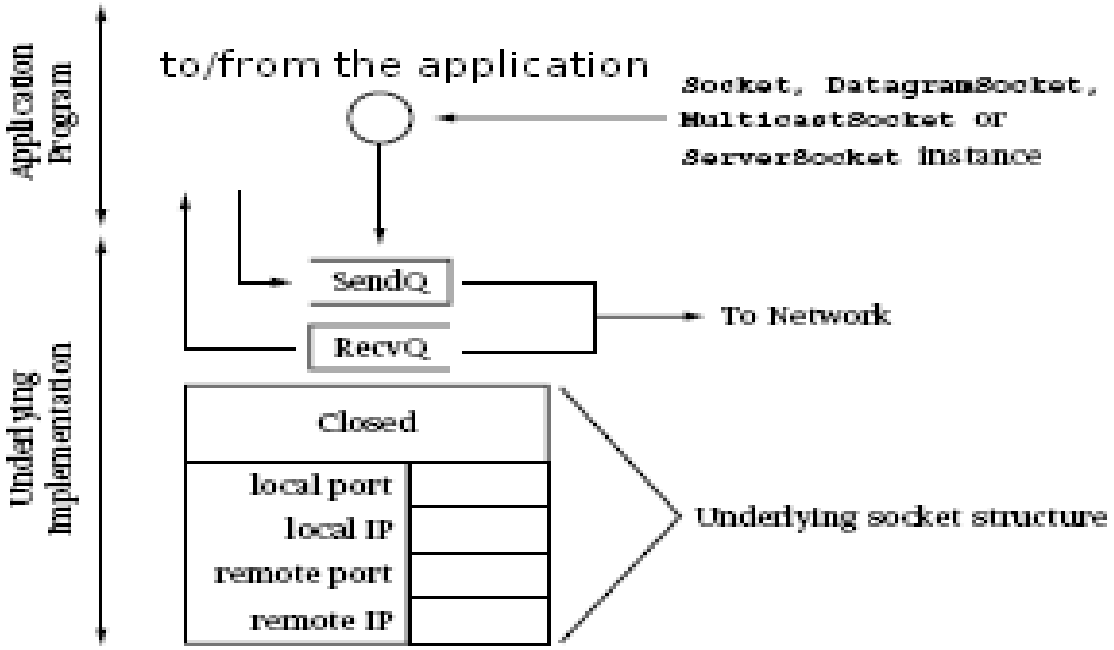
Lezione n.7
LPR-B-08
TCP Sockets & Multicast
19/11/2008

Andrea Corradini
Laura Ricci

Sommario

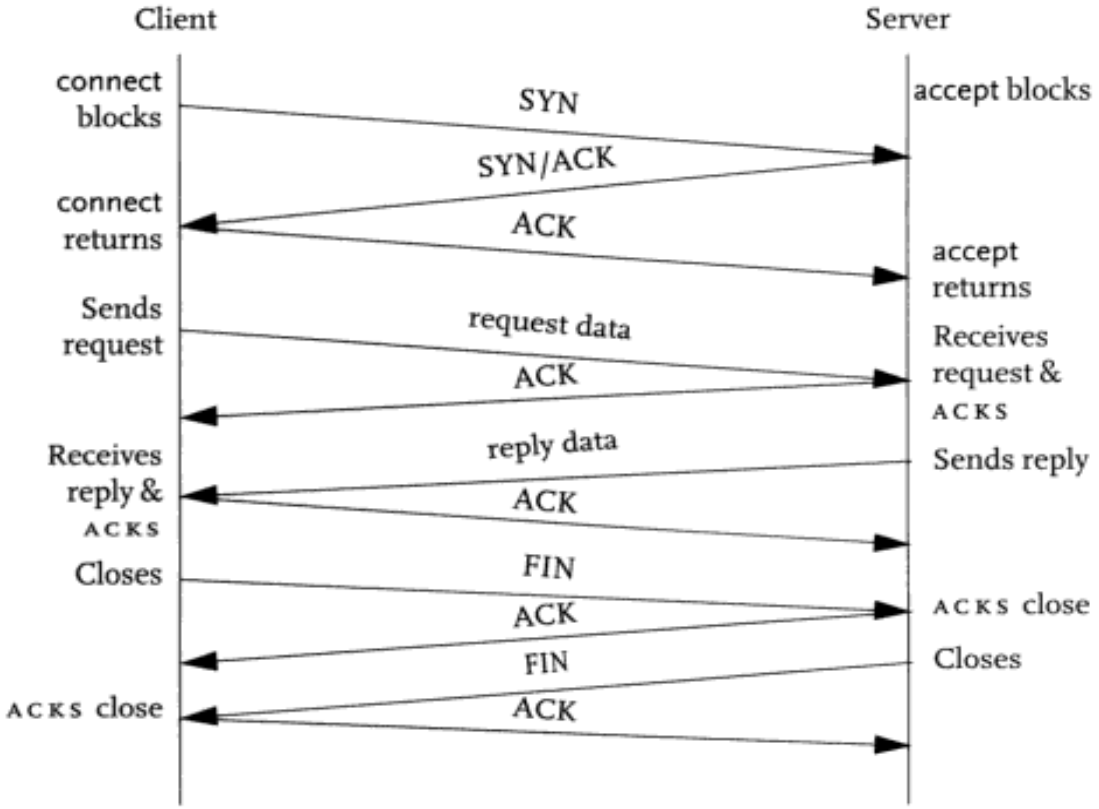
- Ancora sugli stream socket e su three-ways handshaking di TCP
- De-multiplexing di frammenti TCP
- Invio di oggetti tramite TCP con serializzazione (rischio di deadlock)
- Qualcosa sugli esercizi...
- Unreliable Multicast: concetti e API Java
- Panoramica su Linux Networking Tools (grazie a Daniele Sgandurra)

