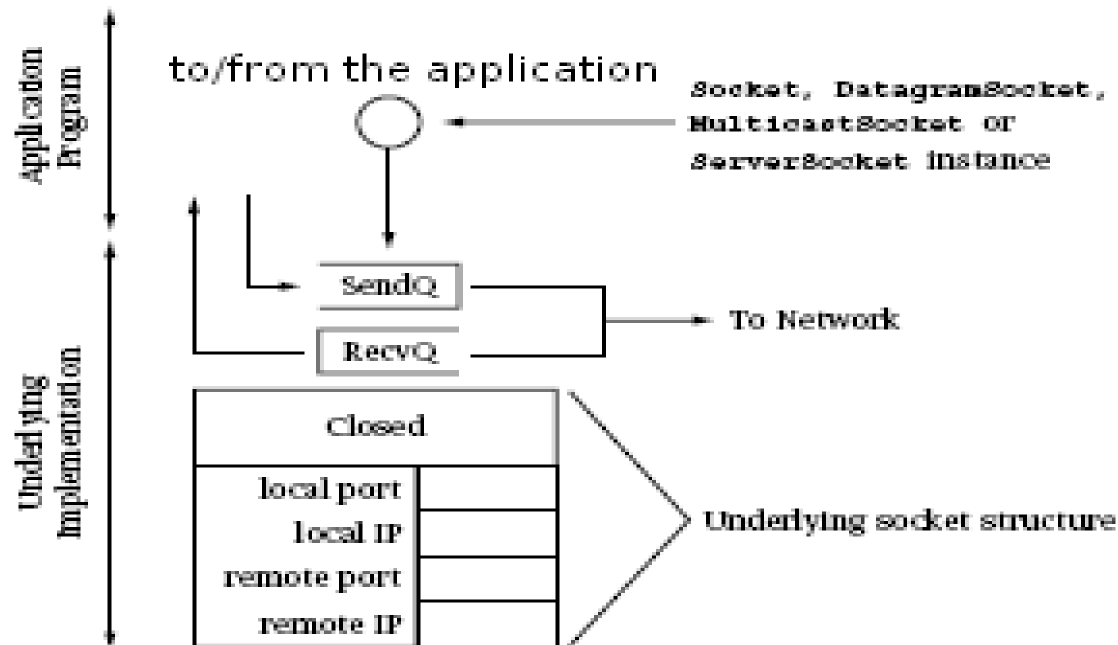
Università degli Studi di Pisa
Dipartimento di Informatica

