers (BSR), and one of them will be elected for a network (PIM domain) based on the BSR priority. Each RP candidate periodically announces itself to the BSR using unicast Candidate-RP-Advertisement messages. The BSR pools the candidate RPs and periodically sends Bootstrap messages containing the set of Candidate-RPs. Each routers on the PIM domain elect an RP for each multicast group using the same algorithm, thus they will use the same RP for each multicast group.

Switching to the SPT is set by `switch-to-spt-threshold` statement, where in this case it will happen if total 10240 bytes arrive within 100 seconds. You can also see that the traceoption flags are enabled for this protocol.

The `fib2mrib` protocol statement at the end of this part tells XORP to populate MRIB with unicast routing entries.

```

/* PART IIIb: PIM-SM definition */
protocols {
  pimsm4 {
    disable: false
    interface fxp0 {
      vif fxp0 {
        disable: false
        dr-priority: 1
      }
    }
  }
}

```

```
interface fxp1 {
  vif fxp1 {
    disable: false
    dr-priority: 255
  }
}
interface register_vif {
  vif register_vif {
    /* Note: this vif should be always enabled */
    disable: false
  }
}

static-rps {
  rp 10.1.1.1 {
    group-prefix 224.0.0.0/4 {
      /* rp-priority: 192 */
    }
  }
}

bootstrap {
  disable: false
  cand-bsr {
    scope-zone 224.0.0.0/4 {
      cand-bsr-by-vif-name: "fxp0"
      bsr-priority: 128
      hash-mask-len: 30
    }
  }

  cand-rp {
    group-prefix 224.0.0.0/4 {
      cand-rp-by-vif-name: "fxp0"
      rp-priority: 192
      rp-holdtime: 150
    }
  }
}

switch-to-spt-threshold {
  /* approx. 1K bytes/s (10Kbps) threshold */
  disable: false
  interval-sec: 100
  bytes: 102400
}

traceoptions {
  flag all {
    disable: false
  }
}
}

pimsm6 {
  disable: false
}
```

```
interface fxp0 {
  vif fxp0 {
    disable: false
    dr-priority: 1
  }
}
interface fxp1 {
  vif fxp1 {
    disable: false
    dr-priority: 255
  }
}
interface register_vif {
  vif register_vif {
    disable: false
  }
}

static-rps {
  rp 3ffe:1:2:a::1 {
    group-prefix ff00::/8 {
      rp-priority: 192
      hash-mask-len: 126
    }
  }
}

bootstrap {
  disable: false
  cand-bsr {
    scope-zone ff00::/8 {
      cand-bsr-by-vif-name: "fxp0"
      bsr-priority: 128
      hash-mask-len: 30
    }
  }

  cand-rp {
    group-prefix ff00::/8 {
      cand-rp-by-vif-name: "fxp0"
      rp-priority: 192
      rp-holdtime: 150
    }
  }
}

switch-to-spt-threshold {
  disable: false
  interval-sec: 100
  bytes: 102400
}

traceoptions {
  flag all {
    disable: false
  }
}
```

```

    }
}

protocols {
    fib2mrib {
        disable: false
    }
}

```

Running XORP

XORP must be run as root by invoking the router manager, i.e.

```
/usr/local/xorp/bin/xorp_rtrmgr
```

While to access the CLI you run the shell:

```
/usr/local/xorp/bin/xorpsh
```

To enable XORP running at boot time, create a startup file `/usr/local/etc/rc.d/xorp.sh` and make it executable. The contents are

```
#!/bin/sh

case "$1" in
start)
    /usr/local/xorp/bin/xorp_rtrmgr &
    echo -n ' xorp'
    ;;
stop)
    killall xorp_rtrmgr
    echo -n ' xorp'
    ;;
*)
    echo "Usage: 'basename $0' {start|stop}" >&2
    ;;
esac

exit 0

```

Operating PIM-SM Multicast Routing

We will use the network topology in Figure 3.2 to explain how to operate multicast routing with PIM-SM. Table 3.1 shows the addresses for each interface.