STRUTTURA GENERALE DI UN SOCKET

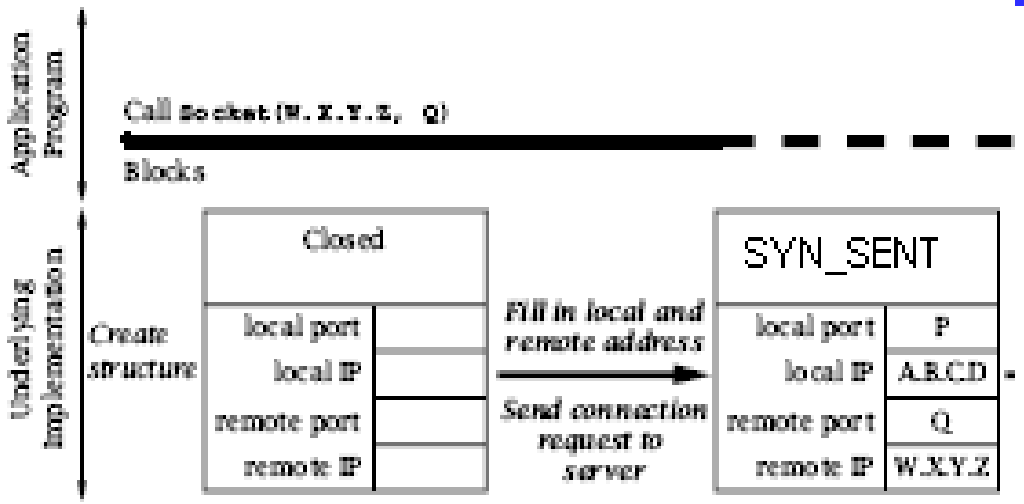


- remote port ed host significative solo per socket TCP
- SendQ, RecQ: buffer di invio/ricezione
- ogni socket è caratterizzato da informazioni sul suo stato (ad esempio closed). Lo stato del socket è visibile tramite il comando netstat

TCP: GESTIONE DELLE CONNESSIONI



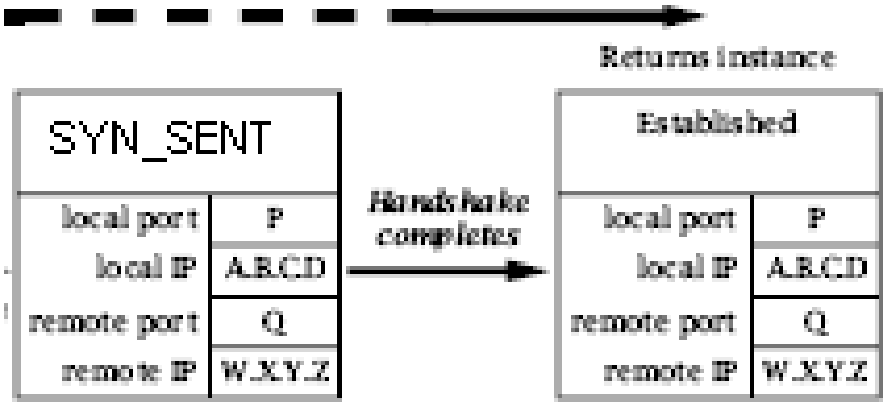
CONNESSIONE LATO CLIENT: STATO DEL SOCKET



Quando il client invoca il costruttore Socket( ).

- lo stato iniziale del socket viene impostato a Closed, la porta (P) e l'indirizzo locale (A.B.C.D) sono impostate dal supporto
- dopo aver inviato il messaggio iniziale di handshake, lo stato del socket passa a SYN_SENT (inviato segmento SYN)
- il client rimane bloccato fino a che il server riscontra il messaggio di handshake mediante un ack

CONNESSIONE LATO CLIENT: STATO DEL SOCKET

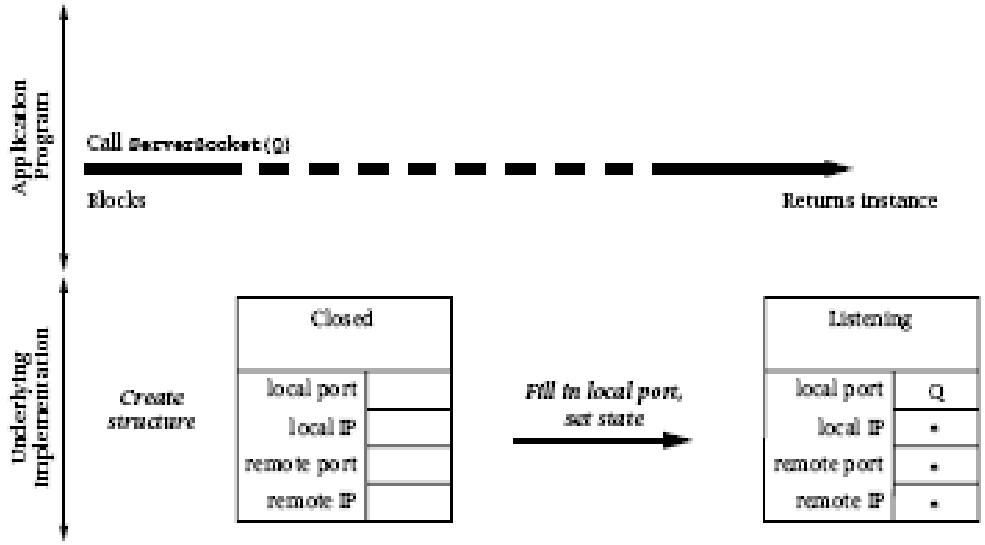


- il messaggio di handshake può venire trasmesso più volte
- il client può rimanere bloccato per un lungo periodo.

