Redes de Comunicação 2021/2022

TP08
IP Multicast

Vasco Pereira
University of Coimbra

TP08: IP Multicast addresses

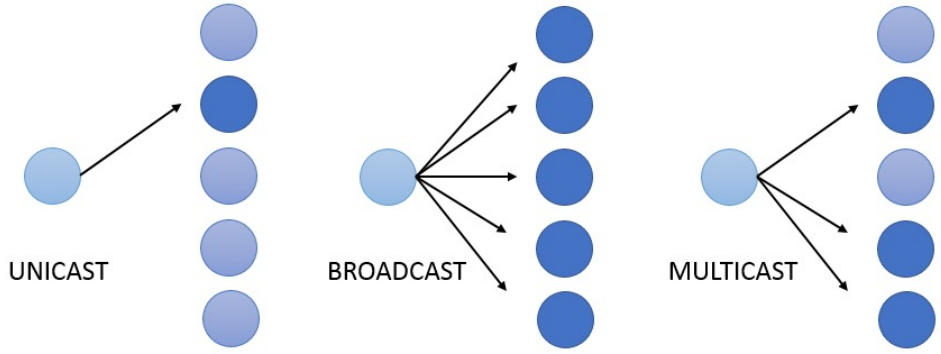
Overview:

- Communications with IP (multicast)
- IPv4 classful addressing
- Network programming (Linux)

Communications with IP (multicast)

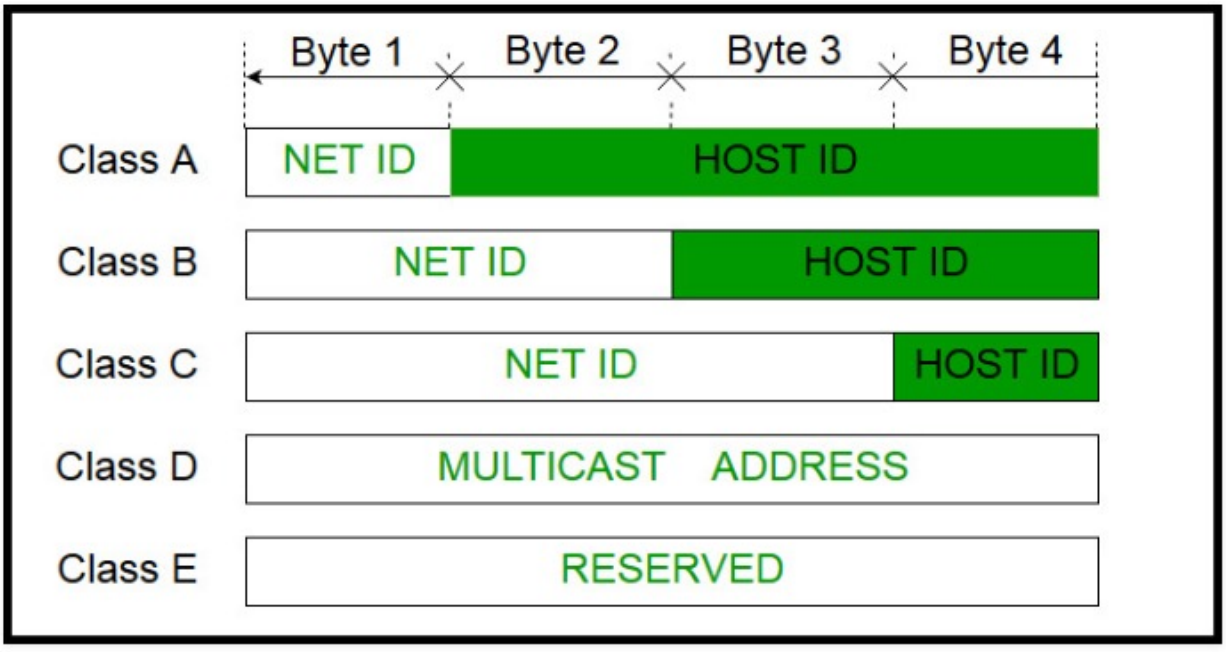
IP SUPPORTS THE FOLLOWING SERVICES:

- ONE TO ONE (UNICAST)
- ONE TO ALL (BROADCAST)
- ONE TO SEVERAL (MULTICAST)



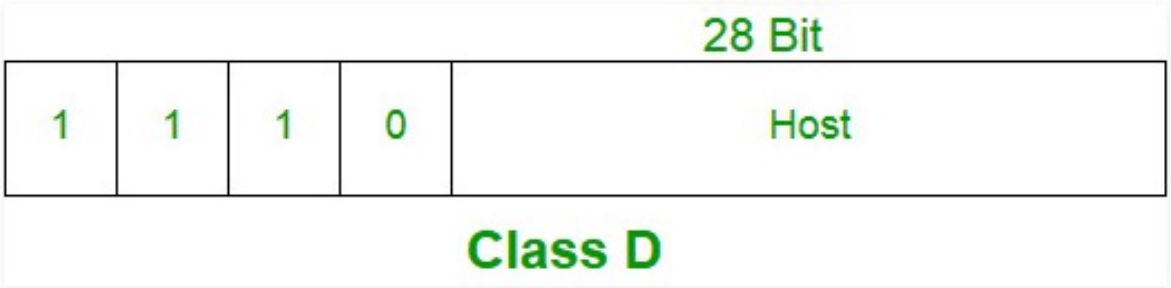
IP MULTICAST ALSO SUPPORTS A MANY TO MANY SERVICE
IP MULTCAST REQUIRES SUPPORT OF OTHER PROTOCOLS (IGMP, MULTICAST ROUTING)

IPv4 classful addressing



IPv4 classful addressing

- **Class D** IPv4 addresses:
  - Reserved for multicast communications
  - Higher order bits of the first byte is always 1110
  - From 224.0.0.0 to 239.255.255.255



Multicast addresses

- Addresses in the range of 224.0.0.0 to 224.0.0.255 are individually assigned by IANA and designated for multicasting on the local subnetwork only.
  - E.g., Routing Information Protocol (RIPv2) uses 224.0.0.9, Open Shortest Path First (OSPF) uses 224.0.0.5 and 224.0.0.6, and Multicast DNS uses 224.0.0.251.
  - Routers must not forward these messages outside the subnet from which they originate.
  - DO NOT USE THESE ADDRESSES!

Network Programming (Linux)

- **IP multicasting** provides the capability for an application to send a single IP datagram that a group of hosts in a network can receive. The hosts in the group may reside on a single subnet or may be on different subnets (connected by multicast capable routers).
- **Hosts may join and leave groups at any time**. There are no restrictions on the location or number of members in a host group. A class D Internet address in the range 224.0.0.1 to 239.255.255.255 **identifies a host group**.
- An application program can send or receive multicast datagrams by using the socket() API and connectionless SOCK_DGRAM type sockets.
- When a socket of type SOCK_DGRAM is created, an application can use the **setsockopt()** function to control the multicast characteristics associated with that socket.

Network Programming (Linux)

The **setsockopt()** function accepts the following IPPROTO_IP level flags:

- IP_ADD_MEMBERSHIP: Joins the multicast group specified
- IP_DROP_MEMBERSHIP: Leaves the multicast group specified
- IP_MULTICAST_IF: Sets the interface over which outgoing multicast datagrams are sent
- IP_MULTICAST_TTL: Sets the Time To Live (TTL) in the IP header for outgoing multicast datagrams. By default it is set to 1. TTL of 0 are not transmitted on any sub-network. Multicast datagrams with a TTL of greater than 1 may be delivered to more than one sub-network, if there are one or more multicast routers attached to the first sub-network.
- IP_MULTICAST_LOOP: Specifies whether or not a copy of an outgoing multicast datagram is delivered to the sending host as long as it is a member of the multicast group.