PIM-SM network design for this topology is as follows:

1. RPs are elected dynamically using Bootstrap mechanism.
2. R1 becomes the only candidate BSR and candidate RP.

With this configuration, the above configuration sample becomes the configuration file for R1 if the static RP configuration command is removed.

Table 3.1: Interface Addresses for Figure 3.2

Interface	MAC address	IPv6 address	IPv4 address
R1-fxp0	0:e0:81:1:1:10	3ffe:1:2:a::1	10.1.1.1
R1-fxp1	0:e0:81:1:1:20	3ffe:1:2:b::1	10.1.2.1
R2-fxp0	0:e0:81:2:2:10	3ffe:1:2:b::2	10.1.2.2
R2-fxp1	0:e0:81:2:2:20	3ffe:1:2:c::2	10.1.3.2
S1-fxp0	0:e0:81:8:8:80	3ffe:1:2:a::8	10.1.1.8
S2-fxp0	0:e0:81:7:7:70	3ffe:1:2:c::9	10.1.3.7

Figure 3.2: Sample network topology

The PIM-SM states that have to be watched are:

1. interface
2. neighbor
3. MRIB
4. bootstrap
5. Rendezvous Points
6. Join

Interface states

PIM router interface state can be displayed using the following commands at XORP CLI: `show pim interface` for IPv4, and `show pim6 interface` for IPv6. Below are the results for the sample network topology. From these results you can see that fxp1 interfaces of router R1 is assigned with Priority 255, and they are the Designated Router for their links. The DR address is also shown in the results. The `register_vif` interface is the interface

to send PIM Register packets to the Rendezvous Point. For IPv6, you can see that the DR address of `register_vif` is a global address, while other interfaces use link-local address. You can also see how many neighbors are there for each interface.

R1

```
Xorp> show pim interface
Interface    State    Mode    V PIMstate Priority DRaddr           Neighbors
fxp0         UP      Sparse 2 DR      1 10.1.1.1         0
fxp1         UP      Sparse 2 DR      255 10.1.2.1         1
register_vif UP      Sparse 2 DR      1 10.1.1.1         0
Xorp> show pim6 interface
Interface    State    Mode    V PIMstate Priority DRaddr           Neighbors
fxp0         UP      Sparse 2 DR      255 fe80::2e0:81ff:fe01:110 0
fxp1         UP      Sparse 2 DR      255 fe80::2e0:81ff:fe01:120 1
register_vif UP      Sparse 2 DR      1 3ffe:1:2:a::1     0
```

R2

```
Xorp> show pim interface
Interface    State    Mode    V PIMstate Priority DRaddr           Neighbors
fxp0         UP      Sparse 2 NotDR    1 10.1.2.1         1
fxp1         UP      Sparse 2 DR      255 10.1.3.2         0
register_vif UP      Sparse 2 DR      1 10.1.2.2         0
Xorp> show pim6 interface
Interface    State    Mode    V PIMstate Priority DRaddr           Neighbors
fxp0         UP      Sparse 2 NotDR    1 fe80::2e0:81ff:fe01:120 1
fxp1         UP      Sparse 2 DR      255 fe80::2e0:81ff:fe02:220 0
register_vif UP      Sparse 2 DR      1 3ffe:1:2:b::2     0
```

Neighbor states

You can display PIM neighbor states using `show pim neighbors`, and `show pim6 neighbors` commands. For each neighbor you can see its priority to become DR, how much time is left before the router deletes the neighbor if the router doesn't hear a Hello message from the neighbor. XORP also displays all addresses of each neighbor.

