## 13.5. Internet Multicast Protocols

### 13.5.1. Overview

Multicasting is a single point to multi-point communication strategy where data is transmitted from a single machine to a group of machines using a connectionless unreliable datagram service. Group communication primitives that provide reliability can be built on top of the multicasting service provided by the Internet. The multicasting services provided by the Internet can be deployed over a wide-area network (i.e., essentially spanning the Internet).

The Internet multicast infrastructure has components running at the end hosts and the routers. The components running at the end hosts are mostly concerned with managing the set of active receivers for the multicast group while the components running at the routers are concerned with communication overlays spanning the routers to transmit the multicast data.

### 13.5.2. Objective

The objective of this assignment is to design and implement IP multicast in GINI. Because the end hosts in GINI are emulated by user-mode Linux machines, IP standards compliant multicast support is already provided at the end hosts. Therefore, the assignment is concerned with implementing multicast functionality at the GINI routers.

The overall objective of this assignment is to design and implement a simple IP multicasting scenario. As part of this assignment, the GINI routers should be enhanced to handle IP multicasting. The enhancements at the GINI router involves two parts: processing IGMP packets for group management at GINI routers and implementing a wide-area multicast protocol such as DVMRP (distance vector multicast routing protocol).

The IGMP protocol creates and maintained soft state at the “edge” routers to indicate the composition of the membership of an active multicast group. Because user-mode Linux drives part of the IGMP messaging in this assignment, any simplification should conform to the IGMP standards. One suggested simplification is to use version 1 or 2 of IGMP instead of version 3 that is used as default in Linux. This simplifies the handshake protocol that should be implemented the GINI routers to manage the soft state regarding the group.

In the second part, a DVMRP-like protocol is implemented to create and maintain a multicast tree that spans the edge routers that have end hosts in the multicast group. Suitable assumptions can be made to simplify the DVMRP-like protocol (i.e., the protocol implemented to build the multicast tree need not be standards compliant at all).

### 13.5.3. Background Material

Multicasting is an important routing technique. However, due to various reasons it is not yet widely used on the Internet. One of the key advantages of the multicasting is that it avoids duplicate packet transfers by having the routers duplicate the packets such that packets are distributed as and when required. The benefits of multicasting become apparent with real-time multimedia applications such as video or audio conferencing that send out significant amount of packets. Using multicasting we can significantly reduce the load placed on a network and provide clients with the same or better service.

Although reliable multicasting is heavily researched, the standard IP multicasting provided an unreliable datagram service very much like the UDP. Applications can build their own reliable data transfer protocol on top of this service. This assignment is aimed at giving you some insights into how IP multicasting works.

### 13.5.4. Description

Consider the simplest multicast scenario, where several machines are connected to a router, as in Figure 13.2. The listeners need to "subscribe" to the multicast group before they receive data that is being sent by the sender to that multicast group. If no receivers are subscribed to a particular group, the router does not forward the incoming packet at all.

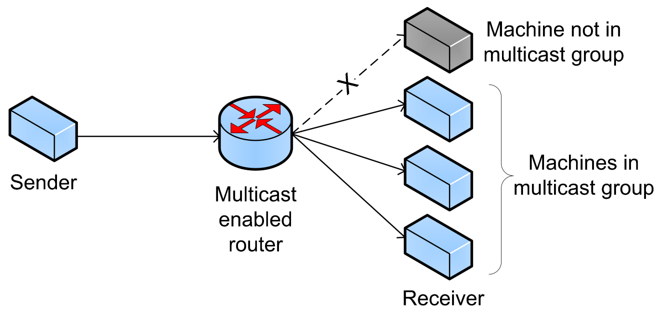

**Figure 13.2:** The simplest IP multicasting scenario involving a single router.

A C program that implements the sender (called `sender.c`) is shown below. The source of this program is included in GiniLinux-fs and can be found in `/root/gini_book/chpt9/multicast/`.

```
/*
 * sender.c -- multicasts "hello, world!" to a multicast group once a second
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

/* define multicast group parameters */
#define DST_PORT        12345
#define DST_GROUP       "225.0.0.36"

int main(int argc, char ** argv)
{
        struct sockaddr_in addr;
        int fd;
        char * msg = "Hello,␣World!";

        /* create a UDP socket */
        if ((fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                perror("socket");
                exit(1);
        }

        /* setup the destination address
         * this is no different from any ordinary UDP program
         */
        memset(&addr, 0, sizeof(addr));
        addr.sin_family=AF_INET;
        addr.sin_addr.s_addr = inet_addr(DST_GROUP);
        addr.sin_port = htons(DST_PORT);

        /* send to destination
         * it is a multicast address
         * repeat every 1 second
         */
        for (;;) {
                if (sendto(fd, msg, strlen(msg), 0, (struct sockaddr *) &addr,
                                sizeof(addr)) < 0) {
                        perror("sendto");
                        exit(1);
                }
                sleep(1);
        }
        return 0;
}
```

It should be noted that the program does not have any multicast specifics. It is a normal UDP program that sends data to a IP address that is in class D (an IPv4 multicast address). It is the responsibility of the IP router that is connected to the subnet to relay the packet generated by the above program to the appropriate destination machines (members of the group).