Lezione n.8
LPR-A - INFORMATICA
TCP SOCKETS
UDP MULTICAST

17/11/1008

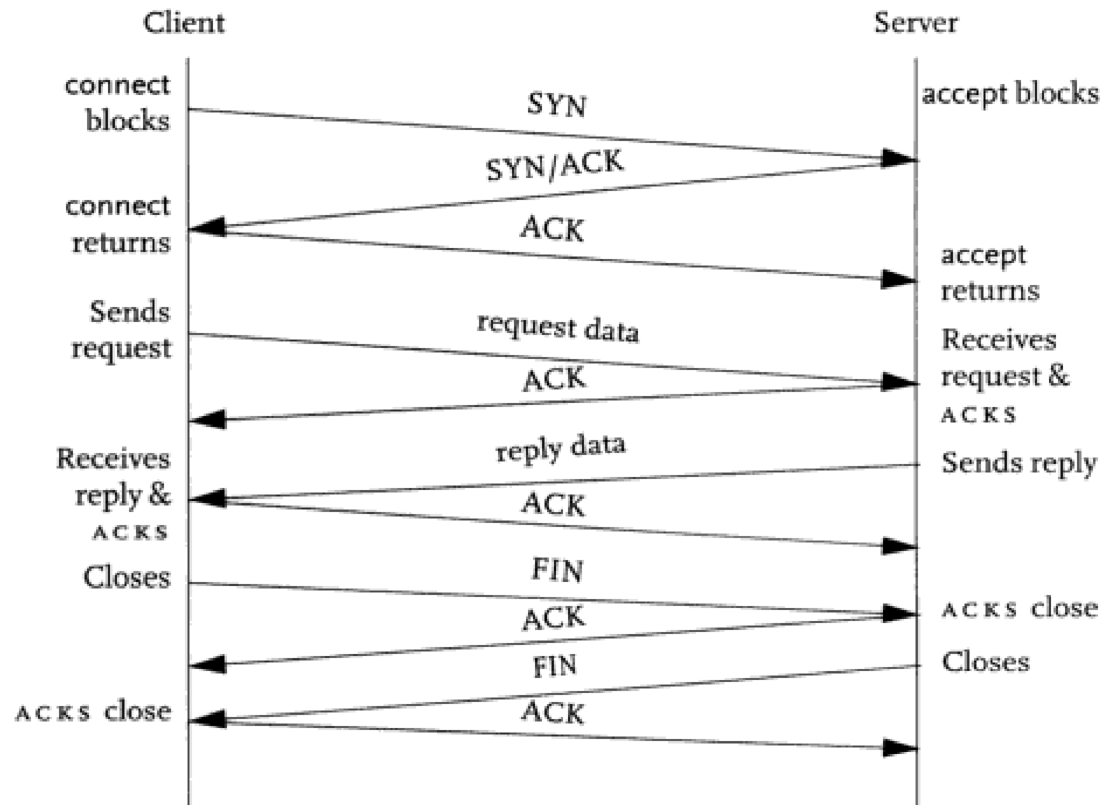
Laura Ricci

STRUTTURA GENERALE DI UN SOCKET

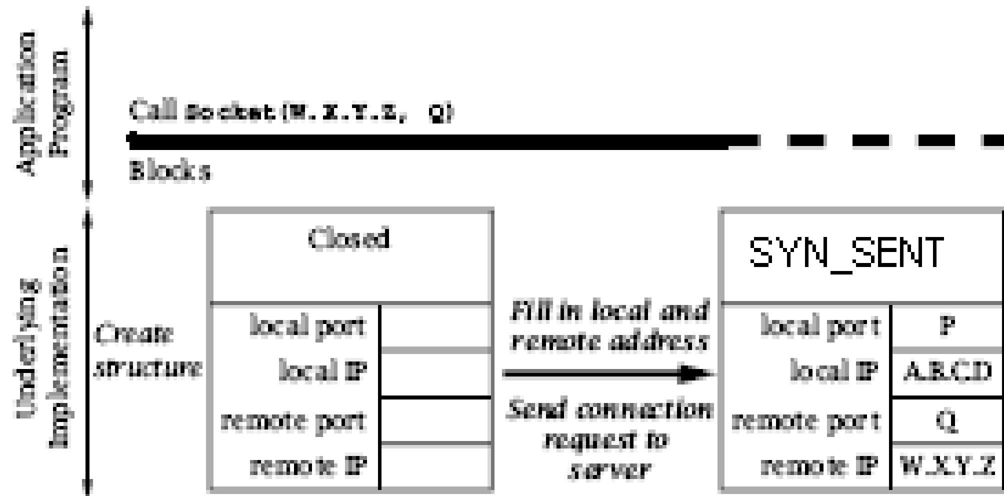


- remote port ed host significative solo per socket TCP
- SendQ, RecQ: buffer di invio/ricezione
- ogni socket è caratterizzato da informazioni sul suo stato (ad esempio closed). Lo stato del socket è visibile tramite il comando netstat