R1

```
Xorp> show pim neighbors
Interface    DRpriority NeighborAddr    V Mode    Holdtime Timeout
fxp1         1 10.1.2.2        2 Sparse    105      79
Xorp> show pim6 neighbors
Interface    DRpriority NeighborAddr    V Mode    Holdtime Timeout
fxp1         1 fe80::2e0:81ff:fe02:210 2 Sparse    105      80
              3ffe:1:2:b::2
```

R2

```
Xorp> show pim neighbors
Interface    DRpriority NeighborAddr    V Mode    Holdtime Timeout
fxp0         255 10.1.2.1        2 Sparse    105      91
Xorp> show pim6 neighbors
Interface    DRpriority NeighborAddr    V Mode    Holdtime Timeout
fxp0         255 fe80::2e0:81ff:fe01:120 2 Sparse    105      85
              3ffe:1:2:b::1
```

MRIB

You can display the multicast routing information base using `show pim mrib` and `show pim6 mrib` commands. The MRIBs for our sample network are as follows:

R1

```
Xorp> show pim mrib
```

DestPrefix	NextHopRouter	VifName	VifIndex	MetricPref	Metric
10.1.1.0/24	10.1.1.1	fxp0	0	0	0
10.1.2.0/24	10.1.2.1	fxp1	1	0	0
10.1.3.0/24	10.1.2.2	fxp1	1	254	65535

```
Xorp> show pim6 mrib
```

DestPrefix	NextHopRouter	VifName	VifIndex	MetricPref	Metric
3ffe:1:2:a::/64	3ffe:1:2:a::1	fxp0	0	0	0
3ffe:1:2:b::/64	3ffe:1:2:b::1	fxp1	1	0	0
3ffe:1:2:c::/64	fe80::2e0:81ff:fe02:210	fxp1	1	254	65535
fe80::/64	fe80::2e0:81ff:fe01:110	fxp0	0	0	0

R2

```
Xorp> show pim mrib
```

DestPrefix	NextHopRouter	VifName	VifIndex	MetricPref	Metric
10.1.1.0/24	10.1.2.1	fxp0	0	254	65535
10.1.2.0/24	10.1.2.2	fxp0	0	0	0
10.1.3.0/24	10.1.3.2	fxp1	1	0	0

```
Xorp> show pim6 mrib
```

DestPrefix	NextHopRouter	VifName	VifIndex	MetricPref	Metric
3ffe:1:2:a::/64	fe80::2e0:81ff:fe01:120	fxp0	0	254	65535
3ffe:1:2:b::/64	3ffe:1:2:b::2	fxp0	1	0	0
3ffe:1:2:c::/64	3ffe:1:2:c::2	fxp1	1	0	0
fe80::/64	fe80::2e0:81ff:fe02:210	fxp0	0	0	0

Bootstrap states

Issuing `show pim bootstrap` for IPv4, and `show pim6 bootstrap` for IPv6 at the XORP CLI shows Bootstrap Routers and their states. XORP keeps three databases: Active, Expiring, and Configured zones. BSRs in Active zones are the ones that are currently active. Expiring zones shows the BSRs that have not been heard of for a while and will be deleted soon. Configured zones shows the locally configured BSRs.

From the results below, you can see that R1 is configured to be a BSR candidate, while R2 is not. Also, there is only one Bootstrap Router whose IPv6 address is 3ffe:1:2:a::1 (10.1.1.1 for IPv4), which is R1, and R1 is the elected Bootstrap Router. There is no BSR that will be deleted from the XORP database.

R1

```
Xorp> show pim bootstrap
```

```
Active zones:
```

BSR	Pri	LocalAddress	Pri	State	Timeout	SZTimeout
10.1.1.1	128	10.1.1.1	128	Elected	30	-1

```
Expiring zones:
```

BSR	Pri	LocalAddress	Pri	State	Timeout	SZTimeout
-----	-----	--------------	-----	-------	---------	-----------

```
Configured zones:
```

```

BSR          Pri LocalAddress  Pri State      Timeout SZTimeout
10.1.1.1     128 10.1.1.1    128 Init        30      -1
Xorp> show pim6 bootstrap
Active zones:
BSR          Pri LocalAddress  Pri State      Timeout SZTimeout
3ffe:1:2:a::1 128 3ffe:1:2:a::1 128 Elected    51      -1
Expiring zones:
BSR          Pri LocalAddress  Pri State      Timeout SZTimeout
Configured zones:
BSR          Pri LocalAddress  Pri State      Timeout SZTimeout
3ffe:1:2:a::1 128 3ffe:1:2:a::1 128 Init        51      -1