Example: sending a multicast datagram

1. Create an AF_INET, SOCK_DGRAM type socket
2. Initialize a *sockaddr_in* structure with the destination group IP address and port number
3. Set the IP_MULTICAST_LOOP socket option according to whether the sending system should receive a copy of the multicast datagrams that are transmitted
4. Set the IP_MULTICAST_IF socket option to define the local interface over which you want to send the multicast datagrams
5. Send the datagram

Example: receiving a multicast datagram

1. Create an AF_INET, SOCK_DGRAM type socket
2. Set the SO_REUSEADDR option to allow multiple applications to receive datagrams that are destined to the same local port number
3. Use bind() to specify the local port number. Specify the IP address as INADDR_ANY in order to receive datagrams that are addressed to a multicast group
4. Use the IP_ADD_MEMBERSHIP socket option to join the multicast group that receives the datagrams. When joining a group, specify the class D group address along with the IP address of a local interface.
5. Receive the datagram

Configurate multicast routing in IOS routers

- Every interface in every router, must enable multicast
  - Example:
    - Enabling multicast in interface Ethernet 0/0 and FastEthernet 1/0

```
enable
configurate terminal        // enters configuration mode
ip multicast-routing        // Enables IP multicast routing.

interface Ethernet 0/0
ip pim sparse-dense-mode    // default-mode
exit

interface FastEthernet 1/0
ip pim sparse-dense-mode
exit
```

- Note: To disable PIM on an interface, use the **no ip pim** interface configuration command.
- Note 2: In populating the multicast routing table, dense-mode interfaces are always added to the table. Sparse-mode interfaces are added to the table only when periodic join messages are received from downstream devices or when there is a directly connected member on the interface.

Example of multicast in C

- Example (1/4)
  - (Adapted from https://web.cs.wpi.edu/~claypool/courses/4514-B99/samples/multicast.c)

```
/*
The following program sends or receives multicast packets. If invoked
with one argument, it sends a packet containing the current time to an
arbitrarily chosen multicast group and UDP port. If invoked with no
arguments, it receives and prints these packets. Start it as a sender on
just one host and as a receiver on all the other hosts
*/

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

Example of multicast in C

- Example (2/4)

```
#define EXAMPLE_PORT 6000
#define EXAMPLE_GROUP "239.0.0.1"

int main(int argc, char *argv[])
{
   struct sockaddr_in addr;
   int addrlen, sock, cnt;
   struct ip_mreq mreq;
   char message[50];

   /* set up socket */
   sock = socket(AF_INET, SOCK_DGRAM, 0);
   if (sock < 0) {
     perror("socket");
     exit(1);
   }

   int multicastTTL = 255; // by default TTL=1; the packet is not transmitted to other networks
   if (setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL, (void *) &multicastTTL, sizeof(multicastTTL)) < 0){
            perror("socket opt");
            exit(1);
    }

   bzero((char *)&addr, sizeof(addr));
   addr.sin_family = AF_INET;
   addr.sin_addr.s_addr = htonl(INADDR_ANY);
   addr.sin_port = htons(EXAMPLE_PORT);
   addrlen = sizeof(addr);
```


Example of multicast in C

- Example (3/4)

```
if (argc > 1) {
     // multicast sender - sends messages to a multicast address

     addr.sin_addr.s_addr = inet_addr(EXAMPLE_GROUP);
     while (1) {
            time_t t = time(0);
            sprintf(message, "time is %-24.24s", ctime(&t));
            printf("sending: %s\n", message);
            cnt = sendto(sock, message, sizeof(message), 0, (struct sockaddr *) &addr, addrlen);
            if (cnt < 0) {
               perror("sendto");
               exit(1);
            }
            sleep(5);
     }
```

Example of multicast in C

- Example (4/4)

```
} else {
      // multicast receiver - receives multicast messages

      if (bind(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
         perror("bind");
             exit(1);
      }
      mreq.imr_multiaddr.s_addr = inet_addr(EXAMPLE_GROUP);
      mreq.imr_interface.s_addr = htonl(INADDR_ANY);
      if (setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq)) < 0) {
             perror("setsockopt mreq");
             exit(1);
      }
      while (1) {
             cnt = recvfrom(sock, message, sizeof(message), 0,
                                 (struct sockaddr *) &addr, (socklen_t *)&addrlen);
             if (cnt < 0) {
                perror("recvfrom");
                exit(1);
             } else if (cnt == 0) {
                break;
             }
             printf("%s: message = \"%s\"\n", inet_ntoa(addr.sin_addr), message);
        }
    }
}
```


TP08: Summary

What have we covered here?:

- Communications with IP (multicast)
- IPv4 classful addressing
- Network programming (Linux)