TCP: GESTIONE DELLE CONNESSIONI



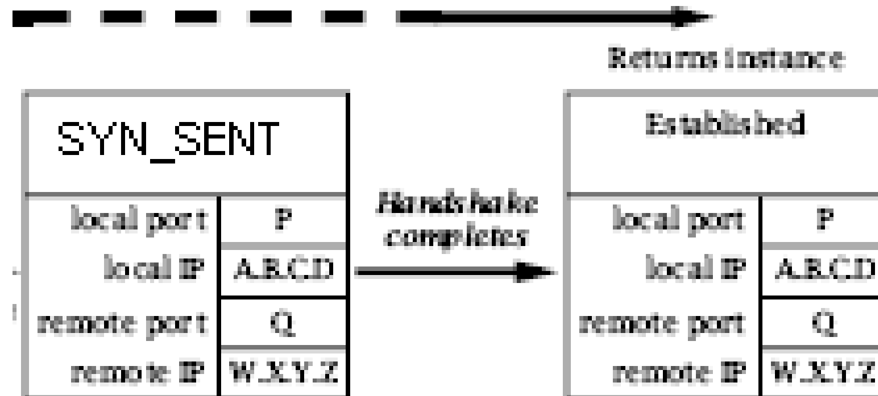
CONNESSIONE LATO CLIENT: STATO DEL SOCKET



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



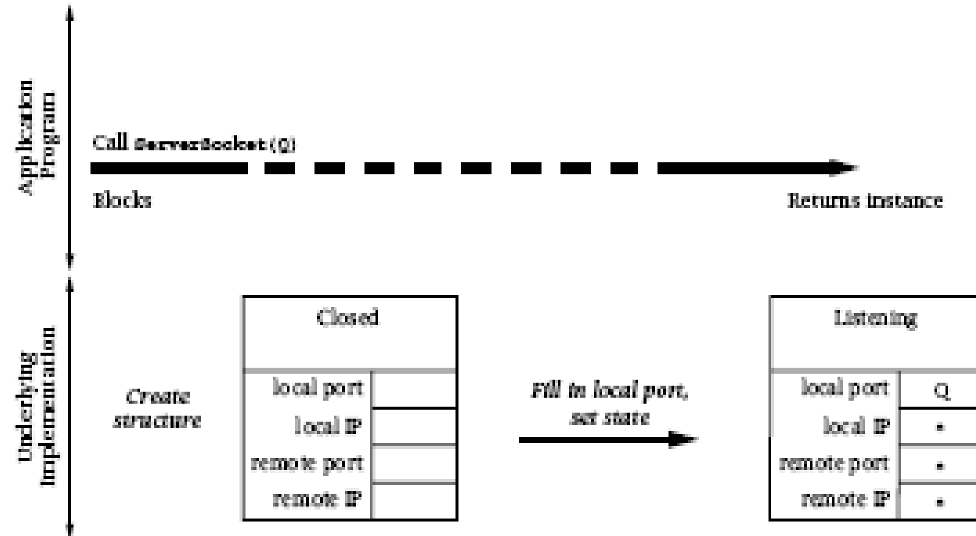
- il messaggio di handshake può venire trasmesso più volte

  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 serversocket
  - Lo stato del nuovo socket è SYN_RCVD

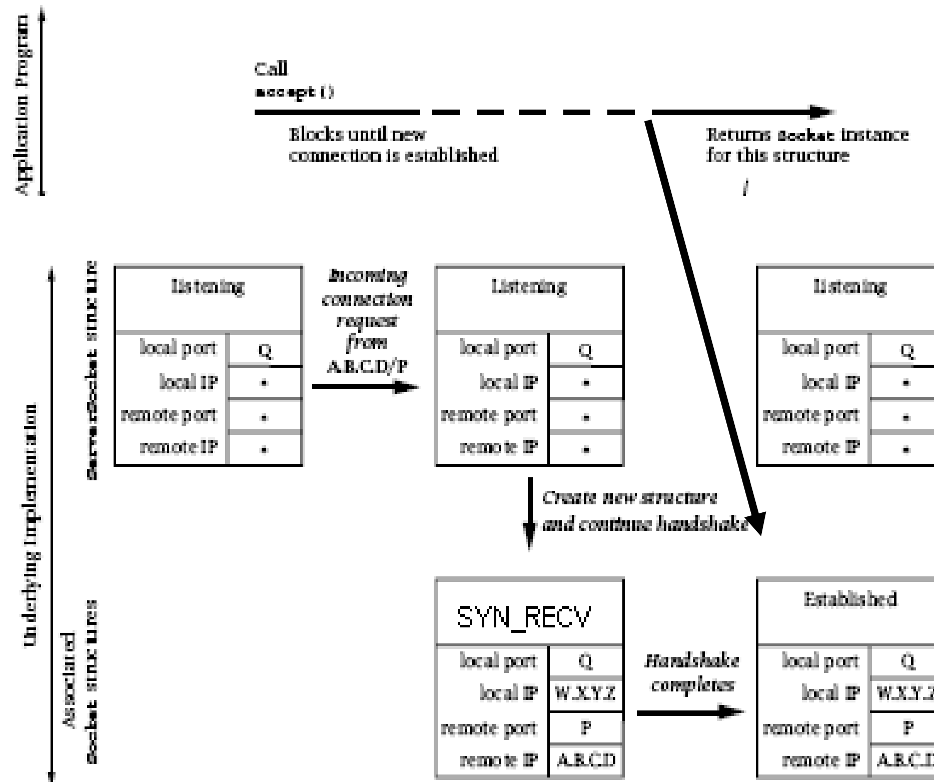
    è stato inviato il SYN/ACK al client e che si sta attendendo l'ACK dal client, per terminare il 3-way handshake

