Reti e Laboratorio 3
Modulo Laboratorio 3
AA. 2024-2025

docente: Laura Ricci

laura.ricci@unipi.it

Lezione 10
costruzione pacchetti UDP - multicast sockets

22/11/2024

COSTRUZIONE DI PACCHETTI DA STRINGHE

- i dati inviati mediante UDP devono essere rappresentati come vettori di bytes
- alcuni metodi per la conversione stringhe/vettori di bytes
  - `Byte[] getBytes()`
    - applicato ad un oggetto `String`
    - restituisce una sequenza di bytes che codificano i caratteri della stringa usando la codifica di default dell'host e li memorizza nel vettore
  - `String (byte[] bytes, int offset, int length)`
    - costruisce un nuovo oggetto di tipo String prelavando length bytes dal vettore bytes, a partire dalla posizione offset
- altri meccanismi per generare pacchetti a partire da dati strutturati:
  - utilizzare i filtri per generare streams di bytes a partire da dati strutturati/ad alto livello

COSTRUZIONE DI PACCHETTI DA DATI STRUTTURATI

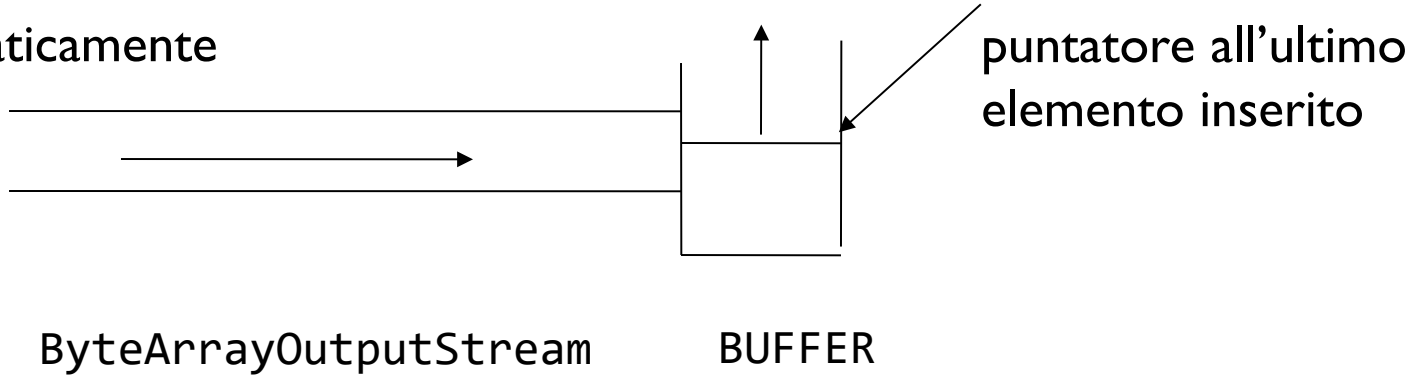
```
public ByteArrayOutputStream ( )
public ByteArrayOutputStream (int size)
```

- gli oggetti istanze di questa classe rappresentano stream di bytes
- ogni dato scritto sullo stream viene riportato in un buffer di memoria a dimensione variabile (dimensione di default = 32 bytes).

```
protected byte buf []
protected int count
```

count indica quanti sono i bytes memorizzati in buf

- quando il buffer si riempie la sua dimensione viene raddoppiata automaticamente

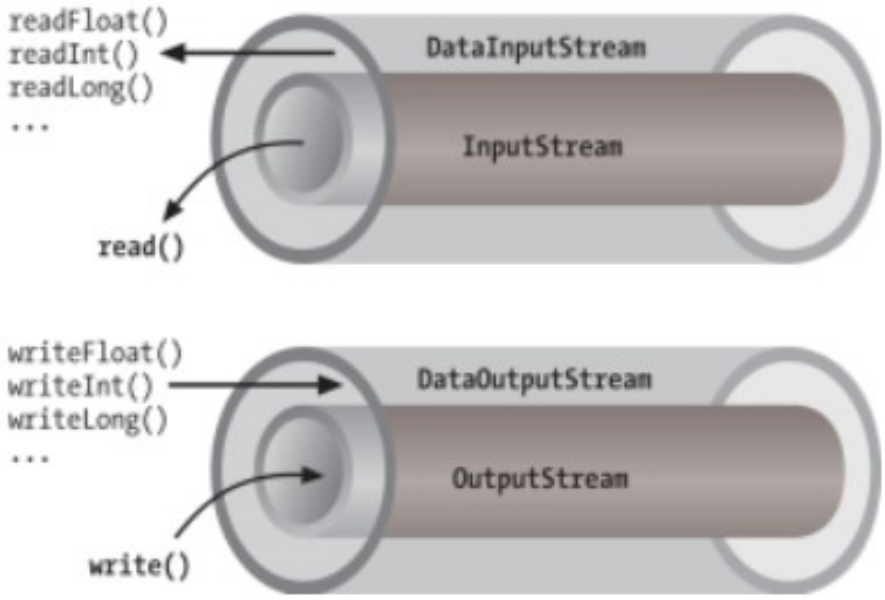


COSTRUZIONE DI PACCHETTI DA DATI STRUTTURATI

- è possibile collegare ad un `ByteArrayOutputStream` un altro filtro

```
ByteArrayOutputStream baos= new ByteArrayOutputStream();
DataOutputStream      dos = new DataOutputStream(baos)
```