```

R2

```

Xorp> show pim bootstrap
Active zones:
BSR          Pri LocalAddress  Pri State      Timeout SZTimeout
10.1.1.1     128 0.0.0.0       0 AcceptPreferred 73      1243
Expiring zones:
BSR          Pri LocalAddress  Pri State      Timeout SZTimeout
Configured zones:
BSR          Pri LocalAddress  Pri State      Timeout SZTimeout
Xorp> show pim6 bootstrap
Active zones:
BSR          Pri LocalAddress  Pri State      Timeout SZTimeout
3ffe:1:2:a::1 128 ::           0 AcceptPreferred 73      1243
Expiring zones:
BSR          Pri LocalAddress  Pri State      Timeout SZTimeout
Configured zones:
BSR          Pri LocalAddress  Pri State      Timeout SZTimeout

```

RP states

The RP for each multicast group prefix are displayed by invoking `show pim rps` for IPv4, and `show pim6 rps` for IPv6, commands at the XORP CLI. The below results show that RP is learned from the Bootstrap mechanism. The RP for IPv4 is 10.1.1.1, and for IPv6 is 3ffe:1:2:a::1

R1

```

Xorp> show pim rps
RP          Type      Pri Holdtime Timeout ActiveGroups GroupPrefix
10.1.1.1    bootstrap 192    150    -1          1 224.0.0.0/4
Xorp> show pim6 rps
RP          Type      Pri Holdtime Timeout ActiveGroups GroupPrefix
3ffe:1:2:a::1 bootstrap 192    150    -1          0 ff00::/8

```

R2

```

Xorp> show pim rps
RP          Type      Pri Holdtime Timeout ActiveGroups GroupPrefix
10.1.1.1    bootstrap 192    150    131         1 224.0.0.0/4
Xorp> show pim6 rps
RP          Type      Pri Holdtime Timeout ActiveGroups GroupPrefix
3ffe:1:2:a::1 bootstrap 192    150    93          0 ff00::/8

```

Join states

The commands to view the PIM Join states of a router are `show pim join` and `show pim6 join`. Here we give an example where S2 joins to group ff0e::1:1.

Below are the multicast states of R1 and R2 after S2 joined the group (phase one). The first line of a multicast state entry has these fields:

- **Group** : the multicast group address
- **Source** : the source address
- **RP** : the RP address for this entry
- **Flags** : the set of flags for this entry. The flags are:
 - RP : (*,*,RP) routing entry
 - WC : (*,G) routing entry
 - SG : (S,G) routing entry
 - SG_RPT : (S,G,rpt) routing entry
 - SPT : the entry has the Shortest-Path Tree flag set
 - DirectlyConnectedS : the entry is for a directly connected source

The next three lines show the upstream interfaces toward the RP, the MRIB nexthop router toward RP, and the next hop router toward RP according to PIM. You can see that the upstream interface toward RP at R1 is the `register_vif`, while at R2 is `fxp0`. This is because R1 is the RP for this entry. Because R1 is the RP, the next hop router toward RP at R1 is itself, and XORP expresses it as `UNKNOWN`. The next hop router toward RP at R2 is the link-local address of interface `fxp1` of RP1. Note that the MRIB nexthop router toward RP and the next hop toward RP according to PIM may be different on a multi-access link due to PIM Assert mechanism.

The next two lines show the upstream state and the number of seconds until the upstream Join timeout.