CONNESSIONE LATO SERVER: STATO DEL SOCKET

Dopo aver creato il nuovo socket, il server

- riscontra il SYN inviato dal client mediante un ack
- quando riceve, a sua volta, il riscontro dal client (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 punti, 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 Serversocket 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 quelsegmento

CHIUSURA DI SOCKETS TCP

- Metodi per la chiusura di un socket: close, shutdownOutput, shutdownInput
- In generale, il mittente M , dopo aver inviato tutti i dati al destinatario D, chiude il socket mediante
  - close, se M non attende risposte dal destinatario D
  - shutdownOutput, in caso contrario
- Quando M chiude il socket, il protocollo di chiusura di TCP prevede
  - l'invio di del segmento di FIN, che deve essere riscontrato dal destinatario D
  - l'attesa che D chiuda, a sua volta, il socket e ne segnali la chiusura ad M, mediante il corrispondente segmento di FIN
  - L'invio da parte di M a D del riscontro del segmento di FIN ricevuto

CHIUSURA DI SOCKET TCP

- Ipotesi Semplificative:
  - uno dei due partner della connessione completa l'handshake per la chiusura del socket, prima che l'altro parte inizi la stessa procedura
  - Linger= false, il metodo close/shutdownOutput ristorna immediatamente il controllo al chiamante e il protocollo di chiusura viene eseguito in background
- Protocollo di chiusura del socket: M invoca una close/shutdownOutput( )
  - i dati presenti nel send buffer di M vengono inviati al destinatario
  - il supporto in esecuzione su M invia quindi il segmento di FIN e lo stato del socket passa a FIN_WAIT1
  - quando il supporto del destinatario riceve il messaggio di FIN, trasforma la segnalazione di chiusura ricevuta in un messaggio di fine sequenza (valore = -1) da inviare all'applicazione (continua pag.succ.....)

CHIUSURA DI SOCKET TCP

Protocollo di chiusura del socket: M invoca una close()/shutdownOutput( )

- Quando M riceve da D il riscontro del FIN inviato (ricezione dell'ack), il socket passa nello stato di half closed (FIN_WAIT2).
  - Se D fallisce prima di aver completato la procedura di chiusura, il socket può rimanere indefinitamente in questo stato
- Il socket associato a D passa a questo punto in uno stato di CLOSE_WAIT, in attesa che l'applicazione in esecuzione su D chiuda a sua volta il socket
- Quando l'applicazione chiude il socket., il supporto invierà a M il segmento di FIN
- A questo punto anche se la connessione risulta completamente chiusa, ma il socket, passa nello stato di TIME_WAIT