Il costruttore può sollevare una eccezione se,

- non esiste il servizio richiesto sulla porta selezionata
- il messaggio di handshake non viene riscontrato entro un certo intervallo di tempo(timeout)

CONNESSIONE LATO SERVER: STATO DEL SOCKET



Il Server crea un server socket sulla porta P

- se non viene specificato alcun indirizzo IP (wildcard = *), il server può ricevere connessioni da una qualsiasi delle sue interfacce
- lo stato del socket viene posto a Listening: questo indica che il server sta attendendo connessioni da una qualsiasi interfaccia, sulla porta P

CONNESSIONE LATO SERVER: STATO DEL SOCKET

- il server si sospende sul metodo accept( ) in attesa di una nuova connessione
- quando riceve una richiesta di connessione dal client, crea una nuova struttura che implementa il nuovo socket creato. In tale struttura
  - indirizzo e porta remoti vengono inizializzati con l'indirizzo IP e la porta ricevuti dal client che ha richiesto la connessione
  - L'indirizzo locale viene settato con l'indirizzo dell'interfaccia da cui è stata ricevuta la connessione.
  - La porta locale viene inizializzata con quella a cui associata al server socket
  - Lo stato del nuovo socket è SYN_RCVD

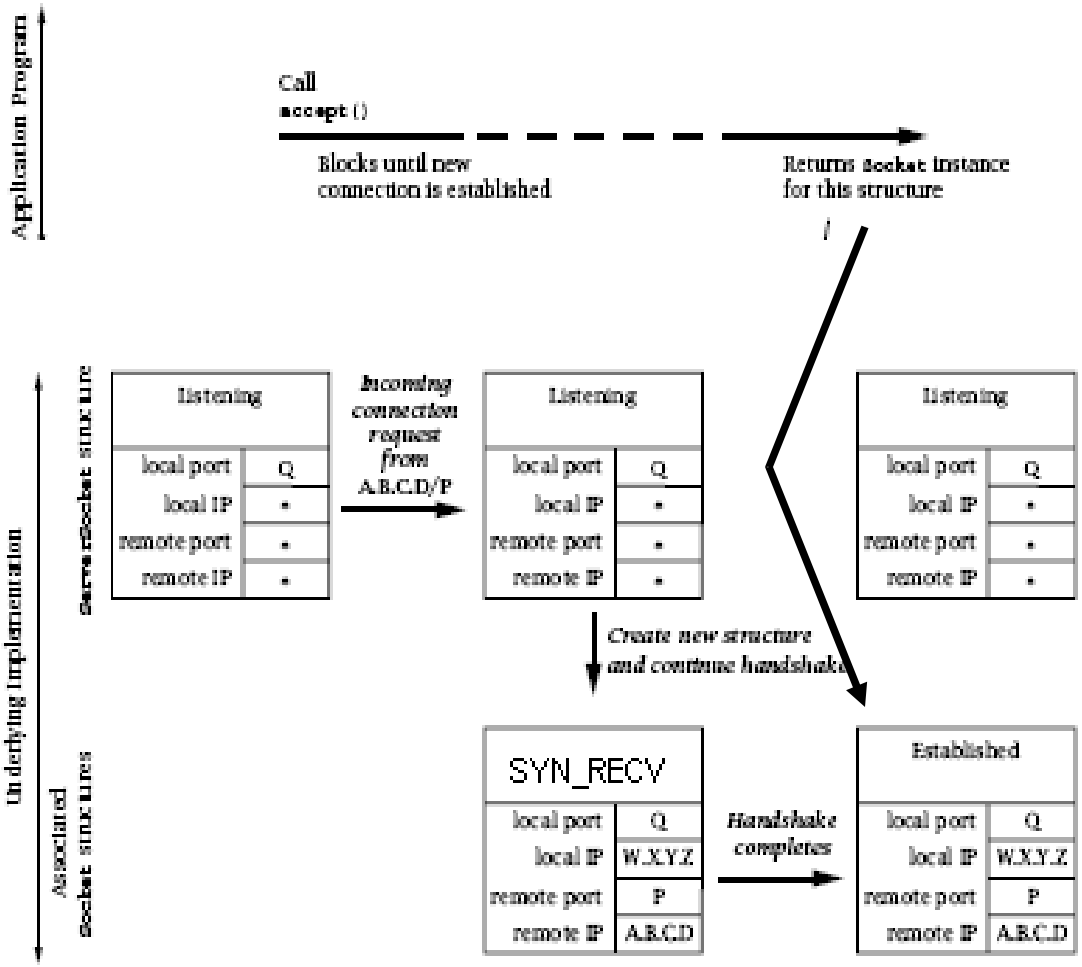
    è stato inviato il SYN/ACK al client e si sta attendendo l'ACK dal client, per terminare il 3-way handshake

