Lezione n.7
LPR – Informatica Applicata
Multicast UDP
Thread Pool: Politiche di Saturazione

26/4/2010

Laura Ricci

DATAGRAM SOCKET API: RIASSUNTO DELLE PUNTATE PRECEDENTI

- lo stesso Datagram Socket può essere utilizzato per spedire messaggi verso destinatari diversi
- processi diversi possono inviare datagrams sullo stesso socket di un processo destinatario
- send non bloccante

  se il destinatario non è in esecuzione quando il mittente esegue la send, il messaggio può venir scartato
- receive bloccante

  uso di timeouts associati al socket per non bloccarsi indefinitamente sulla receive
- i messaggi ricevuti possono essere troncati se la dimensione del buffer del destinatario è inferiore a quella del messaggio spedito

DATAGRAM SOCKET API: RIASSUNTO DELLE PUNTATE PRECEDENTI

- protocollo UDP: non implementa controllo del flusso:

  se la frequenza con cui il mittente invia i messaggi è sensibilmente maggiore di quella con cui il destinatario li riceve (li preleva dal buffer di ricezione) è possibile che alcuni messaggi sovrascivano messaggi inviati in precedenza

- Esempio: CountDown Server (vedi esercizio lezione precdente).

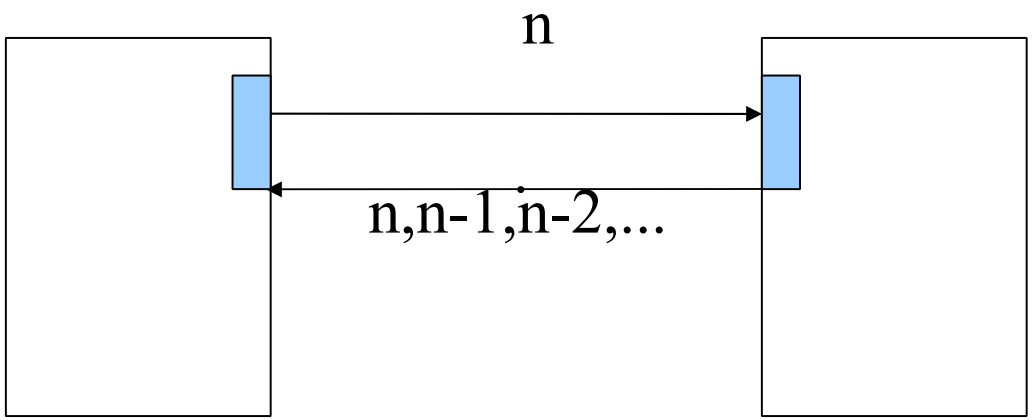
  Il client invia al server un valore di n molto grande

  (provare valori>1000). Allora:

  - il server deve inviare al client un numero molto alto di pacchetti
  - il tempo che intercorre tra l'invio di un pacchetto e quello del pacchetto successivo è basso
  - dopo un certo numero di invii il buffer del client si riempie⇒ perdita di pacchetti

COUNT DOWN SERVER UDP

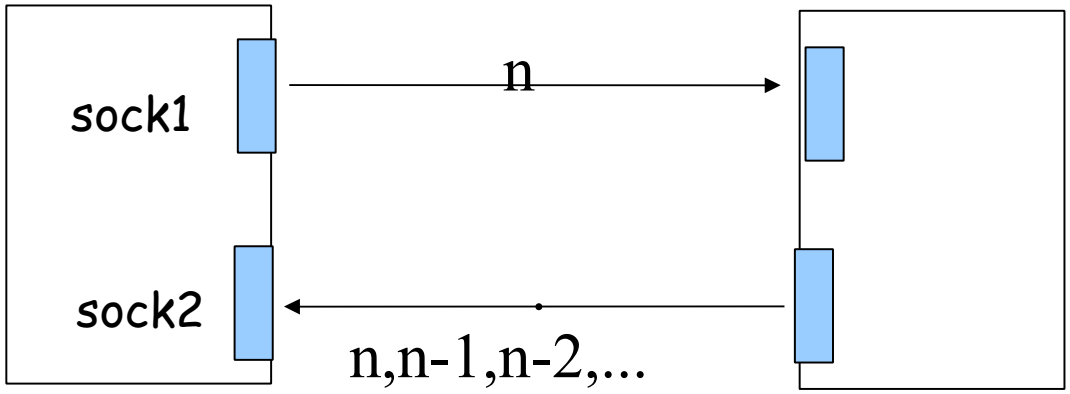
CountDownClient CountDownServer



- Si può utilizzare lo stesso socket per inviare il valore n e per ricevere i risultati
- Quando il CountDownClient invia il valore n il CountDownServer deve aver allocato il socket, altrimenti il Messaggio viene perduto