NETSTAT: ANALIDI DELLO STATO DEI SOCKET

C:\WINDOWS\system32\cmd.exe

```
Connessioni attive

  Proto  Indirizzo locale          Indirizzo esterno           Stato
  TCP    ricci:2994                localhost:2995              ESTABLISHED
  TCP    ricci:2995                localhost:2994              ESTABLISHED
  TCP    ricci:2996                localhost:2997              ESTABLISHED
  TCP    ricci:2997                localhost:2996              ESTABLISHED
  TCP    ricci:3970                localhost:3971              ESTABLISHED
  TCP    ricci:3971                localhost:3970              ESTABLISHED
  TCP    ricci:3972                localhost:3973              ESTABLISHED
  TCP    ricci:3973                localhost:3972              ESTABLISHED
  TCP    ricci:2599                bw-in-f147.google.com:http  TIME_WAIT
  TCP    ricci:2600                bw-in-f147.google.com:http  CLOSE_WAIT
  TCP    ricci:2602                bw-in-f147.google.com:http  TIME_WAIT
  TCP    ricci:2603                bw-in-f147.google.com:http  CLOSE_WAIT
  TCP    ricci:2604                fx-in-f127.google.com:http  CLOSE_WAIT
  TCP    ricci:2605                bw-in-f147.google.com:http  CLOSE_WAIT
  TCP    ricci:2606                85.10.195.234:http          ESTABLISHED
  TCP    ricci:2607                beta.speak2us.net:http  ESTABLISHED
  TCP    ricci:2608                fx-in-f108.google.com:http  ESTABLISHED
  TCP    ricci:3879                by1msg4246218.gateway.edge.messenger.live.com:18
63   ESTABLISHED

C:\Documents and Settings\Laura>
```

IMPOSTAZIONE LINGER TIME

- SetSoLinger( ) = false, timeout = non significativo. L'applicazione non attende il completamento del protocollo di chiusura che viene eseguito in background
- SetSoLinger( )= true, timeout ≠ 0,
  - l'applicazione si blocca fino allo scadere del timeout, oppure fino a che la procedura di chiusura non viene completata.
  - allo scadere delt ime out, si effettua una procedura di 'hard closure', i dati vengono scartati, non si esegue il protocollo FIN-ACK. Viene invece inviato un segmento RST (reset connection) . Il destinatario solleva una SocketException()
  - attualmente JAVA non consente di distinguere i due eventi.
- SetSolinger( )=true, timeout=0,
  - Si esegue la procedura di hard closure descritta nel caso prcedente

INVIO DI OGGETTI SU CONNESSIONI TCP

- Per inviare oggetti su una connessione TCP occorre definire l'oggetto come istanza di una classe che implementa l'interfaccia Serializable
- E' possibile associare i filtri ObjectInputStream/ ObjectOutputStream agli stream di bytes associati al socket e restituiti da getInputStream/getOutputStream
- Quando creao un ObjectOutputstream viene scritto lo stream header sullo stream. In seguito scrivo gli oggetti che voglio inviare sullo stream
- L'header viene scritto una sola volta quando lo stream viene creato e 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 Server {
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 Client { public static void main (String args[  ]) {
try { Socket socket = new Socket ("localhost",3575);
ObjectInputStream input =
                    new ObjectInputStream (socket.getInputStream ());
String beginMessage = (String) input.readObject();
System.out.print (studente.getMatricola()+" - ");
System.out.print (studente.getNome()+" "+studente.getCognome()+" - ");
System.out.print (studente.getCorsoDiLaurea()+"\n");
String endMessage = (String)input.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: DEADLOCK

- Supponiamo che un'applicazione A1 apra una connessione verso A2 ed invii ad A2 uno stream di oggetti
- A1 associa alla connessione un ObjectOutputStream, mentre A2 associa alla medesima connessione un ObjectInputStream
- Quando A1 crea l'ObjectOutputStream, l'header dello stream viene registrato ed inviato sulla connessione
- Quando A2 crea l' ObjectInputStream
  - la JVM accede tenta di recuperare l'header dello stream dal socket associato alla connessione
  - Se l'header non è presente, la JVM si blocca in attesa di ricevere l'header sul socket
  - ATTENZIONE: per prevenire situazioni di deadlock occorre porre attenzione sull'ordine con cui vengono creati gli stream di Input/Output

OBJECT INPUT/OUTPUT STREAM: DEADLOCK

Se i due partners della connessione eseguono entrambi il seguente frammento di codice (s è il socket associato alla connessione)

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

si verifica una situazione di deadlock..