The rest of the lines for a multicast state entry display various information about the entry using `Information : InterfaceSet` format. An interface set is a series of marks, either "." or "O", representing an interface starting with the first interface. An interface is marked with "O" if it is included in the interface set. Some of the information are explained below:

- **Local receiver include WC** : interfaces that have local (*,G) receivers according to the MLD/IGMP module
- **Joins RP, and Joins WC** : interfaces that have received (*,*,RP) Join, and (*,G) Join
- **Join state, and Prune state**: interfaces that are in Join, and Prune, state
- **Prune pending state** : interfaces that are in Prune-Pending state
- **Immediate olist RP, and Immediate olist WC** : interfaces that are included in the immediate outgoing interfaces for (*,*,RP), and (*,G), entry.

- **Inherited olist SG** : interfaces that are included in the outgoing interface list for packets forwarded on (S,G) state taking into account (*,*,RP) state, (*,G) state, asserts, etc.
- **Inherited olist SG RPT** : interfaces that are included in the outgoing interface list for packets forwarded on (*,*,RP) or (*,G) state taking into account (S,G,rpt) prune state, and asserts, etc.
- **PIM Include WC** : interfaces to which traffic might be forwarded because of group member hosts on the interface

Comparing the results of R1 and R2 below, you can see that R1 has downstream router(s) on fxp1 interface as shown by `Join WC: .0..` R2 has local receivers on fxp1 interface, based on `Local receiver include WC: .0.` information displayed by XORP.

R1

```
Xorp> show pim6 join|find ff0e::1:1
Group          Source          RP              Flags
ff0e::1:1      ::              3ffe:1:2:a::1  WC
  Upstream interface (RP):  register_vif
  Upstream MRIB next hop (RP): UNKNOWN
  Upstream RPF'(*,G):      UNKNOWN
  Upstream state:          Joined
  Join timer:              10
  Local receiver include WC: ...
  Joins RP:                ...
  Joins WC:                .0.
  Join state:              .0.
  Prune state:             ...
  Prune pending state:    ...
  I am assert winner state: ...
  I am assert loser state: ...
  Assert winner WC:        ...
  Assert lost WC:          ...
  Assert tracking WC:      .00
  Could assert WC:        .0.
  I am DR:                 000
  Immediate olist RP:      ...
  Immediate olist WC:      .0.
  Inherited olist SG:      .0.
  Inherited olist SG_RPT: .0.
  PIM include WC:         ...
```

R2

```
Xorp> show pim6 join| find ff0e::1:1
Group          Source          RP          Flags
ff0e::1:1     ::              3ffe:1:2:a::1  WC
  Upstream interface (RP):  fxp0
  Upstream MRIB next hop (RP): fe80::2e0:81ff:fe01:120
  Upstream RPF'(*,G):      fe80::2e0:81ff:fe01:120
  Upstream state:          Joined
  Join timer:              29
  Local receiver include WC: .0.
  Joins RP:                ...
  Joins WC:                ...
  Join state:              ...
  Prune state:             ...
  Prune pending state:    ...
  I am assert winner state: ...
  I am assert loser state: ...
  Assert winner WC:       ...
  Assert lost WC:         ...
  Assert tracking WC:     00.
  Could assert WC:       .0.
  I am DR:                .00
  Immediate olist RP:     ...
  Immediate olist WC:     .0.
  Inherited olist SG:     .0.
  Inherited olist SG_RPT: .0.
  PIM include WC:        .0.
```