- `DataOutput/InputStream` consente di scrivere dati primitivi sullo stream, la trasformazione in bytes è effettuata automaticamente



Data streams in java

BYTE ARRAY OUTPUT STREAMS

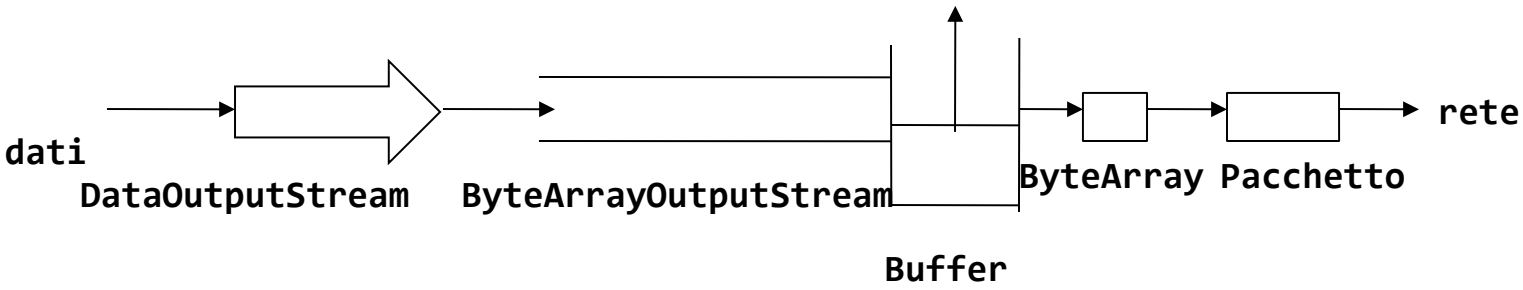
- ad un `ByteArrayOutputStream` può essere collegato un altro filtro

```
ByteArrayOutputStream baos= new ByteArrayOutputStream();
DataOutputStream      dos = new DataOutputStream (baos)
```

- i dati presenti nel buffer B associato ad un `ByteArrayOuputStream` baos possono essere copiati in un array di bytes

```
byte [ ] barr = baos.toByteArray( )
```

- flusso dei dati:

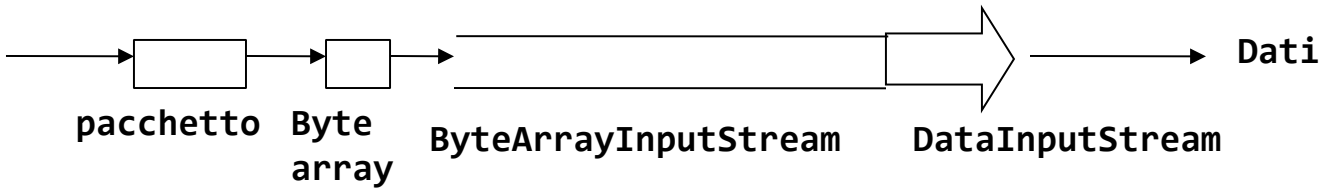


BYTE ARRAY INPUT STREAMS

```
public ByteArrayInputStream ( byte [ ] buf )
public ByteArrayInputStream ( byte [ ] buf, int offset,int length )
```

- creano stream di byte a partire dai dati contenuti nel vettore di byte buf.
- il secondo costruttore copia length bytes iniziando alla posizione offset.
- è possibile concatenare un `DataInputStream`

Flusso dei dati:



LA CLASSE BYTEARRAYOUTPUTSTREAM

metodi per la gestione dello stream:

- `public int size( )` restituisce count, cioè il numero di bytes memorizzati nello stream (non la lunghezza del vettore buf!)
- `public synchronized void reset( )` svuota il buffer, assegnando `0` a count. tutti i dati precedentemente scritti vengono eliminati.

```
baos.reset ( )
```

- `public synchronized byte toByteArray ( )` restituisce un vettore in cui sono stati copiati tutti i bytes presenti nello stream.
  - non modifica count
  - il metodo `toByteArray` non svuota il buffer.

BYTE ARRAY INPUT/OUTPUT STREAMS

Ipotesi semplificativa, non consideriamo perdita/riordinamento di pacchetti:

```
import java.io.*;
import java.net.*;
public class multidatastreamsender{
   public static void main(String args[ ]) throws Exception
    {// fase di inizializzazione
      InetAddress ia=InetAddress.getByName("localhost");
      int port=13350;
      DatagramSocket ds= new DatagramSocket();
      ByteArrayOutputStream bout= new  ByteArrayOutputStream();
      DataOutputStream dout = new DataOutputStream (bout);
      byte [ ] data = new byte [20];
      DatagramPacket dp= new DatagramPacket(data,data.length, ia , port);
```

BYTE ARRAY INPUT/OUTPUT STREAMS

```
for (int i=0; i< 10; i++)
   {dout.writeInt(i);
   data = bout.toByteArray();
   dp.setData(data,0,data.length);
   dp.setLength(data.length);
   ds.send(dp);
   bout.reset( );
   dout.writeUTF("***");
   data = bout.toByteArray( );
   dp.setData (data,0,data.length);
   dp.setLength (data.length);
   ds.send (dp);
  bout.reset(  ); } } }
```