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 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

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

GRUPPI DI PROCESSI: COMUNICAZIONE

Altre applicazioni:

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

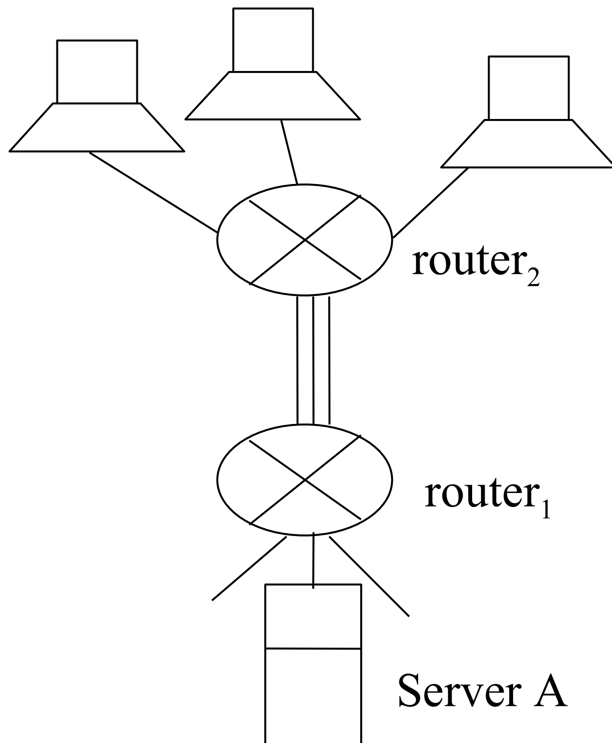
COMUNICAZIONE TRA GRUPPI DI PROCESSI: IMPLEMENTAZIONE

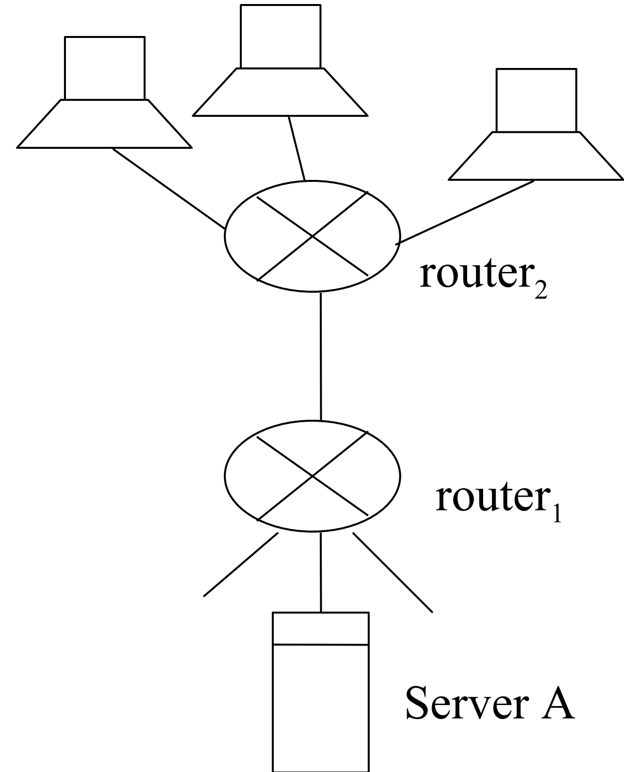
L'implementazione del multicast richiede:

- 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 ($router_2$)