A C program implementing the listener (`listener.c`) is shown below. The source code for the program can be found in the same location as `sender.c`.

```
/*
 * listener.c -- joins the given multicast group and prints all
 * incoming messages on the screen
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <stdio.h>

/* define the multicast group parameters */
#define SRC_PORT        12345
#define SRC_GROUP       "225.0.0.36"

#define MSGBUFSIZE      256

/* This is the UPD receiver routine.
 * It joins the given multicast group using the setsockopt() routine.
 * As a result the host will receive all messages that are sent to the
 * multicast group
 */
int main(int argc, char ** argv)
{
        struct sockaddr_in addr;
        int fd;
        int nbytes;
        int addrlen;
        int allow_reuse;
        struct ip_mreq mreq;
        char msgbuf[MSGBUFSIZE];

        /* create a UDP socket */
        if ((fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                perror("socket");
                exit(1);
        }

        /* Setup the receiver address
         * Nothing specific to multicast here!
         * just setup the receiver port and IP address
         */
        memset(&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_addr.s_addr = htonl(INADDR_ANY); /* N.B.: differs from sender */
        addr.sin_port = htons(SRC_PORT);

        /* bind to receive address */
        if (bind(fd, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
                perror("bind");
                exit(1);
        }

        /* use setsockopt() to request that the kernel join the multicast group */
        mreq.imr_multiaddr.s_addr = inet_addr(SRC_GROUP);
        mreq.imr_interface.s_addr = htonl(INADDR_ANY);
        if (setsockopt(fd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq)) < 0) {
                perror("setsockopt");
                exit(1);
        }

        /* now just enter a read-print loop */
        for (;;) {
                addrlen = sizeof(addr);
                if ((nbytes = recvfrom(fd, msgbuf, MSGBUFSIZE, 0,
                                (struct sockaddr *) &addr, &addrlen)) < 0) {
                        perror("recvfrom");
                        exit(1);
                }
                puts(msgbuf);
        }
        return 0;
}
```

When the listener executes the setsockopt() routine, an IGMP report is generated by the host. This IGMP report is then multicast to the target group. Consider the target group with the class-D address of `225.0.0.36`. As shown in Figure **??**, the end host transmits an IGMP report to `225.0.0.36`. The IGMP packet will encapsulated by an IP header and carried over the Ethernet. The Ethernet needs a destination hardware address to packet delivery. For the multicast packets, the Ethernet hardware addresses are derived from the multicast addresses as shown in Figure 13.4.

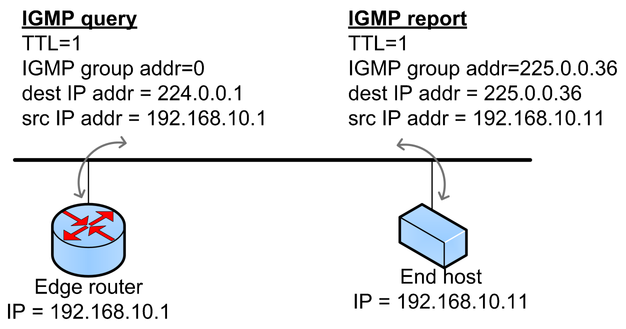

**Figure 13.3:** IGMP query and report messaging.

Because the IGMP report sent by the host be can be lost, the host keeps repeating the message after a random time interval, usually between 0 and 10 seconds. When the router receives an IGMP report, it responds with an IGMP query message. When the host receives an IGMP query, it does not respond immediately. It schedules a response for a future time that is $t$ away from the current time ($t$ is randomly chosen from a given interval). This randomization of the response time, prevents a flood of reports from multiple routers that are part of the group. The router needs just one report to continue to maintain the *state* information for the multicast group that was setup when the first report reached the router. With the random intervals chosen by the hosts for sending their respective reports, hosts can detect prior reports for the same group and cancel their own next scheduled IGMP report generation. If a router does not receive any reports for a while, it will assume that no hosts are interested in the multicast group and the corresponding group entry will be timed out.

As mentioned above, multicast packets should be picked up by all interested parties (routers and hosts) that are connected by an Ethernet LAN. In an Ethernet segment, the packets are picked only if the hardware address in the incoming packet (frame) matches the interface's hardware address. An exception is made for frames with multicast hardware addresses. They are picked if by the interface if there is local interest. Figure 13.4 shows how the Ethernet multicast address can be derived from the IP multicast address. It should be noted that 5 bits are not defined by the multicast address. Therefore, we can have collisions that should be resolved by examining the destination IP address.