COUNT DOWN SERVER UDP

CountDownClient CountDownServer



- Posso utilizzare sockets diversi per la spedizione/ricezione
- In questo caso può accadere che sock2 non sia ancora stato creato quando CountDownServer inizia ad iniziare la sequenza di numeri
- E' possibile che il CountDownClient si blocchi sulla receive poiché i dati inviati dal CountDownServer sono stati inviati prima della creazione del socket e quindi sono andati persi

COMUNICAZIONE UNO A MOLTI

- Diverse applicazioni di rete richiedono che un messaggio inviato sia ricevuto da un gruppo di hosts
- Comunicazione uno a molti = comunicazione in multicast

Applicazioni che possono richiedere il multicast:

- videoconferenze: un segnale audio video generato su un nodo della rete deve essere ricevuto dagli hosts associati ai partecipanti alla videoconferenza
- massive multiplayer games: alto numero di giocatori che intergiscono in un mondo virtuale
- Instant messaging
- DNS (Domain Name System): aggiornamenti delle tabelle di naming inviati a gruppi di DNS

COMUNICAZIONE UNO A MOLTI

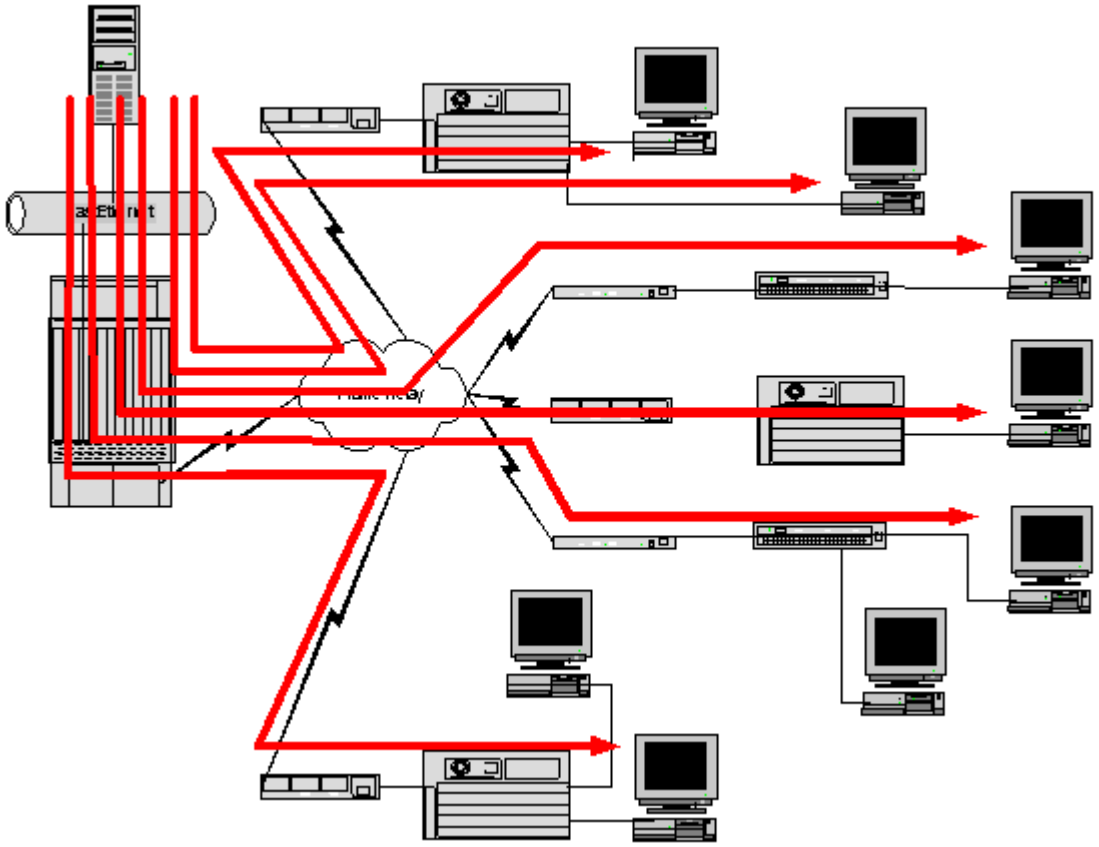
Comunicazione di tipo multicast

- 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

Multicast API: deve contenere primitive per

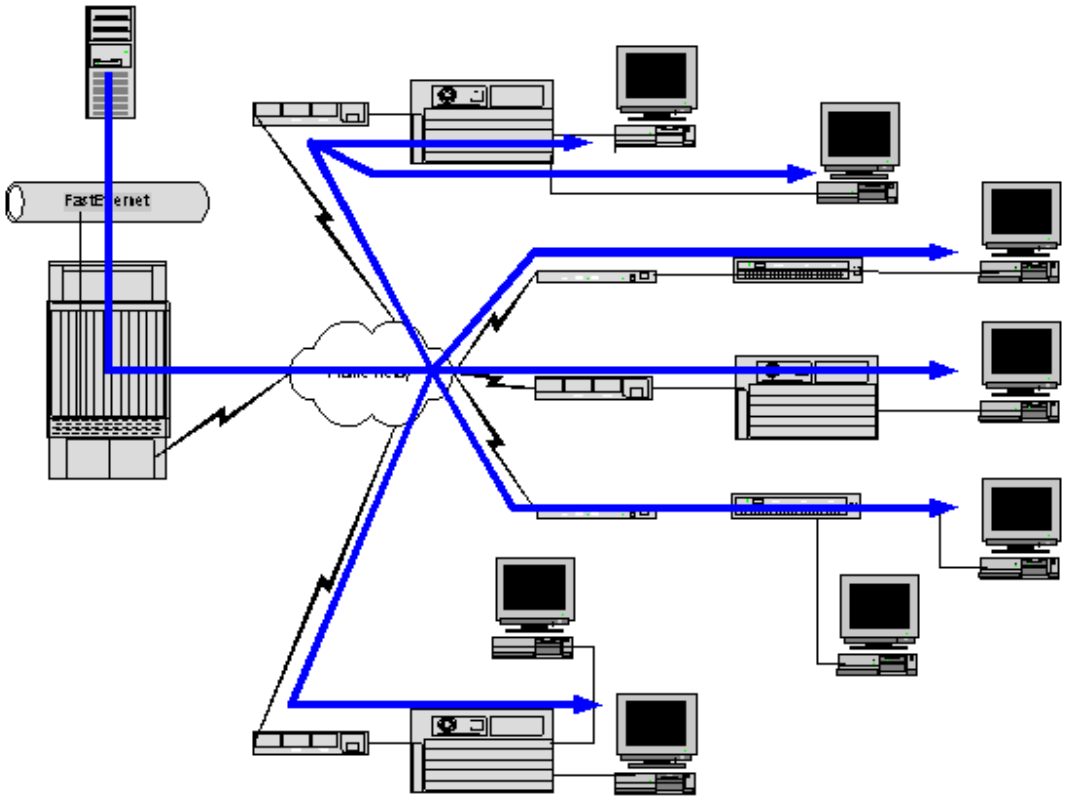
- unirsi ad un gruppo di multicast (join). E' richiesto uno schema di indirizzamento per identificare univocamente un gruppo.
- lasciare un gruppo di multicast (leave).
- spedire messaggi ad un gruppo. Il messaggio viene recapitato a tutti i processi che fanno parte del gruppo in quel momento
- ricevere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

UNICAST VIDEO/AUDIO TRASMISSION



Streaming video/audio in unicast: un flusso di dati per ogni possibile destinatario