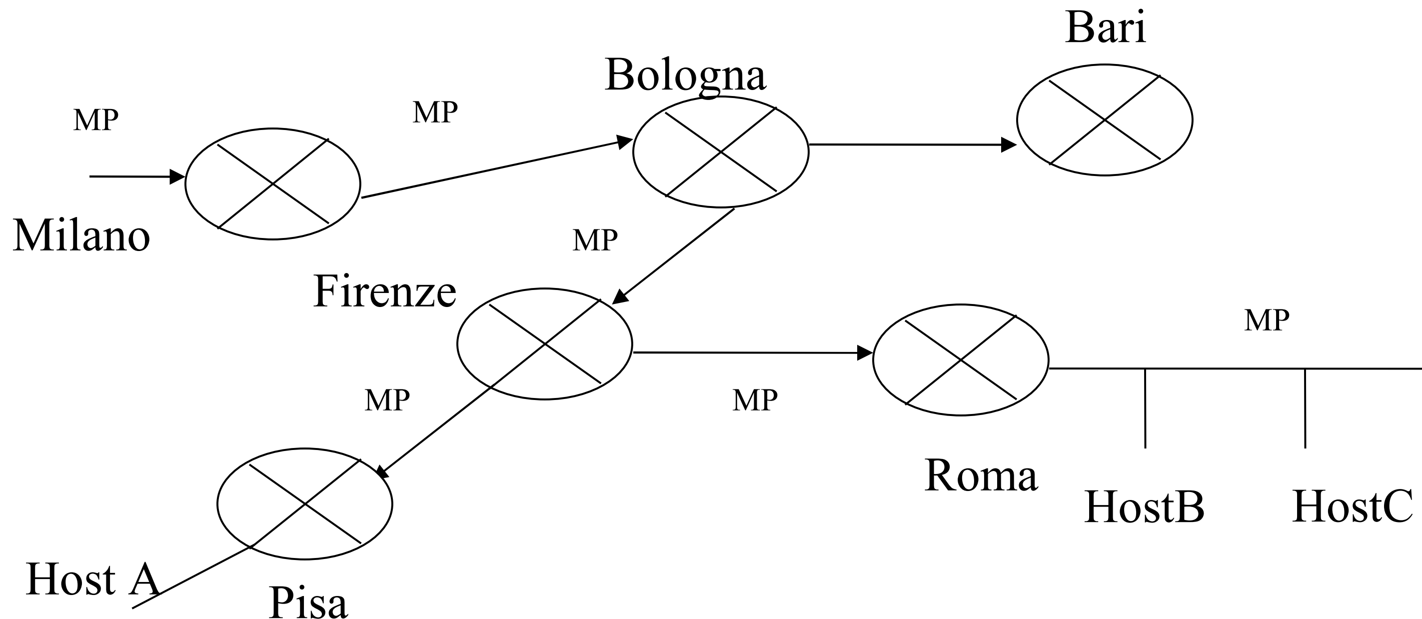
Soluzione 1: $router_1$ invia 3 messaggi distinti con collegamenti di tipo unicast

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.

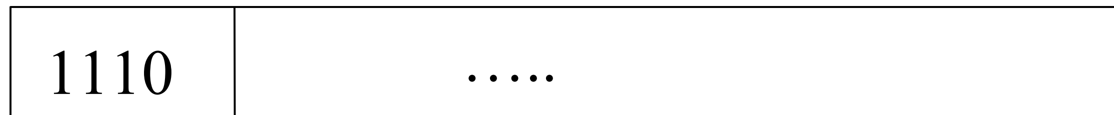
Esempio: pacchetto di multicast spedito da Milano agli hosts Hosta, HostB, HostC



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

- Gruppo di multicast = Insieme di hosts che condividono un indirizzo di multicast

Permanenti : l'indirizzo di multicast viene assegnato da IANA. 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 multicast

- 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, è disponibile un mrouter sulla sottorete locale.
  - routers che non supportano il multicast, possono utilizzare la tecnica del *tunnelling* = trasmissione di pacchetti di multicast mediante unicast UDP

CONNECTIONLESS MULTICAST

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: L'API JAVA

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

```
import java.net.*;
import java.io.*;
public class multicast
    {public static void main (String [  ] args)
    {try {  MulticastSocket ms =new MulticastSocket( );}
  catch (IOException ex) {System.out.println ("errore"); }
    }}
```