CONNESSIONE LATO SERVER: STATO DEL SOCKET

Dopo aver creato il nuovo socket, il server

- riscontra il **SYN** inviato dal client mediante un **SYN/ACK**
- quando riceve, a sua volta, il riscontro dal client (**ACK**, terzo messaggio del 3-way handshake)
  - imposta lo stato del socket ad ESTABLISHED
  - inserisce il socket creato in una lista di socket associata al ServerSocket da cui è stata ricevuta la richiesta di connession
  - Solo a questo punto, l'esecuzione del metodo accept( ) termina e restituisce un puntatore alla struttura creata
- Anche il client imposta lo stato del proprio socket ad ESTABLISHED dopo aver terminato il 3-way handshake

CREAZIONE DI CONNESSIONI LATO SERVER



DEMULTIPLEXING DEI SEGMENTI TCP

- Tutti i sockets associati allo stesso ServerSocket 'ereditano' da esso
  - la porta di ascolto
  - l' indirizzo IP da cui è stata ricevuta la richiesta di connessione
- Questo implica che sullo stesso host possano esistere più sockets associati allo stesso indirizzo IP ed alla stessa porta locale (il **Server Socket** e tutti i **Sockets** associati,.....)
- **Meccanismo di demultiplexing**: utilizzato per decidere a quale socket è destinato un segmento TCP ricevuto su quella interfaccia e su quella porta
- La conoscenza dell'indirizzo e porta destinazione non risulta più sufficiente per individuare il socket a cui è destinato il segmento

DEMULTIPLEXING DEI SEGMENTI TCP

Definizione del meccanismo di demultiplexing:

- La porta locale riferita nel socket deve coincidere con quella contenuta nel segmento TCP
- Ogni campo del socket contenente una wildcard (*), può essere messo in corrispondenza con qualsiasi valore corrispondente contenuto nel segmento
- Se esiste più di un socket che corrisponde al segmento in input, viene scelto il socket

  che contiene il minor numero di wildcards.

- in questo modo un segmento spedito dal client viene ricevuto sul socket S associato alla connessione con quel client, piuttosto che sul serversocket perchè S risulta 'più specifico' per quel segmento

INVIO DI OGGETTI SU CONNESSIONI TCP

- Per inviare oggetti su una connessione TCP basta usare la serializzazione: gli oggetti inviati devono implementare l'interfaccia Serializable
- Si possono usare i filtri ObjectInputStream / ObjectOutputStream per incapsulare gli stream ottenuti invocando getInputStream() / getOutputStream() sul socket
- Quando creo un ObjectOutputstream viene scritto lo stream header sullo stream. In seguito scrivo gli oggetti che voglio inviare sullo stream
- L'header viene letto quando viene creato il corrispondente ObjectInputStream
- L'invio/ ricezioni degli oggetti sullo/dallo stream avviene mediante scritture/letture sullo stream (writeObject(), readObject())

INVIO DI OGGETTI SU UNA CONNESSIONE TCP

```
import java.io.*;

    public class Studente implements Serializable {
    private int matricola;
    private String nome, cognome, corsoDiLaurea;
    public Studente (int matricola, String nome, String cognome,
                        String corsoDiLaurea) {
        this.matricola = matricola; this.nome = nome;
        this.cognome = cognome; this.corsoDiLaurea = corsoDiLaurea;}
    public int getMatricola () { return matricola; }
    public String getNome () { return nome; }
    public String getCognome () { return cognome; }
    public String getCorsoDiLaurea () { return corsoDiLaurea; }   }
```

INVIO DI OGGETTI SU UNA CONNESSIONE TCP- LATO SERVER

```
import java.io.*; import java.net.*;
public class TCPObjectServer {
public static void main (String args[]) {
try { ServerSocket server = new ServerSocket (3575);
      Socket clientsocket = server.accept();
      ObjectOutputStream output =
                  new ObjectOutputStream (clientsocket.getOutputStream ());
      output.writeObject("<Welcome>");
      Studente studente = new Studente (14520,"Mario","Rosso","Informatica");
      output.writeObject(studente);  output.writeObject("<Goodbye>");
      clientsocket.close();
      server.close();
} catch (Exception e) { System.err.println (e); } } }
```

INVIO DI OGGETTI SU UNA CONNESSIONE TCP-LATO CLIENT

```
import java.io.*; import java.net.*;
public class TCPObjectClient { public static void main (String args[ ]) {
try { Socket socket = new Socket ("localhost", 3575);
ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
String beginMessage = (String) in.readObject();
Studente stud = (Studente) in.readObject();
System.out.println(beginMessage);
System.out.print(stud.getMatricola() + " - " + stud.getNome() + " ");
System.out.println(stud.getCognome() + " - " + stud.getCorsoDiLaurea());
String endMessage = (String) in.readObject();
System.out.println (endMessage); socket.close();} catch (Exception e)
   { System.out.println (e); } } }
```

INVIO DI OGGETTI SU UNA CONNESSIONE TCP- LATO CLIENT

Stampa prodotta lato Client

<Welcome>

14520 - Mario Rossi - Informatica

<Goodbye>

OBJECT INPUT/OUTPUT STREAM: RISCHIO DI DEADLOCK

- La creazione dell'ObjectInputStream cerca di leggere lo header. Se questo non è stato ancora creato, si blocca.
- Quindi **si verifica una situazione di deadlock** se i due partner della connessione eseguono le istruzioni nel seguente ordine (s è il socket locale):

```
ObjectInputStream in = new ObjectInputStream(s.getInputStream( ));
ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream( ));
```