MULTICAST VIDEO/AUDIO TRASMISSION

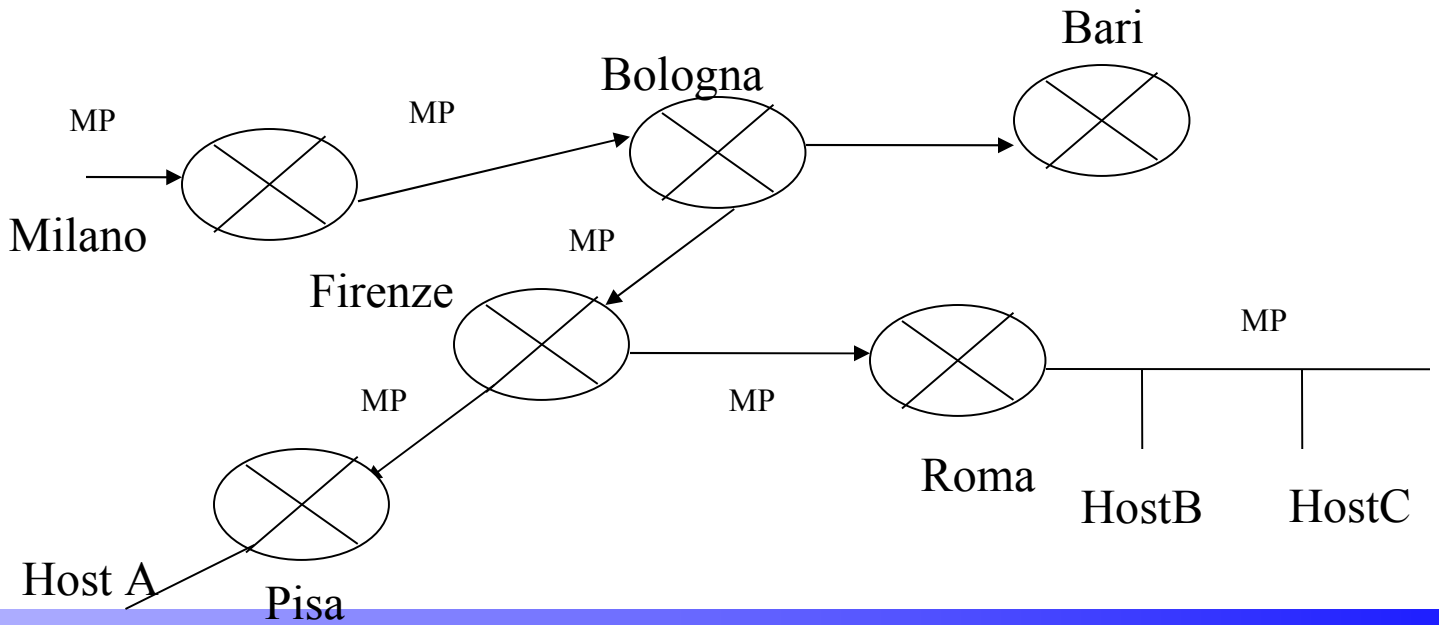


Streaming video/audio in multicast: un unico flusso di dati

MULTICAST: MOTIVAZIONE

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



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

INDIRIZZAMENTO GRUPPI DI MULTICAST

- Ogni gruppo viene identificato mediante un particolare indirizzo IP
- Indirizzo di multicast: IANA (Internet Assigned Number Authority) ha riservato al multicast gli indirizzo IP di classe D, cioè appartenenti all' intervallo [224.0.0.0 – 239.255.255.255]

1110	…..

- i primi 4 bits dell'indirizzo identificano che si tratta di un indirizzo di multicast, i restanti 28 bits individano il particolare gruppo di multicast
- alcuni indirizzi in calsse D sono riservati e non possono essere utilizzati per creare nuovi gruppi di multicast
- è possibile associare un nome simbolico ad un gruppo di multicast 224.0.1.1 associato a ntp.mcast.net (network time protocol distributed service)

INDIRIZZI DI MULTICAST RISERVATI

- Link-Local Block: contiene gli indirizzi di multicast appartenenti all'intervallo [224.0.0.0 - 224.0.0.255]
  - Lo scope di questi indirizzi è la rete locale: sono riservati per i protocolli di routing e per altre funzionalità a livello di rete locale
  - I pacchetti contenenti indirizzi di multicast appartenenti a questo intervallo vengono trasemssi con un TTL (time To Live)= uguale ad 1 e quindi non possono essere trasemssi fuori dalla rete locale