BYTE ARRAY INPUT/OUTPUT STREAMS

Ipotesi semplificativa: non consideriamo perdita/riordinamento di pacchetti

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

public class multidatastreamreceiver
  {public static void main(String args[ ]) throws Exception
    {// fase di inizializzazione
    FileOutputStream fw = new FileOutputStream("text.txt");
    DataOutputStream dr = new DataOutputStream(fw);
    int port =13350;
    DatagramSocket ds = new DatagramSocket (port);
    byte [ ] buffer = new byte [200];
    DatagramPacket dp= new DatagramPacket
                              (buffer, buffer.length);
```

BYTE ARRAY INPUT/OUTPUT STREAMS

```
for (int i=0; i<10; i++)
    {ds.receive(dp);
     ByteArrayInputStream bin= new ByteArrayInputStream
                                (dp.getData(),0,dp.getLength());
    DataInputStream ddis= new DataInputStream(bin);
    int x = ddis.readInt();
    dr.writeInt(x);
    System.out.println(x);
    ds.receive(dp);
    bin= new ByteArrayInputStream(dp.getData(),0,dp.getLength());
    ddis= new DataInputStream(bin);
    String y=ddis.readUTF( );
    System.out.println(y); }}}
```

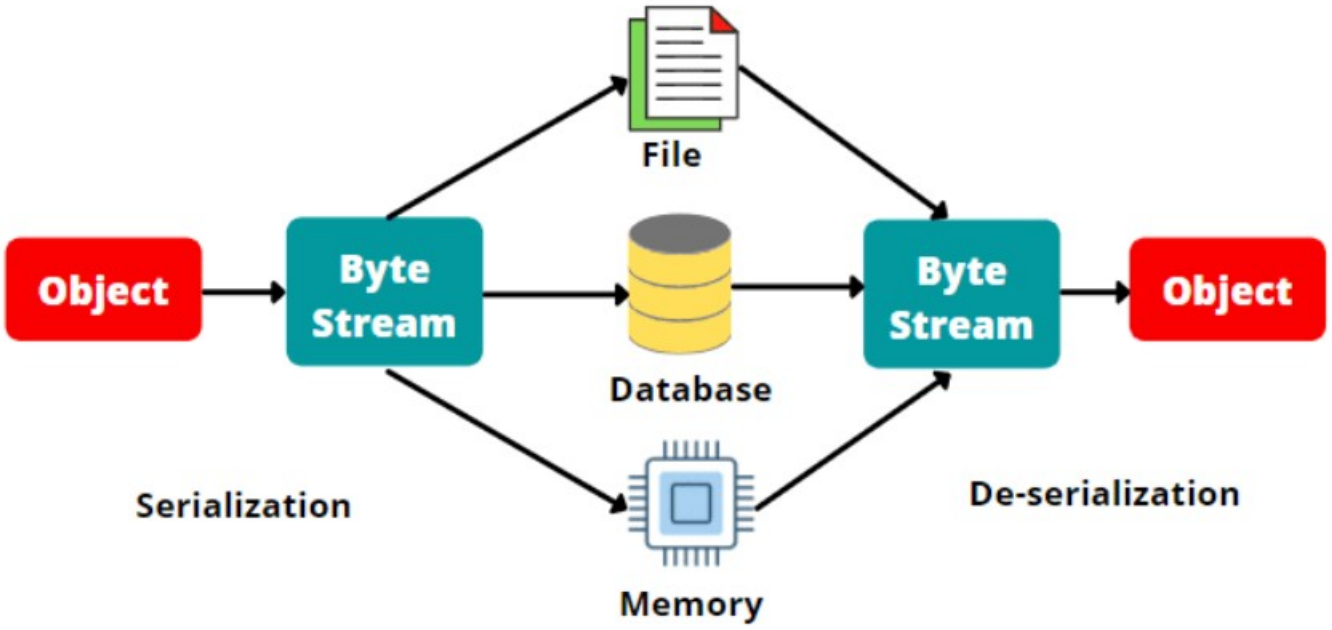
BYTE ARRAY INPUT/OUTPUT STREAMS

- nel programma precedente, la corrispondenza tra la scrittura nel mittente e la lettura nel destinatario potrebbe non essere più corretta
- esempio:
  - il mittente alterna la spedizione di pacchetti contenenti valori interi con pacchetti contenenti stringhe
  - il destinatario alterna la lettura di interi e di stringhe
  - ma se un pacchetto viene perso: per il destinatario scritture/letture possono non corrispondere
- realizzazione di UDP affidabile: utilizzo di ack per confermare la ricezione + identificatori unici

UDP E SERIALIZZAZIONE

- inviare oggetti in pacchetti?
- usare la serializzaione per genrìerare uno stream di Byte
- collegare l'outputstream generato ad ad un `ByteArrayOutputStream`