Now S1 sends multicast traffic to ff0e::1:1, starting the phase two of PIM. Since S1 is on the same link with R1, the RP for this group, there is no need to send PIM Register messages. Below is the multicast state entries at R1, and as you can see, there are now two multicast state entries for group ff0e::1:1 at R1: one with any source (::), and one with source S1 (3ffe:1:2:a::8). There is no change on the any-source state entry. The entry with S1 as the source has several flags: SG, SPT, and DirectlyConnectedS. It also contains the register state, because R1 acts as the DR for the link to S1; and the register state is RegisterNotCouldRegister since R1 is also the RP for this group. The multicast state at R2 doesn't change.

```
Xorp> show pim6 join|find ff0e::1:1
Group          Source          RP          Flags
ff0e::1:1     ::              3ffe:1:2:a::1  WC
...
...

Group          Source          RP          Flags
ff0e::1:1     3ffe:1:2:a::8  3ffe:1:2:a::1  SG SPT DirectlyConnectedS
  Upstream interface (S):  fxp0
  Upstream interface (RP): register_vif
  Upstream MRIB next hop (RP): UNKNOWN
  Upstream MRIB next hop (S): UNKNOWN
  Upstream RPF'(S,G):     UNKNOWN
  Upstream state:          Joined
  Register state:          RegisterNoinfo RegisterNotCouldRegister
```

```

Join timer:                32
KAT(S,G) running:         true
Local receiver include WC: ...
Local receiver include SG: ...
Local receiver exclude SG: ...
Joins RP:                  ...
Joins WC:                   .0.
Joins SG:                   ...
Join state:                 ...
Prune state:                ...
Prune pending state:       ...
I am assert winner state:  ...
I am assert loser state:   ...
Assert winner WC:          ...
Assert winner SG:          ...
Assert lost WC:            ...
Assert lost SG:            ...
Assert lost SG_RPT:        ...
Assert tracking SG:        00.
Could assert WC:           .0.
Could assert SG:           .0.
I am DR:                    000
Immediate olist RP:        ...
Immediate olist WC:         .0.
Immediate olist SG:        ...
Inherited olist SG:        .0.
Inherited olist SG_RPT:    .0.
PIM include WC:            ...
PIM include SG:            ...
PIM exclude SG:           ...

```

Below are the multicast states at R1 and R2 in the final phase after R2 initiated the switch to SPT. R2 now has an entry for source S1 flagged with SG and SPT, while the entry for any source doesn't change. The change at R1 compared to the state at phase two is the entry with source S1 now includes (marks as "O") fxp1 interface in the following:

- Joins SG
- Join state
- Immediate olist SG

R1

```

Xorp> show pim6 join|find ff0e::1:1
Group      Source      RP      Flags
ff0e::1:1  ::         3ffe:1:2:a::1  WC
  Upstream interface (RP):  register_vif
  Upstream MRIB next hop (RP): UNKNOWN
  Upstream RPF'(*,G):      UNKNOWN
  Upstream state:          Joined
  Join timer:              43
  Local receiver include WC: ...
  Joins RP:                 ...
  Joins WC:                 .0.

```

```

Join state: .0.
Prune state: ...
Prune pending state: ...
I am assert winner state: ...
I am assert loser state: ...
Assert winner WC: ...
Assert lost WC: ...
Assert tracking WC: .00
Could assert WC: .0.
I am DR: 000
Immediate olist RP: ...
Immediate olist WC: .0.
Inherited olist SG: .0.
Inherited olist SG_RPT: .0.
PIM include WC: ...

ff0e::1:1      3ffe:1:2:a::8  3ffe:1:2:a::1  SG SPT DirectlyConnectedS
Upstream interface (S): fxp0
Upstream interface (RP): register_vif
Upstream MRIB next hop (RP): UNKNOWN
Upstream MRIB next hop (S): UNKNOWN
Upstream RPF'(S,G): UNKNOWN
Upstream state: Joined
Register state: RegisterNoinfo RegisterNotCouldRegister
Join timer: 25
KAT(S,G) running: true
Local receiver include WC: ...
Local receiver include SG: ...
Local receiver exclude SG: ...
Joins RP: ...
Joins WC: .0.
Joins SG: .0.
Join state: .0.
Prune state: ...
Prune pending state: ...
I am assert winner state: ...
I am assert loser state: ...
Assert winner WC: ...
Assert winner SG: ...
Assert lost WC: ...
Assert lost SG: ...
Assert lost SG_RPT: ...
Assert tracking SG: .00.
Could assert WC: .0.
Could assert SG: .0.
I am DR: 000
Immediate olist RP: ...
Immediate olist WC: .0.
Immediate olist SG: .0.
Inherited olist SG: .0.
Inherited olist SG_RPT: .0.
PIM include WC: ...
PIM include SG: ...
PIM exclude SG: ...