    224.0.0.1 tutti gli host della rete locale

    224.0.0.2 tutti i multicast routers della rete locale
- Internet Control Block: contiene indirizzi di multicast appartenenti all'intervallo [224.0.1.0 -224.0.1.255].
  - Indirizzi statici utilizzati per protocolli che devono propagare informazione su tutta la rete

INDIRIZZAMENTO GRUPPI DI MULTICAST

- L'allocazione degli indirizzi di multicast su Internet è una procedura molto complessa, che prevede un ampio numero di casi
- Ad ogni indirizzo di multicast viene associato uno scope
  - tutta la rete, un dominio amministrativo, la rete locale,....
- Gli scenari più frequenti
  - Indirizzi Statici: assegnati da una autorità di controllo. Sono utilizzati da particolari protocolli/ applicazioni.
    - L'indirizzo rimane assegnato a quel gruppo, anche se, in un certo istante non ci sono partecipanti
  - Indirizzi Dinamici: si utizzano protocolli particolari che consentono di evitare che lo stesso indirizzo di multicast sia assegnato a due gruppi diversi
    - Esistono solo fino al momento in cui esiste almeno un partecipante.
    - Richiedono un protocollo per l'assegnazione dinamica degli indirizzi

INDIRIZZI DI MULTICAST STATICI

- Gli indirizzi statici possono essere assegnati
  - da IANA
  - dall'amministratore di rete
- Gli indirizzi statici assegnati da IANA
  - sono validi per tutti gli host della rete e possono essere 'cablati' nel codice delle applicazioni
  - ad esempio l'indirizzo di multicast 224.0.1.1 è assegnato al network time protocol, protocollo utilizzato per sincronizzare i clocks di un insieme di hosts
- Gli indirizzi statici assegnati dall'amministratore di una certa organizzazione
  - valgono per tutti gli host della rete di quella amministrazione
  - vengono indicati come Administratively Scoped Blocks

ADMINISTRATIVELY SCOPED BLOCKS

- Administratively Scoped Blocks
  - Blocco di indirizzi di multicast nel range [239.0.0.0 - 239.255.255.255]
- Gli indirizzi in questo range non possono generare conflitti con altri indirizzi di multicast in Internet, poiché
  - i multicast routers vengono configurati in modo che gli indirizzi appartenenti a questo intervallo non superi i bordi della rete appartenente ad un certo dominio amministrativo
  - non è possibile che un indirizzo in questo range generi un conflitto con un indirizzo di multicast assegnato ad un altro gruppo in un altro dominio
- Un indirizzo appertenente ad un Administratively Scoped Block può quindi essere riutilizzato in regioni diverse della rete
- L'amministratore di rete può a sua volta suddividere questo range in regioni più piccole per limitare la diffusione dei pacchetti diffusi da particolari applicazioni

INDIRIZZI DI MULTICAST DINAMICI

- Per ottenere un indirizzo di multicast in modo dinamico, è necessario utilizzare un protocollo opportuno

- Nell'ambito di una sottorete gestita mediante un unico dominio amministrativo

  - Multicast Address Dynamic Client Allocation Protocol (MADCAP) .

- Nell'ambito più generale della rete

  - Multicast Address Set Claim (MASC), SSM , SDR

INDIRIZZI MULTICAST DINAMICI

- Multicast Address Dynamic Client Allocation Protocol (MADCAP)
  - basato sulla definizione di un server MADCAP
  - i client ottengono un indirizzo multicast da tale server (multicast address lease), in maniera analoga alla procedura con cui si ottiene un indirizzo IP da un server DHCP
  - il discovery del server MADCAP avviene mandando un messaggio che utilizza un indirizzo di multicast fisso
  - dopo aver scoperto il server opportuno, il client invia un messaggio di richiesta di un 'lease' per un indirizzo di multicast
- Requisiti per l'utilizzo di MADCAP:
  - Il linguaggio utilizzato per lo sviluppo della applicazione deve consentire di interfacciarsi con un server MADCAP
  - L'amministratore deve configurare opportunamente uno o più secrver MADCAP