```
ByteArrayOutputStream baos= new ByteArrayOutputStream();
ObjectOutputStream    dos = new ObjectOutputStream(baos)
```



CONCLUSIONI: STREAMS ARE EVERYWHERE!

- trasmissione connection oriented: una connessione viene modellata con uno stream.

  invio di dati scrittura sullo stream

  ricezione di dati lettura dallo stream

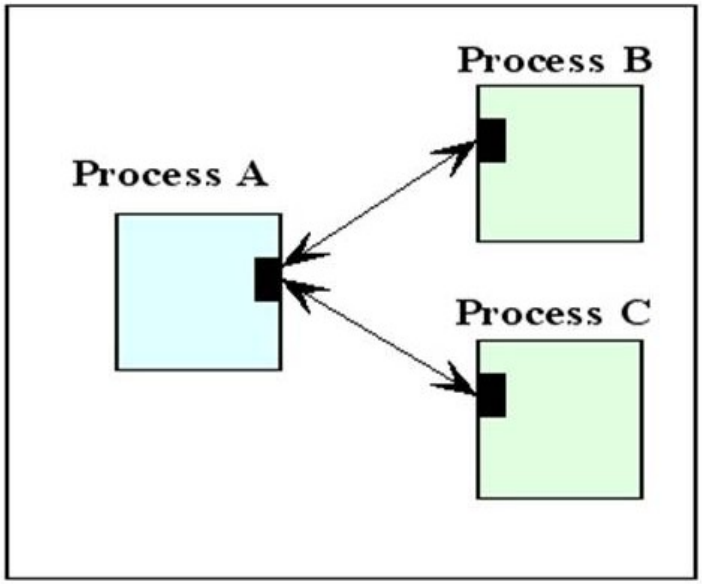
- trasmissione connectionless: stream utilizzati per la generazione dei pacchetti:

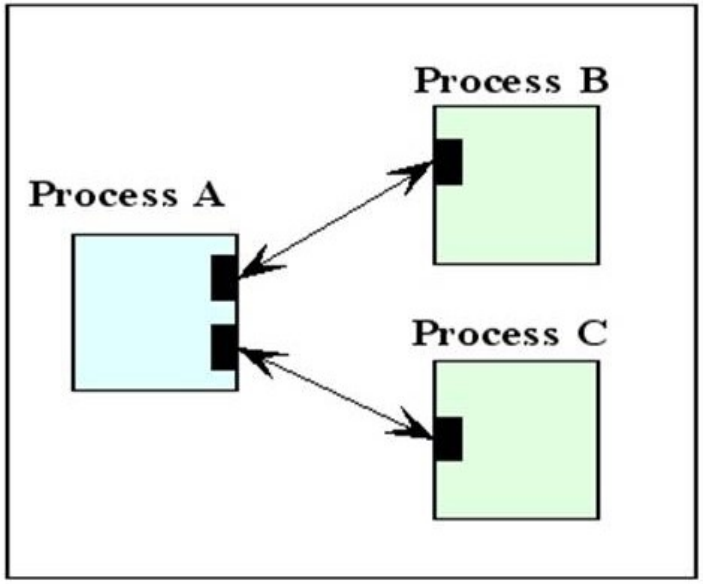
  `ByteArrayOutputStream`, consentono la conversione di uno stream di bytes in un vettori di bytes da spedire con i pacchetti UDP

  `ByteArrayInputStream,` converte un vettore di bytes in uno stream di byte.

CONCLUSIONI: SOCKET UDP

- possibile usare lo stesso socket per spedire pacchetti verso destinatari diversi
- processi (applicazioni) diverse possono spedire pacchetti sullo stesso socket in questo caso l'ordine di arrivo dei massaggi è non deterministico, in accordo con il protocollo UDP
- ...ma è possibile anche utilizzare socket diversi per comunicazioni diverse

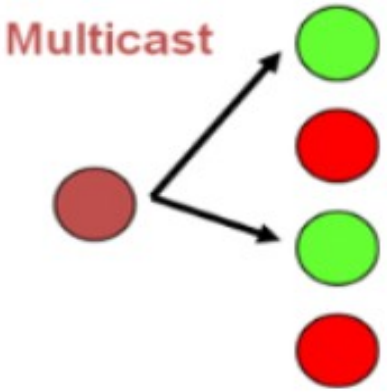


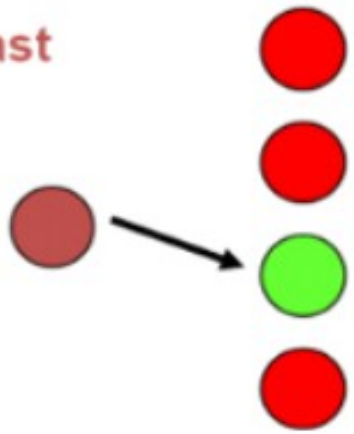


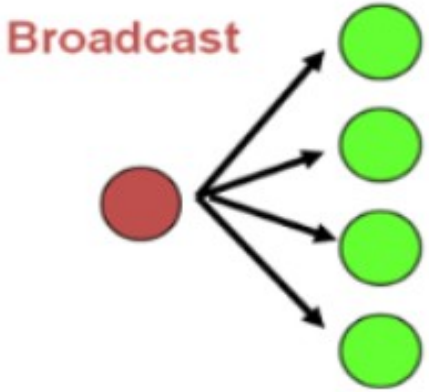
MULTICASTING

- inviare dati da un host ad un insieme di altri nodi
  - non tutti gli host della rete: solo quelli che hanno espresso interesse ad unirsi ad un gruppo di multicast
- esempi:
  - diverse applicazioni usano IP multicasting per notificare/scoprire dinamicamente i servizi in una rete, senza usare DNS o terze parti
- la maggior parte del lavoro viene svolto dai router ed è trasparente al programmatore
