Multicast IP

- A **unicast** address identifies a **single interface**. A **broadcast** address identifies **all interfaces** on the **subnet**. A **multicast** address identifies a **set of interfaces** on the **Internet**.

- Suppose that thousands of users, located throughout the world, want to receive over the Internet a live video program (e.g., a live sporting event).

- How should the video be delivered from the source to all of the users? The most straightforward approach is to execute thousands of simultaneous **unicasts**, that is, send a separate packet stream to each of the users.

- This approach is straightforward because it could be achieved with traditional IP: each stream would have a unique destination IP address for its packets.

- But having the source send multiple simultaneous unicasts has severe drawbacks:

  1. It is wasteful of link bandwidth because over a link multiple copies of the data are sent. Generally speaking, the closer a link is to root of the distribution, the greater the number of identical packets that are sent over it.

  2. It can place a tremendous burden on the server (i.e., the source). The server will have to stream out multiple streams, perhaps thousands of streams, simultaneously.

- Multicast addresses are assigned dynamically by network protocols.

- Recall that **class D** IP addresses (**224.0.0.0** to **239.255.255.255**) do not identify individual interfaces in an Internet but instead identify groups of interfaces.

- Therefore, **class D** addresses are called **multicast groups**. A datagram with a class D destination address is delivered to every interface on the Internet that has **joined** the corresponding **multicast group**.

- Applications that use multicasting include Real Audio and video, video conferencing and whiteboard applications.

- An alternative approach, called multicast IP, is for the network to create a multicast tree which originates from the server and terminates at the receiving hosts.

- The server sends out only one stream of packets, with each of these packets carrying a special multicast address in the destination IP address field. Whenever one of these packets reaches a router that has two or more out-going branches in the multicast tree, the router makes multiple copies of the packet -- one for each branch.

- As compared with sending multiple unicast IP streams, multicast IP places less stress on the server and more efficiently utilizes network resources.

- The **network** will use a **routing protocol** to establish a **multicast tree** from the source to all the all of receivers that wish to participate in the session.

Addressing

- Multicast addresses range from 224.0.0.0 to 239.255.255.255 . Each multicast session is assigned an address from this field. Each host participating in a particular multicast session is assigned the multicast address of the session.

- Multicast sessions are announced throughout the Internet so that hosts can be made aware of them and join them.

- This **advertising** is done with an **application-layer protocol**, called the **Session Description Protocol** (**SDP**). This protocol sends the advertisements over a special multicast session with the address **224.2.2.2** and uses **UDP port 4000**.

- For each multicast session, the SDP advertises the session's **name**, **active times**, **type of media** (audio, video, whiteboard, etc.) and **multicast address**.

- Any host that is interested in learning about the multicast sessions in progress (or will be in progress in the near future) can join the special multicast session and receive the advertisements.

- A host runs a **user agent** to collect the advertisements and to display a summary of them graphically to the user. One commonly used user agent is called **Session Directory** (**SD**).

- **SD** actually does more than display announcements of multicast groups; it also launches multicast applications and selects an address for any new multicast session.

- How are addresses assigned to multicast sessions? When a source begins a new multicast session, its SD chooses a multicast address at random from the multicast address space.

Host Participation

- For a host to join a multicast session, it must do two things. First, it must immediately send a message to its **nearest router** to inform the router that it wants to join the multicast session; an **application-layer protocol**, called **the Internet Group Management Protocol** (**IGMP**), is used to send the message.

- Second it will need to set its IP process to **accept** the IP datagrams which have the session's **multicast address** in the **IP destination address field**. A similar process takes place when a host wants to quit a multicast session.

The IP Multicast API

- IP multicasting is currently supported only on **AF_INET** sockets of type **SOCK_DGRAM** and **SOCK_RAW**, and only on subnets for which the router has been enabled to support multicasting.

- The **SOCK_RAW** option specifies a **raw socket**. A raw socket allows your program to **bypass** the TCP/IP **transport-layer functions** and access **low-level protocols** such as **ICMP**.

- As such, your program must perform the normal transport-layer functions, such as data encapsulation, for the IP layer.

- Note that you have to be root to create a raw socket. This is to prevent normal users from writing their own IP datagrams to the network.

Sending IP Multicast Datagrams

- To send a multicast datagram, specify an **IP multicast address** in the range **224.0.0.0** to **239.255.255.255** as the **destination address** in a **sendto()** call.

- By default, IP multicast datagrams are sent with a **time-to-live** (**TTL**) of 1, which prevents them from being forwarded beyond a single subnetwork.

- A new socket option allows the TTL for subsequent multicast datagrams to be set to any value from 0 to 255, in order to control the scope of the multicasts:

```
u_char ttl;

setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl))
```

- Multicast datagrams with a **TTL of 0** will not be transmitted on any subnet, but may be delivered locally if the sending host belongs to the destination group and if multicast loopback has not been disabled on the sending socket.

- Multicast datagrams with TTL greater than one may be delivered to more than one subnet if there are one or more multicast routers attached to the first-hop subnet.

- To provide meaningful scope control, the multicast routers support the notion of TTL "**thresholds**", which prevent datagrams with less than a certain TTL from traversing certain subnets.

- The thresholds enforce the following convention:

  multicast datagrams with initial TTL 0 are restricted to the same host
  multicast datagrams with initial TTL 1 are restricted to the same subnet
  multicast datagrams with initial TTL 32 are restricted to the same site

multicast datagrams with initial TTL 64 are restricted to the same region
multicast datagrams with initial TTL 128 are restricted to the same continent
multicast datagrams with initial TTL 255 are unrestricted in scope.