```

R2

```
Xorp> show pim6 join|find ff0e::1:1
Group          Source          RP              Flags
ff0e::1:1      ::              3ffe:1:2:a::1  WC
  Upstream interface (RP):  fxp0
  Upstream MRIB next hop (RP): fe80::2e0:81ff:fe01:120
  Upstream RPF'(*,G):      fe80::2e0:81ff:fe01:120
  Upstream state:          Joined
  Join timer:              11
  Local receiver include WC: .0.
  Joins RP:                ...
  Joins WC:                ...
  Join state:              ...
  Prune state:             ...
  Prune pending state:    ...
  I am assert winner state: ...
  I am assert loser state: ...
  Assert winner WC:       ...
  Assert lost WC:         ...
  Assert tracking WC:     00.
  Could assert WC:       .0.
  I am DR:                .00
  Immediate olist RP:     ...
  Immediate olist WC:     .0.
  Inherited olist SG:     .0.
  Inherited olist SG_RPT: .0.
  PIM include WC:        .0.

ff0e::1:1      3ffe:1:2:a::8  3ffe:1:2:a::1  SG SPT
  Upstream interface (S):  fxp0
  Upstream interface (RP): fxp0
  Upstream MRIB next hop (RP): fe80::2e0:81ff:fe01:120
  Upstream MRIB next hop (S): fe80::2e0:81ff:fe01:120
  Upstream RPF'(S,G):     fe80::2e0:81ff:fe01:120
  Upstream state:         Joined
  Join timer:             8
  KAT(S,G) running:      true
  Local receiver include WC: .0.
  Local receiver include SG: ...
  Local receiver exclude SG: ...
  Joins RP:               ...
  Joins WC:               ...
  Joins SG:               ...
  Join state:             ...
  Prune state:           ...
  Prune pending state:   ...
  I am assert winner state: ...
  I am assert loser state: ...
  Assert winner WC:      ...
  Assert winner SG:      ...
  Assert lost WC:        ...
  Assert lost SG:        ...
  Assert lost SG_RPT:    ...
  Assert tracking SG:    00.
  Could assert WC:       .0.
  Could assert SG:       .0.
```

```

I am DR:                .00
Immediate olist RP:     ...
Immediate olist WC:     .0.
Immediate olist SG:     ...
Inherited olist SG:     .0.
Inherited olist SG_RPT: .0.
PIM include WC:         .0.
PIM include SG:         ...
PIM exclude SG:        ...

```

Multicast forwarding cache

You can use `show pim mfc` and `show pim6 mfc` commands to show the multicast forwarding cache when there are multicast traffic. Below is the multicast forwarding cache at R2 when S1 is sending multicast traffic to `ff0e::1:1`.

R1

```

Xorp> show pim6 mfc
Group          Source          RP
ff0e::1:1      3ffe:1:2:a::8   3ffe:1:2:a::1
  Incoming interface :   fxp0
  Outgoing interfaces:   .0.

```

3.5 Exercise

Ex. 1: Enabling PIM multicast routing on kernel

1. As `root`, create and edit a new kernel configuration file.


```

cd /usr/src/sys/i386/conf
cp GENERIC PIMSMKERNEL
vi PIMSMKERNEL

```
2. Insert these configuration lines and save the kernel configuration.


```

options MROUTING
options PIM

```
3. Compile the kernel.


```

config PIMSMKERNEL
cd ../../compile/PIMSMKERNEL
make depend
make

```
4. If there are no problem, install the kernel.


```

make install

```
5. Restart your FreeBSD router.


```

shutdown -r now

```
6. After rebooted, confirm that the new kernel is running.


```

uname -a

```

Ex. 2: Installing XORP

1. As root, create a group named `xorp` with GID 12000.
`pw groupadd xorp -g 12000`
2. Download XORP source code.
You can download from the XORP website <http://www.xorp.org/>
We provide a local, compiled copy in this workshop.
3. Untar the package. At the command prompt, type:
`tar xzpf xorp-1.1-compile.tar.gz`
4. Run the commands below:
`cd xorp`
5. install XORP by typing:
`gmake install`
`gmake clean`

Ex. 3: Configure and operate XORP

In this exercise the instructor will show the topology and IP address assignment of the class network and you configure XORP.