  - i router assicurano che il pacchetto spedito dal mittente sia consegnato a tutti gli host interessati
  - usa `Time-To-Live` IP: massimo numero di router che il datagramma può attraversare
  - il problema maggiore è che non tutti i router supportano il multicast

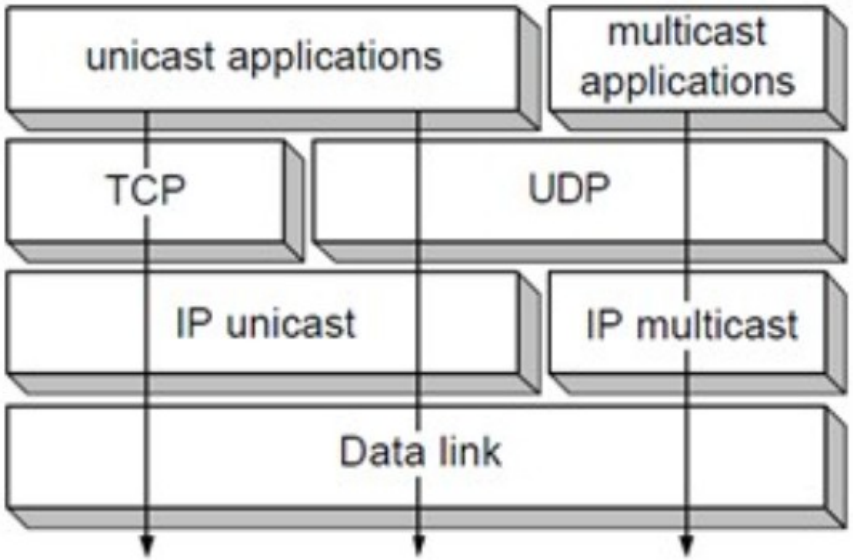
UNICASTING/MULTICASTING/BROADCASTING





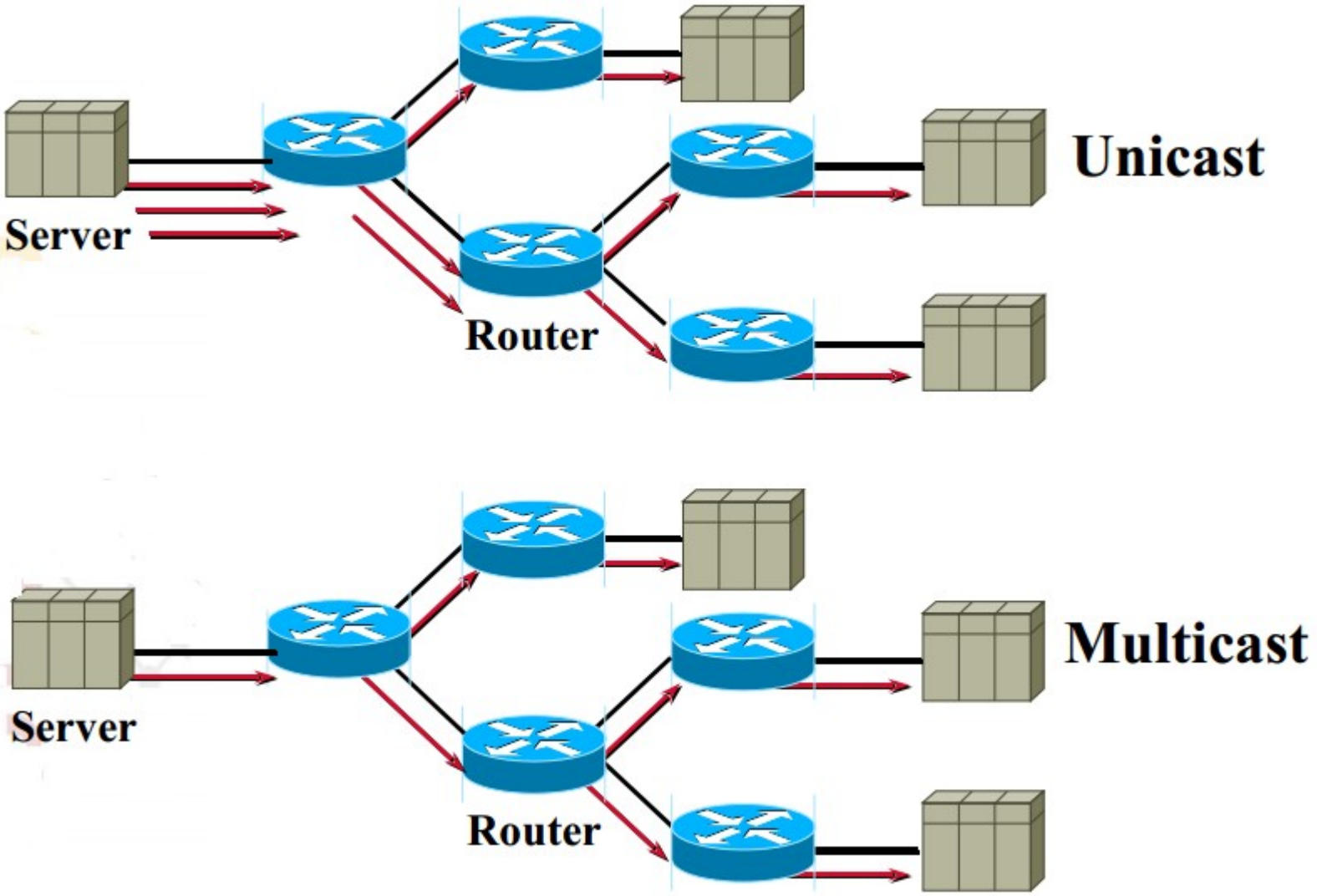


MULTICAST E PROTOCOL STACK



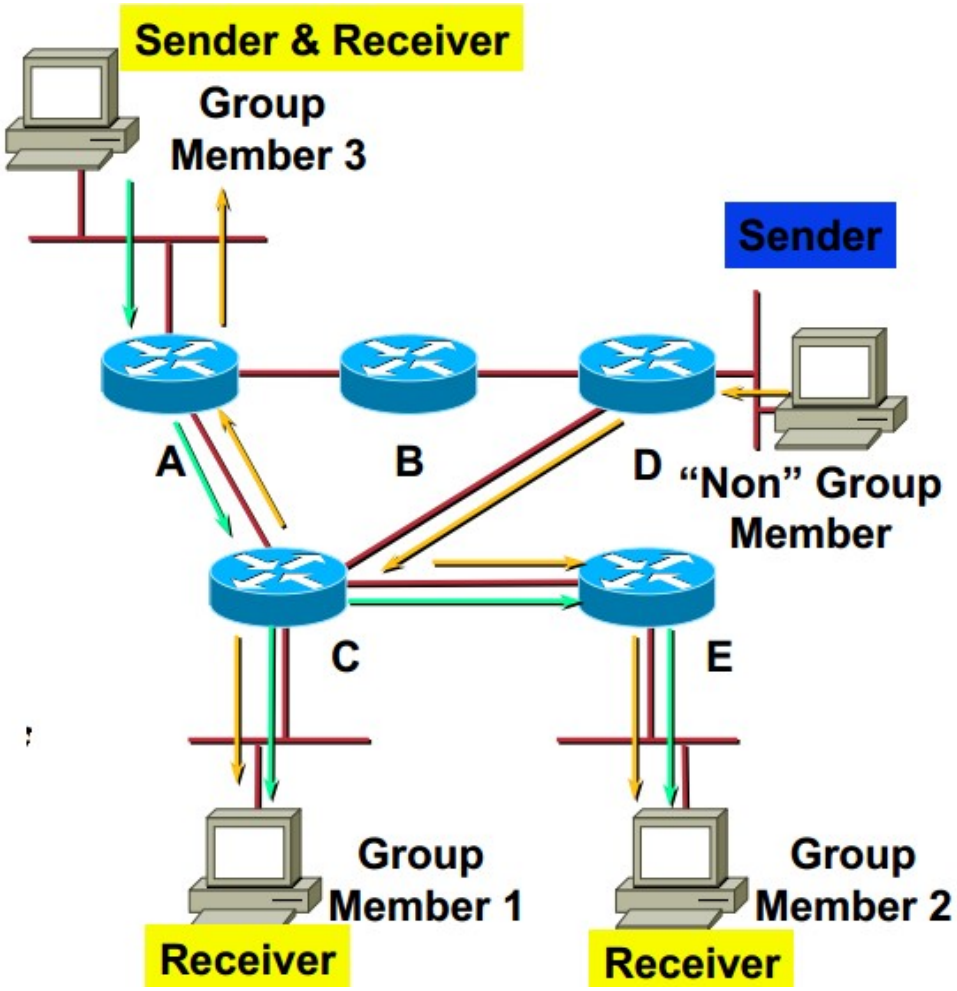
- IP multicasting utilizza UDP
- non esiste multicast TCP!

UNICAST E MULTICAST: CONFRONTO



GRUPPI MULTICAST

- IP multicast basato sul concetto di gruppo
  - insieme di processi in esecuzione su host diversi
- tutti i membri di un gruppo di multicast ricevono un messaggio spedito su quel gruppo
- non occorre essere membri del gruppo per inviare i messaggi su di esso
- gestiti a livello IP dal protocollo IGMP



MULTICAST API

deve contenere almeno primitive per:

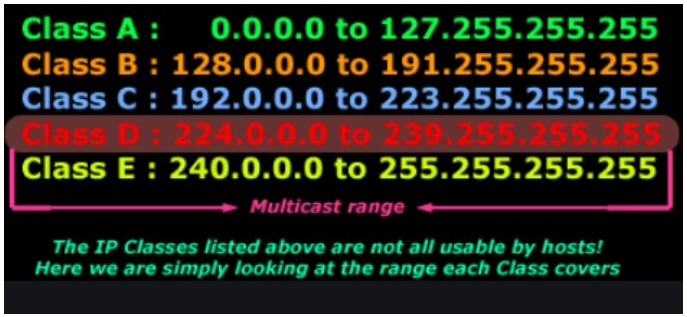
- unirsi ad un gruppo di multicast
- lasciare un gruppo
- spedire messaggi ad un gruppo
  - il messaggio viene recapitato a tutti i processi che fanno parte del gruppo in quel momento
- ricevere messaggi indirizzati ad un gruppo

Il supporto deve fornire

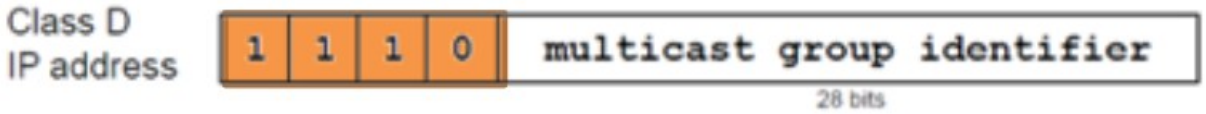
- uno schema di indirizzamento per identificare univocamente un gruppo
- un meccanismo che registri la corrispondenza tra un gruppo ed i suoi partecipanti (`IGMP`)
- un meccanismo che ottimizzi l'uso della rete nel caso di invio di pacchetti ad un gruppo di multicast

SCHEMA DI INDIRIZZAMENTO MULTICAST

