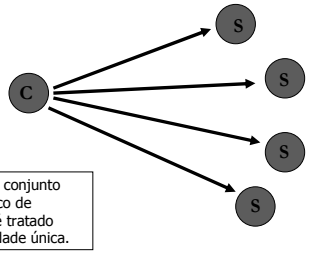
Multicast: Comunicação em Grupo

Multicast: Comunicação em Grupo



Um grupo é um conjunto dinâmico/estático de processos que é tratado como uma entidade única.

Onde é útil usar Multicasting?

- Servidores redundantes (fault-tolerance)
- Discovery services (spontaneous networking)
- Melhorar o desempenho com dados replicados
- Propagação de notificação de eventos

MCast a vários níveis

- Nível físico: Ethernet
- Nível rede: IP multicasting
- Sistema Operação: grupo de processos
- Middleware: Isis, Horus, Consul
- Plataformas Publish/Subscribe.

IP Multicast

- Routers interpret any datagram sent to an IP address in the range of: **224.0.0.0 to 239.255.255.255** as **multicast.**
- Any IP application with a UDP socket can send to a multicast address, with some limitations.
- Applications that join a **multicast group** can receive multicast datagrams sent to that group address.

Receiving/Sending Multicasts

A multicast receiver application must:
1. Get a UDP socket.
2. Bind to the application's port number (name the socket).
3. Join the application's multicast address group.
4. Receive.
5. Close the socket when complete.

A multicast sender must:
1. Get a UDP socket.
2. Set the IP Time-To-Live appropriately.
3. Send to the application's multicast address and port number.
4. Close the socket when complete.

//The sender does not have to join a multicast group.

IP Multicast

- When a multicast message arrives at a computer, copies are forwarded to all of the local sockets that have joined the specific multicast address and the specified port number.
- The membership of multicast groups is dynamic: member can join and leave whenever they want.
- IP multicast is available only via UDP.

Multicast Routers

- IP packets can be multicasted on a local network and on the wider Internet.
- Local multicasts make use of the hardware multicast feature of Ethernet (single message).
- Internet multicasts make use of multicast routers, which forward single datagrams to routers on other networks with members, where they are again hardware multicasted to local members.

TTL: time-to-live

- To limit the distance of propagation of a multicast datagram, the sender specifies the number of routers it is allowed to pass:

  Time-to-Live TTL

- **The default value is 1.**
- **This allows the multicast propagate only on the local network.**

Multicast Address Allocation

- Permanent Multicast Groups:
  224.0.0.1 ..... 224.0.0.255

- Temporary Groups:
  225.0.0.1 ..... 239.255.255.255

Temporary Multicast Group

- When a multicast group is created it requires a free multicast address to avoid accidental participation in an existing group.
- The IP multicast protocol does not address this issue.
- When the users only need to communicate locally they set the TTL to a small value.
- However, programs using IP multicasting over the Internet require a solution to the problem (directory of multicast sessions).

Internet Multicasting

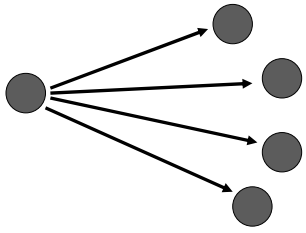
- Most providers of push media currently use unicast to deliver content to their customers. This means that each server sends content to each listener in a stream of individually addressed datagrams.
- This works, but does not scale well.

- **Multicast** allows a content provider to send a single datastream to a single address, a datastream that network routers subsequently distribute to as many receivers as desired.
- **Multicast** requires no additional effort on the part of the sender to add new receivers, since the network handles the distribution from the single datastream.