Infatti,

- entrambi tentano di leggere l'header dello stream dal socket
- l'header viene generato quando viene viene creato l'ObjectOutputStream
- nessuno dei due è in grado di generare l'ObjectOutputStream, perchè bloccato
- E' sufficiente invertire l'ordine di creazione degli stream in almeno uno dei partner

ESERCIZIO:ASTA ELETTRONICA

Sviluppare un programma client server per il supporto di un'asta elettronica. Ogni client possiede un budget massimo B da investire. Il client può richiedere al server il valore V della migliore offerta pervenuta fino ad un certo istante e decidere se abbandonare l'asta, oppure rilanciare. Se il valore ricevuto dal server supera B,l'utente abbandona l'asta, dopo aver avvertito il server. Altrimenti, il client rilancia, inviando al server un valore maggiore di V. Il server invia ai client che lo richiedono il valore della migliore offerta ricevuta fino ad un certo momento e riceve dai client le richieste di rilancio. Per ogni richiesta di rilancio, il server notifica al client se tale offerta può essere accettata (nessuno ha offerto di più nel frattempo), oppure è rifiutata.

ESERCIZIO:ASTA ELETTRONICA

Il server deve attivare un thread diverso per ogni client che intende partecipare all'asta.

La comunicazione tra clients e server deve avvenire mediante socket TCP. Sviluppare due diverse versioni del programma che utilizzino, rispettivamente una codifica testuale dei messaggi spediti tra client e server oppure la serializzazione offerta da JAVA in modo da scambiare oggetti tramite la connessione TCP

GRUPPI DI PROCESSI: COMUNICAZIONE

- Comunicazioni di tipo unicast = coinvolgono una sola coppia di processi
- Ma diverse applicazioni di rete richiedono un tipo di comunicazione che coinvolga un gruppo di hosts.

Applicazioni classiche

- usenet news: pubblicazione di nuove notizie ed invio di esse ad un gruppo di hosts interessati
- videoconferenze: un segnale audio video generato su un nodo della rete deve essere ricevuto dagli hosts associati ai partecipanti alla videoconferenza

Altre applicazioni

- massive multiplayer games: alto numero di giocatori che intergiscono in un mondo virtuale
- DNS (Domain Name System): aggiornamenti delle tabelle di naming inviati a gruppi di DNS, chats, instant messaging, applicazioni p2p

GRUPPI DI PROCESSI: COMUNICAZIONE

Comunicazione tra gruppi di processi realizzata mediante **multicasting** (one to many communication).

Comunicazione di tipo multicast

- un insieme di processi formano un gruppo di multicast
- un messaggio spedito da un processo a quel gruppo viene recapitato a tutti gli altri partecipanti appartenenti al gruppo
- Un processo può lasciare un gruppo di multicast quando non è più interessato a ricevere i messaggi del gruppo

COMUNICAZIONE TRA GRUPPI DI PROCESSI

Multicast API: deve contenere primitive per

- unirsi ad un gruppo di multicast (join). E' richiesto uno schema di indirizzamento per identificare univocamente un gruppo.
- lasciare un gruppo di multicast (leave).
- spedire messaggi ad un gruppo. Il messaggio viene recapitato a tutti i processi che fanno parte del gruppo in quel momento
- ricevere messaggi indirizzati ad un gruppo