- basato sull'idea di riservare un insieme di indirizzi IP per il multicast



- IPV4: indirizzo di un gruppo è un indirizzo in classe D
  - [224.0.0.0 – 239.255.255.255]



i primi 4 bit del primo ottetto = 1110

i restanti bit identificano il particolare gruppo

IPV6: tutti gli indirizzi di multicast iniziano con FF o 1111 1111 in binario

MULTICAST: CARATTERISTICHE

- utilizza il paradigma connectionless (UDP):
  - sarebbero richieste n×(n-1) connessioni per un gruppo di n applicazioni, se si usasse la comunicazione connection oriented
- comunicazione connectionless adatta per il tipo di applicazioni verso cui è orientato il multicast
  - trasmissione di dati video/audio: invio dei frames di un'animazione
  - è più accettabile la perdita occasionale di un frame piuttosto che un ritardo costante tra la spedizione di due frames successivi
- si perde l'affidabilità della trasmissione, ma esistono librerie JAVA non standard che forniscono multicast affidabile
  - garantiscono che il messaggio venga recapitato una sola volta a tutti i processi del gruppo
  - possono garantire altre proprietà relative all'ordinamento con cui i messaggi spediti al gruppo di multicast vengono recapitati ai singoli partecipanti

SOCKET MULTICAST

`Java.net.MulticastSocket`

- estende `DatagramSocket`
- socket su cui ricevere i messaggi da un gruppo di multicast
- effettua overriding dei metodi esistenti in `DatagramSocket` e fornisce nuovi metodi per l'implementazione di funzionalità tipiche del 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"); }
          }
          }
```