One-to-Many Multicasting

- Real-time data distribution (weather, stocks, telemetry, and remote sensing);
- File distribution (software updates, database mirrors, and web caching);
- Cryptographic key distribution;
- Network management;
- System configuration;

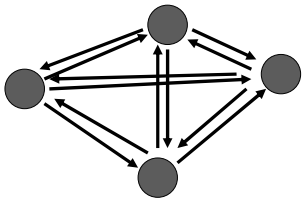
One-to-Many Multicast



Many-to-Many Multicasting

- Conferencing (video, audio, and whiteboard sharing);
- Collaborative document sharing;
- Interactive distance learning;
- Virtual reality

Many-to-Many



Multicast Datagram Socket

- The multicast datagram socket class is useful for sending and receiving IP multicast packets.
- A **MulticastSocket** is a (UDP) DatagramSocket, with additional capabilities for joining "groups" of other multicast hosts on the Internet.
- A multicast group is specified by a class D IP address and by a standard UDP port number.
- Class D IP addresses are in the range: **224.0.0.0 to 239.255.255.255.**

Multicast peer joins a group and sends and receives datagrams

```
import java.net.*;
import java.io.*;

public class MulticastPeer{
    public static void main(String args[]){

     // args give message contents & destination multicast group (e.g. "228.5.6.7")

    MulticastSocket s =null;
     try {
            InetAddress group = InetAddress.getByName(args[1]);
            s = new MulticastSocket(6789);
            s.joinGroup(group);

            byte [] m = args[0].getBytes();
            DatagramPacket messageOut =
                        new DatagramPacket(m, m.length, group, 6789);
            s.send(messageOut);
     // this figure continued on the next slide
```

Exemplo (cont).

```
// get messages from others in group
        byte[] buffer = new byte[1000];
        for(int i=0; i< 3; i++) {
            DatagramPacket messageIn =
                  new DatagramPacket(buffer, buffer.length);
            s.receive(messageIn);
            System.out.println("Received:" + new
    String(messageIn.getData()));
        }
        s.leaveGroup(group);
      }catch (SocketException e){System.out.println("Socket: " +
    e.getMessage());
      }catch (IOException e){System.out.println("IO: " +
    e.getMessage());}
    }finally {if(s != null) s.close();}
    }
}
```


Ex: Receiving Socket

```
import sun.net.*;
import java.net.*;
int port = 5000;
group = "225.4.5.6";
// Create the socket and bind it to port 'port'.
MulticastSocket s = new MulticastSocket(port);
// join the multicast group
s.joinGroup(InetAddress.getByName(group));
// Now the socket is set up and we are ready to receive packets
byte buf[] = byte[1024];
DatagramPacket pack = new DatagramPacket(buf, buf.length);
s.receive(pack);
// Do something useful with the data we just received,
System.out.println("Received data from: " + pack.getAddress().toString() + ":" +
    pack.getPort() + " with length: " + pack.getLength());
    System.out.write(pack.getData(),0,pack.getLength()); System.out.println();
// And when we have finished receiving data leave the multicast group and close the
    socket
s.leaveGroup(InetAddress.getByName(group));
s.close();
```


Ex: Sending Socket

```
import sun.net.*;
import java.net.*;
// Which port should we send to
int port = 5000;
// Which address
String group = "225.4.5.6";
// Which ttl
int ttl = 1;
// Create the socket but we don't bind it as we are only going to send data
MulticastSocket s = new MulticastSocket();
// We don't have to join the multicast group if we are only sending data and not
      receiving
// Fill the buffer with some data
byte buf[] = byte[10];
for (int i=0; i<buf.length; i++)
       buf[i] = (byte)i;
// Create a DatagramPacket
DatagramPacket pack = new DatagramPacket(buf, buf.length,
      InetAddress.getByName(group), port);
// Do a send. Note that send takes a byte for the ttl and not an int.
s.send(pack,(byte)ttl);
// And when we have finished sending data close the socket
s.close();
```


Java: Objectos/Métodos para usar Multicasting

MulticastSocket(); Create a multicast socket.

MulticastSocket(int port); Create a multicast socket in a specific port.

int getTimeToLive(); Get the default time-to-live for multicast packets.

void joinGroup(InetAddress mcastaddr); Joins a multicast group.

void leaveGroup(InetAddress mcastaddr); Leave a multicast group.

void send(DatagramPacket p, byte ttl); Sends a datagram packet to the destination, with a TTL (time- to-live).

void setTimeToLive(int ttl); Set the default time-to-live for multicast packets.

Joining/Leaving Mcast Groups

- When one sends a message to a multicast group, **all** subscribing recipients to that host and port receive the message (within the time-to-live range of the packet).
- The socket needn't be a member of the multicast group to send messages to it.
- When a socket subscribes to a multicast group/port, it receives datagrams sent by other hosts to the group/port, as do all other members of the group and port.
- A socket relinquishes membership in a group by the leaveGroup(InetAddress addr) method.
- **Multiple MulticastSocket's** may subscribe to a multicast group and port concurrently, and they will all receive group datagrams.