- è possibile specificare la porta a cui collegare il multicast socket.
- la classe MulticastSocket estende la DatagramSocket

MULTICAST: L'API JAVA

```
import java.net.*;
import java.io.*;
public class multicast
{public static void main (String [ ] args)
{    try {  MulticastSocket ms =new MulticastSocket(4000);
           InetAddress ia=InetAddress.getByName("226.226.226.226");
          ms.joinGroup (ia);      }
   catch (IOException ex) {System.out.println ("errore"); }}}
```

- operazione necessaria nel caso si vogliano ricevere messaggi dal gruppo di multicast
- lega il multicast socket ad un gruppo di multicast ⇒ tutti i messaggi ricevuti tramite quel socket provengono da quel gruppo
- IOException sollevata se ia non è un indirizzo di multicast

MULTICAST: L'API JAVA

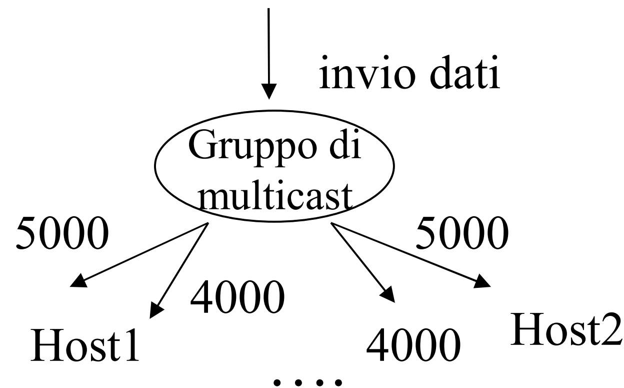
Uso delle porte per multicast sockets:

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: L'API JAVA

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

```
import java.io.*;  import java.net.*;
public class provemulticast {
public static void main (String args[]) throws Exception
      { byte[] buf = new byte[10];
      InetAddress ia = InetAddress.getByName("228.5.6.7");
      DatagramPacket dp = new DatagramPacket (buf, buf.length);
      MulticastSocket ms = new MulticastSocket (4000);
      ms.join(ia);
      ms.receive(dp);    } }
```

se attivo due istanze di provamulticast sullo stesso host, non viene sollevata una BindException (che viene invece sollevata se MulticastSocket è sostituito da un DatagramSocket)

MULTICAST SNIFFER

```
import java.net.*; import java.io.*;
public class multicastsniffer {
public static void main (String[] args)
{InetAddress group = null;
int port = 0;
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);
     byte [ ] buffer = new byte[8192];
     while (true)
     {DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
     ms.receive(dp);
     String s = new String(dp.getData());
     System.out.println(s);}
     } catch (IOException ex){System.out.println (ex);}
```

MULTICAST SNIFFER

```
finally{
        if (ms!= null) {
                try{
                ms.leaveGroup(group);
                ms.close();
                } catch (IOException ex){}
                }}}}
```

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

MULTICAST: L'API JAVA

Per spedire messaggi ad un gruppo di multicast:

- creare un multicastSocket su una porta anonima
- non è necessario collegare il multicast socket ad un gruppo di multicast
- creare un pacchetto inserendo nell'intestazione l'indirizzo del gruppo di multicast a cui si vuole inviare il pacchetto
- Spedire il pacchetto tramite il socket creato

```
public void send (DatagramPacket p) throws IOException
```

MULTICAST: L'API JAVA

Spedizione di un pacchetto tramite un multicast socket

```
import java.io.*;
import java.net.*;
public class multicast {
public static void main (String args[])
{  try   {
     InetAddress ia=  InetAddress.getByName("228.5.6.7");
     byte [  ] data;
     data="hello".getBytes();
     int port= 6789;
     DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
     MulticastSocket ms = new MulticastSocket(6789);
     ms.send(dp);
     } catch(IOException ex){ System.out.println(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 ⇒ 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 qualsiasi 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(1);
```

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.