UNIRSI AD UN GRUPPO MULTICAST

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

- `joinGroup` 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 l'indirizzo di multicast è errato

RICEVERE PACCHETTI DA MULTICAST

```
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.joinGroup(ia);
      ms.receive(dp);    } }
```

- se attivo due istanze del programma sullo stesso host (la reuse socket settata true) non viene sollevata una `BindException`
- l'eccezione verrebbe invece sollevata se si utilizzasse un `DatagramSocket`
- servizi diversi in ascolto sulla stessa porta di multicast
- non esiste una corrispondenza biunivoca porta-servizio

JAVA API PER MULTICAST

- ogni socket multicast ha una proprietà, la reuse socket, che se settata a true, da la possibilità di associare più socket alla stessa porta

- nelle ultime versioni è possibile impostarne il valore

```
try   {sock.setReuseAddress(true);}
catch (SocketException se) {…}
```

- nelle prime versioni di JAVA la proprietà era settata per default a true

CONCLUSIONI

- TCP: trasmissione vista come uno stream continuo di bytes provenienti dallo stesso mittente

- UDP : trasmissione orientata ai messaggi: "preserves message boundaries"
  - send, receive `DatagramPacket`
  - socket come una mailbox: in essa possono essere inseriti messaggi in arrivo da diverse sorgenti (mittenti) o i messaggi inviati a diverse destinazioni
  - ogni ricezione si riferisce ad un singolo messaggio inviato mediante una unica send
    - dati inviati dalla stessa send non possono essere ricevuti in receive diverse

PARAMETRI DI CONFIGURAZIONE: PROPERTIES

- nel progetto, sia il client che il server dovranno essere configurati impostando il valore di diversi parametri di configurazione
- utile la classe `JAVA Properties`
  - fa parte del pacchetto `java.util`
  - è utilizzata per gestire coppie chiave-valore, dove sia le chiavi che i valori sono stringhe.
  - è una specializzazione della classe `Hashtable`
  - può essere usata usata per leggere file di configurazione, impostazioni dell'applicazione, etc.