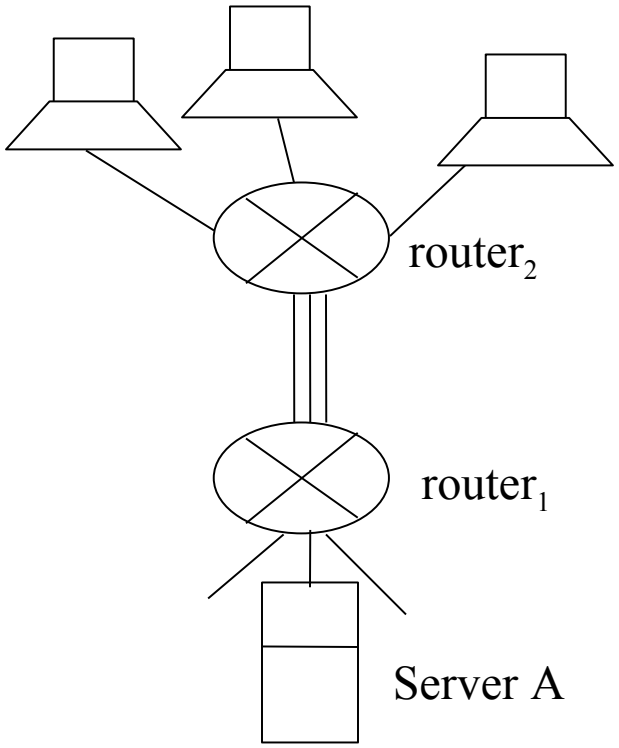
COMUNICAZIONE TRA GRUPPI DI PROCESSI: IMPLEMENTAZIONE

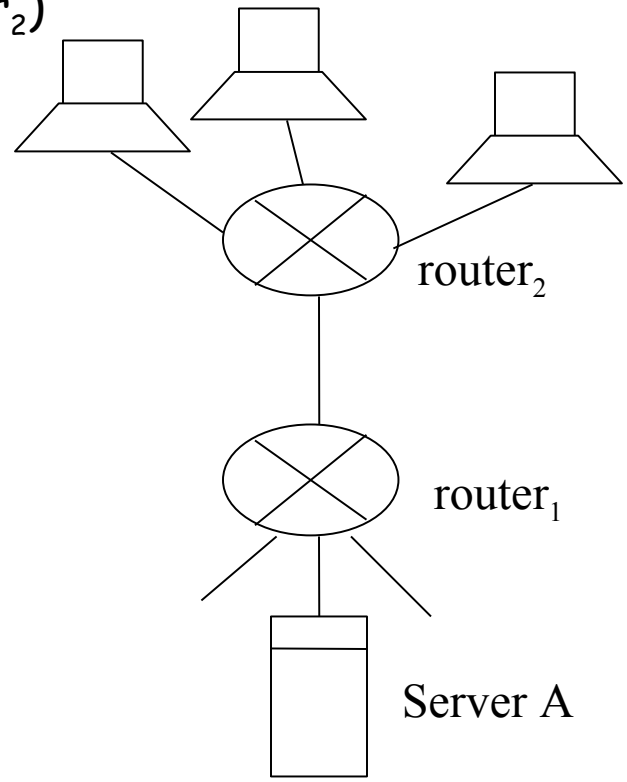
L'implementazione del multicast richiede:

- uno schema di indirizzamento dei gruppi
- un supporto che registri la corrispondenza tra un gruppo ed i partecipanti
- un'implementazione che ottimizzi l'uso della rete nel caso di invio di pacchetti ad un gruppo di multicast

MULTICAST: IMPLEMENTAZIONE

Server A invia un messaggio su un gruppo di multicast composto da 3 clients connessi allo stesso router ($router_2$)





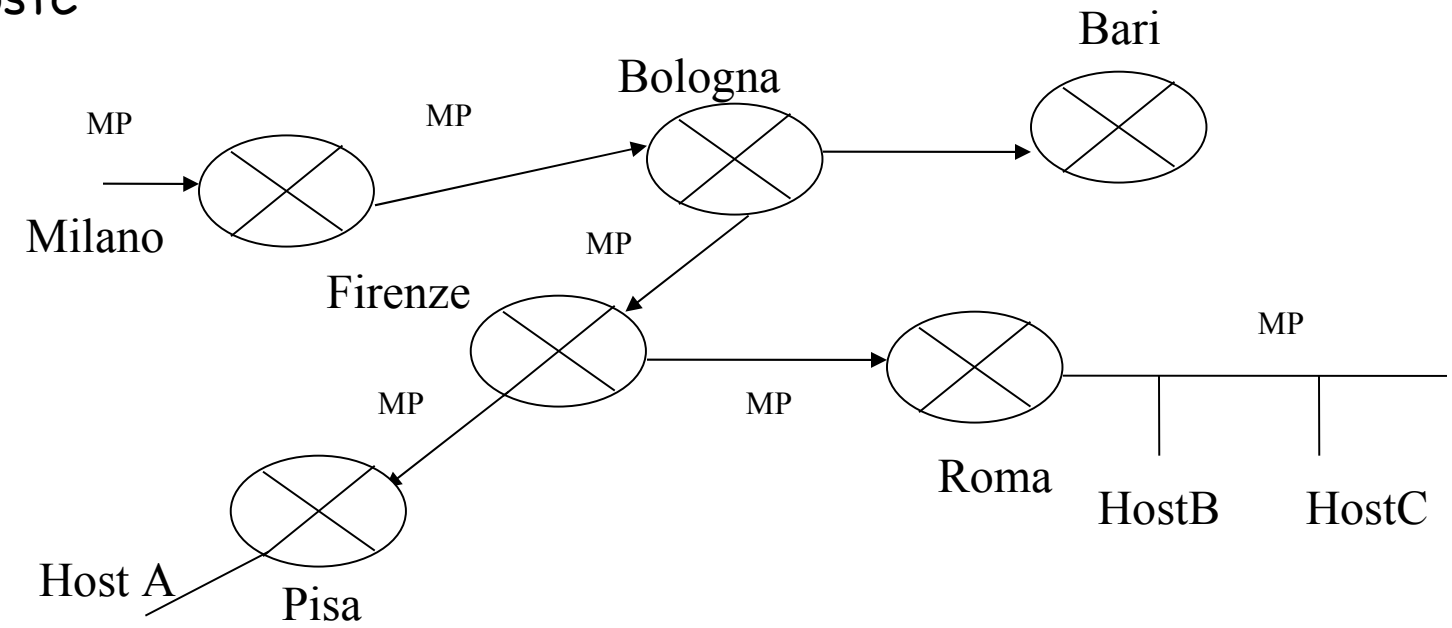
Soluzione 1: $router_1$ invia 3 messaggi con collegamenti di tipo unicast

Soluzione 2: $router_1$ invia un unico messaggio. $router_2$ replica il messaggio per i tre clients

MULTICAST: IMPLEMENTAZIONE

Ottimizzazione della banda di trasmissione: il router che riceve un pacchetto di multicast MP invia un unico pacchetto sulla rete. Il pacchetto viene replicato solo quando è necessario.