- "Sites" and "regions" are not strictly defined, and sites may be further subdivided into smaller administrative units, as a local matter (depending on how the routers are configured).

- An application may choose an initial TTL other than the ones listed above. For example, an application might perform an "**expanding-ring search**" for a network resource by sending a multicast query, first with a **TTL of 0**, and then with larger and larger TTLs, until a reply is received, perhaps using the TTL sequence 0, 1, 2, 4, 8, 16, 32.

- The multicast router accompanying this release refuses to forward any multicast datagram with a destination address between 224.0.0.0 and 224.0.0.255, inclusive, regardless of its TTL.

- This range of addresses is reserved for the use of routing protocols and other low-level topology discovery or maintenance protocols, such as gateway discovery and group membership reporting.

- Each multicast transmission is sent from a single network interface, even if the host has more than one multicast-capable interface. (If the host is also serving as a multicast router, a multicast may be FORWARDED to interfaces other than originating interface, provided that the TTL is greater than 1.)

- A socket option is available to override the default for subsequent transmissions from a given socket:

```
struct in_addr addr;

setsockopt(sock, IPPROTO_IP, IP_MULTICAST_IF, &addr, sizeof(addr));
```

- where "**addr**" is the local IP address of the desired outgoing interface.

- An address of **INADDR_ANY** may be used to revert to the default interface.

- The **local IP address** of an interface can be obtained via the **SIOCGIFCONF** flag in an **ioctl** call.

- To determine if an interface supports multicasting, fetch the interface flags via the **SIOCGIFFLAGS** ioctl and see if the **IFF_MULTICAST** flag is set.

- Normal applications should not need to use this option; it is intended primarily for multicast routers and other system services specifically concerned with internet topology.

- If a multicast datagram is sent to a group to which the sending host itself belongs (on the outgoing interface), a copy of the datagram is, by default, looped back by the IP layer for local delivery.

- Another socket option gives the sender explicit control over whether or not subsequent datagrams are looped back:

```
        u_char loop;

        setsockopt(sock, IPPROTO_IP, IP_MULTICAST_LOOP, &loop,
sizeof(loop));
```

- Here "**loop**" is **0** to disable loopback, and **1** to enable loopback.

- This option provides a performance benefit for applications that may have no more than one instance on a single host (such as a router or a mail demon), by eliminating the overhead of receiving their own transmissions.

- It should generally not be used by applications for which there may be more than one instance on a single host (such as a conferencing program) or for which the sender does not belong to the destination group (such as a time querying program).

- A multicast datagram sent with an initial TTL greater than 1 may be delivered to the sending host on a different interface from that on which it was sent, if the host belongs to the destination group on that other interface. The loopback control option has no effect on such delivery.

<u>Receiving IP Multicast Datagrams</u>

- Before a host can receive IP multicast datagrams, it must become a member of one or more IP multicast groups.

- A process can ask the host to join a multicast group by using the following socket option:

```
    struct ip_mreq mreq;

    setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq,
sizeof(mreq));
```

- Here "**mreq**" is the following structure:

```
    struct ip_mreq
    {
            struct in_addr imr_multiaddr;    /* multicast group to join */
            struct in_addr imr_interface;    /* interface to join on    */
    }
```

- Every membership is associated with a single interface, and it is possible to join the same group on more than one interface.

- "**imr_interface**" should be **INADDR_ANY** to choose the default multicast interface, or one of the host's local addresses to choose a particular (multicast-capable) interface.

- Up to **IP_MAX_MEMBERSHIPS** (currently 20) memberships may be added on a single socket.

- To drop a membership, we use:

```
    struct ip_mreq mreq;

    setsockopt(sock, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq,
sizeof(mreq))
```

- Where "**mreq**" contains the same values as used to add the membership.

- The memberships associated with a socket are also dropped when the socket is closed or the process holding the socket is killed.

- However, more than one socket may claim a membership in a particular group, and the host will remain a member of that group until the last claim is dropped.

- The memberships associated with a socket do not necessarily determine which datagrams are received on that socket.

- Incoming multicast packets are accepted by the kernel IP layer if any socket has claimed a membership in the destination group of the datagram; however, delivery of a multicast datagram to a particular socket is based on the destination port (or protocol type, for raw sockets), just as with unicast datagrams.

- To receive multicast datagrams sent to a particular port, it is necessary to bind to that local port, leaving the local address unspecified (i.e., **INADDR_ANY**).
- More than one process may bind to the same **SOCK_DGRAM** UDP port if the **bind()** is preceded by:

```
int one = 1;

setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(one));
```

- In this case, every incoming multicast or broadcast UDP datagram destined to the shared port is delivered to all sockets bound to the port.
- For backwards compatibility reasons, this does not apply to incoming unicast datagrams -unicast datagrams are never delivered to more than one socket, regardless of how many sockets are bound to the datagram's destination port. **SOCK_RAW** sockets do not require the **SO_REUSEADDR** option to share a single IP protocol type.
- The definitions required for the new, multicast-related socket options are found in <**netinet/in.h**>. All IP addresses are passed in **network byte-order**.
- A final multicast-related extension is independent of IP: two new **ioctls**, **SIOCADDMULTI** and **SIOCDELMULTI**, are available to add or delete link-level (e.g., Ethernet) multicast addresses accepted by a particular interface.
- The address to be added or deleted is passed as a **sockaddr** structure of family **AF_UNSPEC**, within the standard **ifreq** structure. These ioctls are for the use of protocols other than IP, and require superuser privileges.
- A link-level multicast address added via **SIOCADDMULTI** is not automatically deleted when the socket used to add it goes away; it must be explicitly deleted.