PARAMETRI DI CONFIGURAZIONE: PROPERTIES

```
package Prop;
import java.util.Properties; import java.io.*;
public class Prop {
  public static void main (String args[])
    { File configFile = new File("config.properties");
       try {
          FileReader reader = new FileReader(configFile);
          Properties props = new Properties();
          props.load(reader);
          String host = props.getProperty("host");
          System.out.print("Host name is: " + host+"\n");
          String port = props.getProperty("port");
          System.out.print("Port number is: " + port);
          reader.close();
       } catch (FileNotFoundException ex) {
                    // file does not exist
       } catch (IOException ex) {
                    // I/O error
       }}}
```

ASSIGNMENT N. 10

- PING è una utility per la valutazione delle performance della rete utilizzata per verificare la raggiungibilità di un host su una rete IP e per misurare il round trip time (RTT) per i messaggi spediti da un host mittente verso un host destinazione
- lo scopo di questo assignment è quello di implementare un server PING ed un corrispondente client PING che consenta al client di misurare il suo RTT verso il server
- la funzionalità fornita da questi programmi deve essere simile a quella della utility PING disponibile in tutti i moderni sistemi operativi. La differenza fondamentale è che si utilizza UDP per la comunicazione tra client e server, invece del protocollo ICMP (Internet Control Message Protocol)
- inoltre, poichè l'esecuzione dei programmi avverrà su un solo host o sulla rete locale ed in entrambi i casi sia la latenza che la perdita di pacchetti risultano trascurabili, il server deve introdurre un ritardo artificiale ed ignorare alcune richieste per simulare la perdita di pacchetti

PING CLIENT

- accetta due argomenti da linea di comando: nome e porta del server. Se uno o più argomenti risultano scorretti, il client termina, dopo aver stampato un messaggio di errore del tipo `ERR -arg x`, dove x è il numero dell'argomento

- utilizza una comunicazione UDP per comunicare con il server ed invia 10 messaggi al server, con il seguente formato:

  `PING seqno timestamp`

  in cui seqno è il numero di sequenza del `PING` (tra `0-9`) ed il timestamp (in millisecondi) indica quando il messaggio è stato inviato

- non invia un nuovo `PING` fino a che non ha ricevuto l'eco del `PING` precedente, oppure è scaduto un timeout

PING CLIENT

- stampa ogni messaggio spedito al server ed il RTT del ping oppure un * se la risposta non è stata ricevuta entro 2 secondi
- dopo che ha ricevuto la decima risposta (o dopo il suo timeout), il client stampa un riassunto simile a quello stampato dal `PING UNIX`

```
---- PING Statistics ----
10 packets transmitted, 7 packets received, 30% packet loss
round-trip (ms) min/avg/max = 63/190.29/290
```

- il `RTT` medio è stampato con 2 cifre dopo la virgola

PING SERVER

- è essenzialmente un echo server: rimanda al mittente qualsiasi dato riceve
- accetta un argomento da linea di comando: la porta, che è quella su cui è attivo il server + un argomento opzionale, il seed, un valore long utilizzato per la generazione di latenze e perdita di pacchetti. Se uno qualunque degli argomenti è scorretto, stampa un messaggio di errore del tipo `ERR -arg x`, dove x è il numero dell'argomento
- dopo aver ricevuto un `PING`, il server determina se ignorare il pacchetto (simulandone la perdita) o effettuarne l'eco. La probabilità di perdita di pacchetti di default è del `25%`.
- se decide di effettuare l'eco del `PING`, il server attende un intervallo di tempo casuale per simulare la latenza di rete
- stampa l'indirizzo `IP` e la porta del client, il messaggio di `PING` e l'azione intrapresa dal server in seguito alla sua ricezione (`PING` non inviato, oppure `PING` ritardato di `x ms`)

PING SERVER

```
java PingServer 10002 123
128.82.4.244:44229> PING 0 1360792326564 ACTION: delayed 297 ms
128.82.4.244:44229> PING 1 1360792326863 ACTION: delayed 182 ms
128.82.4.244:44229> PING 2 1360792327046 ACTION: delayed 262 ms
128.82.4.244:44229> PING 3 1360792327309 ACTION: delayed 21 ms
128.82.4.244:44229> PING 4 1360792327331 ACTION: delayed 173 ms
128.82.4.244:44229> PING 5 1360792327505 ACTION: delayed 44 ms
128.82.4.244:44229> PING 6 1360792327550 ACTION: delayed 19 ms
128.82.4.244:44229> PING 7 1360792327570 ACTION: not sent
128.82.4.244:44229> PING 8 1360792328571 ACTION: not sent
128.82.4.244:44229> PING 9 1360792329573 ACTION: delayed 262 ms
```

PING CLIENT

```
java PingClient localhost 10002
PING 0 1360792326564 RTT: 299 ms
PING 1 1360792326863 RTT: 183 ms
PING 2 1360792327046 RTT: 263 ms
PING 3 1360792327309 RTT: 22 ms
PING 4 1360792327331 RTT: 174 ms
PING 5 1360792327505 RTT: 45 ms
PING 6 1360792327550 RTT: 20 ms
PING 7 1360792327570 RTT: *
PING 8 1360792328571 RTT: *
PING 9 1360792329573 RTT: 263 ms
---- PING Statistics ----
10 packets transmitted, 8 packets received, 20% packet loss
round-trip (ms) min/avg/max = 20/158.62/299
```

JAVA PINGER

Invocazione corretta client/server:

```
java PingClient
Usage: java PingClient hostname port
java PingServer
Usage: java PingServer port [seed]
```

Invocazione non corretta client/server:

```
java PingClient atria three
ERR - arg 2
java PingServer abc
ERR - arg 1
```