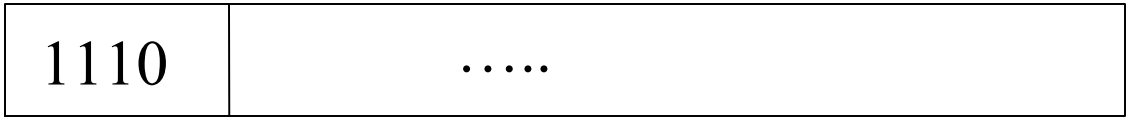
Esempio: pacchetto di multicast spedito da Milano agli hosts Hosta, HostB, HostC



INDIVIDUAZIONE GRUPPI DI MULTICAST

- Indirizzo di multicast: indirizzo IP di classe D, che individua un gruppo di multicast

- Indirizzo di classe D- intervallo 224.0.0.0 – 239-255-255-255



- l'indirizzo di multicast è condiviso da tutti i partecipanti al gruppo

- è possibile associare un nome simbolico ad un gruppo di multicast

- Esempio: **224.0.1.1** **ntp.mcast.net**
  (network time protocol distributed service)

INDIVIDUAZIONE GRUPPI DI MULTICAST

- Il livello IP (nei routers) mantiene la corrispondenza tra l'indirizzo di multicast e gli indirizzi IP dei singoli hosts che partecipano al gruppo di multicast
- Gli indirizzi possono essere:

  Permanenti : l'indirizzo di multicast viene assegnato dalla **IANA** (**Internet Assigned Numbers Authority**).

  L'indirizzo rimane assegnato a quel gruppo, anche se, in un certo istante non ci sono partecipanti

  Temporanei : Esistono solo fino al momento in cui esiste almeno un partecipante. Richiedono la definizione di un opportuno protocollo per evitare conflitti nell'attribuzione degli indirizzi ai gruppi

INDIVIDUAZIONE GRUPPI DI MULTICAST

- Gli indirizzi di multicast appartenenti all'intervallo

  224.0.0.0 - 224.0.0.255

  sono riservati per i protocolli di routing e per altre funzionalità a livello di rete

  ALL-SYSTEMS.MCAST.NET **224.0.0.1**

  tutti gli host della rete locale

  ALL-ROUTERS.MCAST.NET **224.0.0.2**

  tutti i routers della rete locale

- I routers non inoltrano mai i pacchetti che contengono questi indirizzi
- la maggior parte degli indirizzi assegnati in modo permanente hanno come prefisso **224.0**, **224.1**, **224.2**, oppure **239**

MULTICAST ROUTERS

- Per poter spedire e ricevere pacchetti di multicast oltre i confini della rete locale, occorre disporre di un router che supporta il multicast (mrouter)

- Problema: disponbilità limitata di mrouters

- Per testare se la vostra rete è collegata ad un mrouter, dare il comando

  % ping all-routers.mcast.net

  - Se si ottiene una risposta, c'è un mrouter sulla sottorete locale.
  - Routers che non supportano multicast, possono utilizzare la tecnica del ***tunnelling*** = trasmissione di pacchetti multicast mediante unicast UDP

CONNECTIONLESS MULTICAST

La comunicazione Multicast utilizza il paradigma connectionless

Motivazioni:

- gestione di un alto numero di connessioni
- richieste n(n-1) connessioni per un gruppo di n processi
- comunicazione connectionless adatta per il tipo di applicazioni verso cui è orientato il multicast (trasmissione di dati video/audio).

- Esempio: invio dei frames di una animazione. E' più accettabile la perdita occasionale di un frame piuttosto che un ritardo costante tra la spedizione di due frames successivi

UNRELIABLE VS. RELIABLE MULTICAST

Unreliable Multicast (multicast non affidabile):

- non garantisce la consegna del messaggio a tutti i processi che partecipano al gruppo di multicast.
- unico servizio offerto dalla multicast JAVA API standard (esistono package JAVA non standard che offrono qualche livello di affidabilità)

Reliable Multicast (multicast affidabile):

- garantisce che il messaggio venga recapitato una sola volta a tutti i processi del gruppo
- può garantire altre proprietà relative all'ordinamento con cui i messaggi spediti al gruppo di multicast vengono recapitati ai singoli partecipanti

MULTICAST API DI JAVA: MulticastSocket

MulticastSocket = socket su cui spedire/ricevere i messaggi verso/da un gruppo di multicast

- la classe **MulticastSocket** estende la **DatagramSocket** con alcune funzionalità utili per il multicast

- il **ricevente** deve creare un **MulticastSocket** su una determinata **porta**, **iscrivere il socket** al **gruppo**, e fare una **receive**.

- il **mittente** deve inviare il messaggio (un **DatagramPacket**) specificando il **gruppo** e una **porta**.

- il **messaggio** è ricevuto da tutti i **MulticastSocket** iscritti al **gruppo** e che stanno ricevendo sulla **porta indicata**.

MULTICAST: L'API JAVA

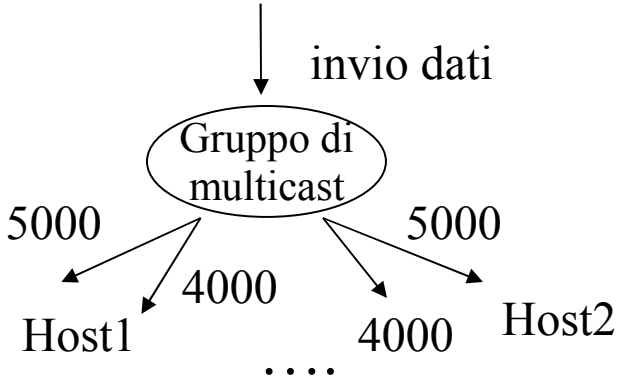
Uso delle porte per multicast sockets:

Unicast

- IP Address individua un host,
- porta individua un servizio

Multicast

- IP Address individua un gruppo di hosts,
- porta consente di partizionare dati di tipo diverso inviati allo stesso gruppo



Esempio: porta 5000 **traffico audio,** porta 4000 **traffico video**

MULTICAST API DI JAVA: il receiver

```
import java.net.*;
public class MulticastTestReceiver{
    public static void main (String [ ] args) throws Exception{
        InetAddress group = InetAddress.getByName(args[0]); // gruppo
        if (!group.isMulticastAddress()){       // controllo se è multicast
            throw new IllegalArgumentException();
        }
        int port = Integer.parseInt(args[1]);  // porta locale
        MulticastSocket ms = new MulticastSocket(port);
        ms.joinGroup (group);  // mi iscrivo al gruppo
        DatagramPacket dp = new DatagramPacket(new byte[8192], 8192);
        ms.receive(dp);  // ricevo e stampo
        System.out.println(new String(dp.getData(),0,dp.getLength())); }}
```

MULTICAST API DI JAVA: il sender

```
import java.net.*;
public class MulticastTestSender{
    public static void main (String [ ] args) throws Exception{
        InetAddress group = InetAddress.getByName(args[0]); // gruppo
        if (!group.isMulticastAddress()){       // controllo se è multicast
            throw new IllegalArgumentException(); }
        int port = Integer.parseInt(args[1]);  // porta destinataria
        System.out.println("String to send? ");
        byte [] data = Input.readLine().getBytes();
        DatagramPacket dp =                          // creo il pacchetto
                new DatagramPacket(data, data.length, group, port);
        MulticastSocket ms = new MulticastSocket();
        ms.setTimeToLive(5);     ms.send(dp); }}   // spedisco
```

MULTICAST: più socket sulla stessa porta

Una porta non individua un servizio (processo) su un certo host:

- Se attivo due istanze di **MulticastTestReceiver** sullo stesso host e sulla stessa porta non viene sollevata una BindException (che viene invece sollevata se **MulticastSocket** è sostituito da un **DatagramSocket**)

MULTICAST SNIFFER

- Il programma riceve in input il nome simbolico di un gruppo di multicast si unisce al gruppo e 'sniffa' i messaggi spediti su quel gruppo, stampandone il contenuto

```
import java.net.*; import java.io.*;
public class MulticastSniffer {
    public static void main (String[] args){
        InetAddress group = null;      // indirizzo del gruppo
        int port = 0;                  // porta locale
        try{group = InetAddress.getByName(args[0]);
            port = Integer.parseInt(args[1]);
        } catch(Exception e){System.out.println("Uso: " +
            "java multicastsniffer multicast_address port");
        System.exit(1); }
```

MULTICAST SNIFFER

```
MulticastSocket ms = null;
try{ ms = new MulticastSocket(port);
      ms.joinGroup(group);                    // mi unisco al gruppo
      byte [ ] buffer = new byte[8192];
      while (true){
            DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
            ms.receive(dp);                   // aspetto un pacchetto
            String s = new String(dp.getData());      // estraggo il messaggio
            System.out.println(s);}                   // .lo stampo
      } catch (IOException ex){System.out.println (ex);
      }
```

MULTICAST SNIFFER

```
finally{                        // in ogni caso...
    if (ms != null) {  // se avevo aperto il multicast socket
        try{
            ms.leaveGroup(group);     // lascio il gruppo
            ms.close();               // chiudo il socket
        } catch (IOException ex){}
    }}}}
```

MULTICAST: TIME TO LIVE

- IP Multicast Scoping: meccanismo utilizzato per limitare la diffusione sulla rete di un pacchetto inviato in multicast
- ad ogni pacchetto IP viene associato un valore rappresentato su un byte, riferito come TTL (Time-To-Live) del pacchetto
- TTL assume valori nell'intervallo 0-255
- TTL indica il numero massimo di routers che possono essere attraversati dal pacchetto
- il pacchetto viene scartato dopo aver attraversato TTL routers
- meccanismo introdotto originariamente per evitare loops nel routing dei pacchetti

MULTICAST: TIME TO LIVE

Internet Scoping, implementazione

- il mittente specifica un valore per del TTL per i pacchetti spediti
- il TTL viene memorizzato in un campo dell'header del pacchetto IP
- TTL viene decrementato da ogni router attraversato
- Se TTL = 0 $\Rightarrow$ il pacchetto viene scartato

MULTICAST: TIME TO LIVE

Valori consigliati per il ttl

Destinazione	Valori di ttl
processi sullo stesso host	0
processi sulla stessa sottorete locale	1
processi su reti locali gestite dallo stesso router	16
processi allocati su un generico host di Internet	255

TIME TO LIVE: API JAVA

- Assegna il valore di default = 1 al TTL (i pacchetti di multicast non possono lasciare la rete locale)

- Per modificare il valore di default: posso associare il **ttl** al multicast socket

```
MulticastSocket s = new MulticastSocket( );
s.setTimeToLive(16);
```

MULTICAST: ESERCIZIO

Definire un Server TimeServer, che invia su un gruppo di multicast dategroup, ad intervalli regolari,la data e l'ora. L'attesa tra un invio ed il successivo può essere simulata mediante il metodo sleep( ). L'indirizzo IP di dategroup viene introdotta linea di comando.

Definire quindi un client TimeClient che si unisce a dategroup e riceve, per dieci volte consecutive, data ed ora, le visualizza, quindi termina.