1. Write down the configuration requirements for your router.
2. As root, create the XORP configuration file.
`vi /usr/local/xorp/config.boot`
3. Write the configuration based on the configuration requirements.
Enable all `traceoptions` for debugging.
4. Check your configuration.
5. Start running XORP router manger by running the command
`/usr/local/xorp/bin/xorp_rtrmgr`
Check that the router manager process is running. If you still find a configuration error, fix your configuration and run it again.

Ex. 4: Running PIM-SM with XORP: initial states

1. Log on using another vty and run the XORP CLI.
`/usr/local/xorp/bin/xorpsh`
2. At the XORP CLI, check the interfaces.
`show interfaces`
3. Read and analyze PIM interface state.
`show pim6 interface`

4. Check and analyze PIM neighbor state.
`show pim6 neighbors`
5. Check and analyze PIM MRIB.
`show pim6 mrib`
6. Check and analyze PIM bootstrap mechanism.
`show pim6 bootstrap`
If you do not see any BSR, wait for a while, then check again.
7. Check and analyze PIM Rendezvous Points.
`show pim6 rps`
8. Check and analyze PIM Join states.
`show pim6 join`
9. Check and analyze PIM MFC.
`show pim6 mfc`
10. Repeat the above procedures for PIM-SM for IPv4.

```
show pim interface
show pim neighbors
show pim mrib
show pim bootstrap
show pim rps
show pim join
show pim mfc
```

11. Discuss your findings.

Ex. 5: Running PIM-SM with XORP: Join states

1. Run `mtest6`, a program to become a IPv4 multicast listener, using another console.
`/home/instructor/bin/mtest6`
2. Join an IP multicast group on your `rl1` interface.
`j 233.18.109.151 10.10.10.1`
10.10.10.1 is the IP address of your `rl1` interface.
3. Check and analyze the PIM Join state of your router using XORP CLI.
`show pim join`
4. The instructor's PC start sending multicast traffic. Check and analyze the PIM Join state and MFC of your router.
`show pim join`
`show pim mfc`
5. Discuss your findings.

6. Leave the IP multicast group.
1 233.18.109.151 10.10.10.1
7. Check and analyze the PIM Join state and MFC of your router.
show pim join
show pim mfc
8. Discuss your findings.
9. Join an IPv6 multicast group on your r11 interface.
j6 ff08::1551 r11
10. Check and analyze the PIM Join state and MFC of your router.
show pim6 join
show pim6 mfc
11. The instructor's PC start sending multicast traffic. Check and analyze the PIM Join state and MFC of your router.
show pim6 join
show pim6 mfc
12. Discuss your findings.
13. Leave the IPv6 multicast group.
l6 ff08::1551 r11
14. Close mtest6 by typing q.
15. Check and analyze the PIM Join state and MFC of your router.
show pim6 join
show pim6 mfc
16. Discuss your findings.

Ex. 6: Installing XORP to the FreeBSD start-up configuration

1. Edit the XORP configuration file, disable all traceoptions.
2. Create the XORP start-up file. vi /usr/local/etc/rc.d/xorp.sh.
The content is

```
#!/bin/sh

case "$1" in
start)
    /usr/local/xorp/bin/xorp_rtrmgr &
    echo -n ' xorp'
    ;;
stop)
    killall xorp_rtrmgr
    echo -n ' xorp'
    ;;
*)
    exit 1
esac
```

```
*)
    echo "Usage: `basename $0` {start|stop}" >&2
    ;;
esac

exit 0
```

3. Make it executable.
`chmod +x /usr/local/etc/rc.d/xorp.sh`
4. Reboot FreeBSD.
5. Confirm that XORP is running properly.