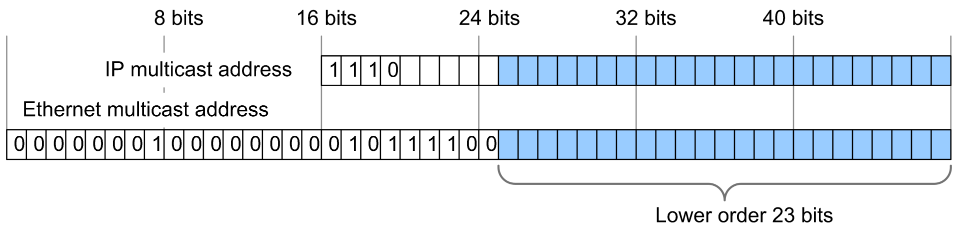

**Figure 13.4:** Relation between multicast Ethernet and IP addresses.

As usual several special multicast addresses are allocated by IANA. Following is a sampling of this list that can be useful for this assignment.

| | | |
|---|---|---|
| 224.0.0.1 | All Systems on this Subnet | [RFC1112,JBP] |
| 224.0.0.2 | All Routers on this Subnet | [JBP] |
| 224.0.0.4 | DVMRP Routers | [RFC1075,JBP] |
| 224.0.0.13 | All PIM Routers | [Farinacci] |

The format of an IGMP message is given below. The checksum is computed following the same procedure used for ICMP.

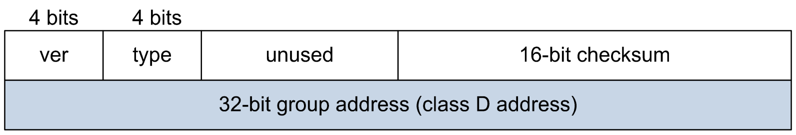

**Figure 13.5:** IGMP message header.

IGMP version 1 is the recommended protocol for this assignment. Because user-mode Linux is capable of handling all three versions of IGMP, you need to use the `sysctl` command as below to set the IGMP version to 1.

```
sysctl -w net.ipv4.conf.all.force_igmp_version=1
```

With IGMP version 1, three types of messages are supported and two of them are relevant here: IGMP report and IGMP query. To send a IGMP query and report, set the type field to 1 and 2, respectively.

The first part of the assignment is to get the simple topology shown in Figure 13.2 with one router working. That is, the listeners should be able to see the "Hello, World!" message that is being multicasted by the sender program. You need to program the router such that it distributes the message to all interested hosts. A host registers to the multicast group by sending an IGMP report message.

The second part of the assignment is to extend the topology in Figure 13.2 to the one in Figure 13.6. Because multiple routers are involved in the topology given in Figure 13.6, a "distribution" tree creating and maintaining protocol is necessary to perform multicast over this topology. Design a multicast tree building protocol based on the *reverse path flood and prune* algorithm. The tree building protocol can use the underlying static routes to determine the best paths to reach a particular destination. For example, the static routes can be used to evaluate whether a flood packet carrying a multicast announcement is reaching a particular along the link that would be used by the reverse shortest path.

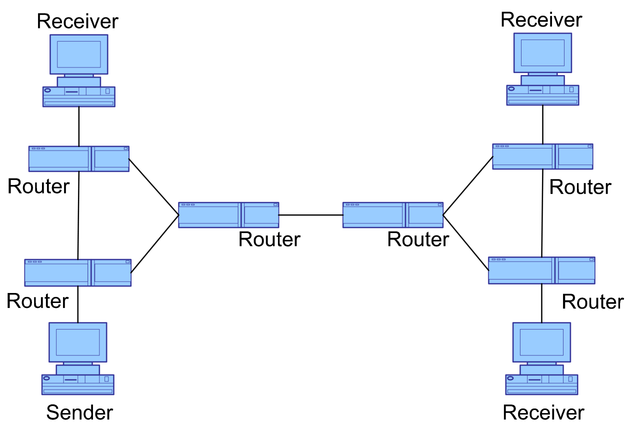

**Figure 13.6:** A more complex network topology for testing wide-area multicast.

## 13.5.5. Suggested Implementation Plan

1. In this assignment, the GINI router should be enhanced to handle IP multicasting. The assignment is done in two parts.

2. In the first part, the topology has a single IP router. The group management functions provided by IGMP is sufficient to perform multicast in this scenario. Design and implement the improvements to the GINI router according to IGMP version 1. Setup the user-mode Linuxes such that they use IGMP version 1 as the default protocol.

3. Test the multicast implementation using the `sender` and `listener` provided with the assignment. The sender injects UDP packets to a multicast group and the listeners run on a subset of machines connected the LANs associated with the IP router. The packet injected by the sender should be simultaneously delivered all running listeners.

4. In this second part, the larger topology is used with multiple IP routers. The IGMP protocol is used by the end host to join the multicast group. In this step, different end hosts can join different routers depending on this locality in the network. For one end host in the multicast group to reach any other end host within the group, the routers part of the group should be interconnected. This interconnection is performed by the distribution tree. Design and implement the distribution tree creation and maintenance routines. Test the multicast functionality following the same procedure the previous case.

### 13.5.6. Expected Deliverables

1. IGMP enhanced GINI router.
2. Distribution tree protocol to span many GINI routers. This protocol need not be standards compliant.
3. Demonstration of IGMP with single router network topologies.
4. Demonstration of the distribution tree with multiple router network topologies.