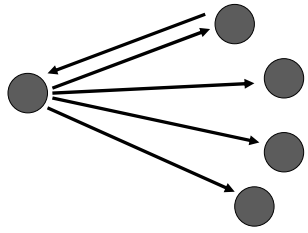
Failure Model

- Datagram multicasts have the same failure characteristics as UDP datagrams:
  messages can be lost,
  arrive out-of-order,
  can be duplicated...
- Thereby, with IP mcast we have the following semantics: Unreliable Multicast

Reliability Semantics of Multicast

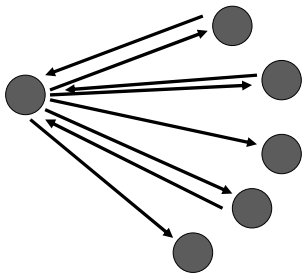
- 0-reliable
- 1-reliable
- m-out-n reliable
- All-reliable

How to achieve 1-Reliable?



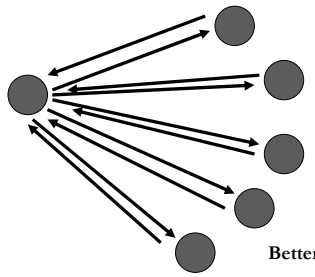
Example: read data from replicated servers

How to achieve m-out-n Reliable?



Example: voting system

How to achieve All-Reliable?



Better Algorithm?...

Example: updating replicated data...

Reliable Multicast: using one-to-one reliable messages

- `B-Multicast(g,m): // mcast message m to a group g`
- `for each process p € g`
  `reliable-send(p,m)`
- `On receive(m) at p: B-deliver(m) at p`

- This implementation uses threads to perform the send operations concurrently (scalability problems...)
- The large number of one-to-one and ack messages may lead to some buffer overflow problems...
- Inefficient use of the network...

Reliable Multicast over IP Multicast

- IP-multicast is successful in most cases...
- Use negative acknowledgements to indicate non-delivery.
- Use piggyback acknowledgements in messages.

Algorithm

- Each process (p) maintains a sequence number (S_g^p) for each group (g) that is belongs to.
- **S_g^p := 0;**
- Each process also records R_g^q, the sequence number of the latest message it has delivered from process (q) that was sent to group (g).

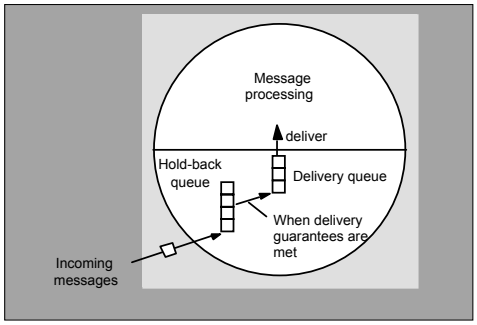
Reliable-IP-Multicast

- Process **p R-Multicast** message **m** to group **g:**
- increment S_g^p by 1
- **piggyback** in to the message
  S_g^p
  ack <q,R_g^p> for all q
- IP-multicast message and piggyback information

Delivery of Multicast Messages

- **R-Deliver** message from p:
- Only if received sequence number S = R_g^p + 1
- Then increment R_g^p by 1
- Retain any message that cannot yet be delivered in **hold-back-queue**

Hold-Back Queue for Arriving Messages



Reliable Delivery

- R-Deliver message from p:
- If S <= R_g^p,
  then message is already delivered, discard
- If S > R_g^p or R > R_g^q for any enclosed acknowledgement <q,R> , then receiver has missed one or more messages. It should request a retransmission of that message through a **negative acknowledgment**.

Properties of this R-Multicast Protocol

- **Integrity** (detection of duplicates)
- **Validity** (message lost can only be detected when a sucessor message is eventually transmitted; requires processes to multicast messages indefinitely)
- **Agreement** (required unbounded history of broadcast messages so that retransmit is always possible).
- These two last assumptions cannot be practical...