INDIRIZZI MULTICAST DINAMICI

SDR -Session Directory

- viene definito un gruppo di multicast associato ad un indirizzo di multicast statico
- ogni nuovo gruppo,deve essere 'annunciato' su quel gruppo di multicast.
  - l'annuncio contiene l'indirizzo IP prescelto per il nuovo gruppo.
- ogni host colleziona e memorizza in un proprio database tutti gli annunci ricevuti dal gruppo di multicast.
- il database contiene tutti gli indirizzi di multicast già utilizzati
- quando si vuole creare un nuovo gruppo, il sistema propone un indirizzo di multicast non contenuto all'interno del database
- non garantisce assenza di conflitti: due diversi host potrebbero
  - decidere nello stesso momento di creare due gruppi di multicast diversi
  - assegnare lo stesso indirizzo ai due gruppi

MULTICAST ROUTERS

- NOTA BENE: la maggior parte degli indirizzi assegnati in modo permanente hanno come prefisso

  224.0, 224.1, 224.2, 239

per cui scegliete indirizzi fuori da questi range (vedere tutorial sulla pagina del corso)

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

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

- 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.

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
- Definito in alcune librerie non incluse nell'API JAVA

MULTICAST: L'API JAVA

JAVA definisce la classe MulticastSocket, una sottoclasse di DatagramSocket, con lo scopo di descrivere le funzionalità peculiari di un socket di multicast

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"); }
    }}
```

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

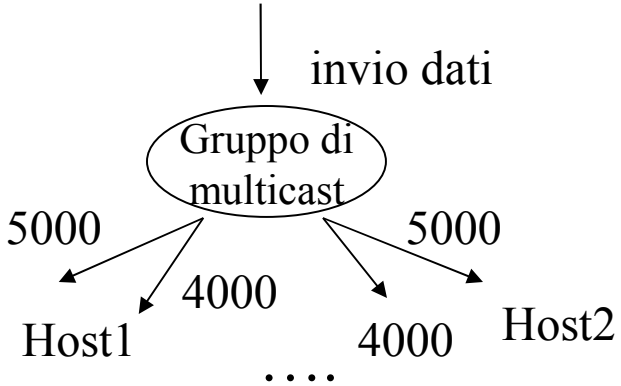
MULTICAST: L'API JAVA

Unicast

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

Multicast

- IP Address individua un gruppo di hosts,
- La porta non individua un servizio, ma 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);     } }
```

MULTICAST: L'API JAVA

- Ogni socket multicast ha una proprietà, la reuse socket, che se settata a true da la possibilità di associare più socket alla stessa porta
- Se questa proprietà è settata a true
  - se attivo due istanze di provemulticast sullo stesso host,e la reuse socket è settat a true non viene sollevata una BindException
  - l'eccezione verrebbe invece sollevata se si utilizzasse un DatagramSocket
- Nelle prime versioni di JAVA la proprietà era settata per default a true
- Nelle ultime versioni è possibile impostarne il valore

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

MULTICAST SNIFFER

Scriver un programma che dopo aver ricevuto 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

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

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 $\Rightarrow$ 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 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);
```

ESERCIZIO 1: DATE SERVER

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.

ESERCIZIO 2: MINITALK CON UDP

Si realizzino un Server e un Client UDP che realizzano un semplice Instant Messanger: ogni linea scritta nella shell del Server viene copiata nella shell del Client e viceversa. La trasmissione delle linee inizia dopo una semplice fase di handshaking, descritta di seguito.

- il Server viene lanciato da shell, fornendo il numero di porta su cui ricevere i pacchetti UDP.
- Il Client viene lanciato da un'altra shell (eventualmente su un altro computer), fornendo tre dati: host e porta del Server, e numero di porta locale su cui ricevere pacchetti UDP.
- Il Client manda una richiesta di connessione al Server, indicando nel messaggio la propria porta. Se non riceve un messaggio di ack entro 3 secondi, riprova a mandare la richiesta altre 5 volte, poi termina. Se invece riceve l'ack, inizia la trasmissione delle linee scritte nella shell locale e la ricezione e stampa delle linee scritte nella shell del Server.

ESERCIZIO 2: MINITALK CON UDP

- Il Client decide quando disconnettersi in base a una condizione a vostra scelta (per esempio, allo scadere di un timeout, oppure se la linea da mandare è vuota, oppure se la stringa è "CLOSE",…). Per disconnettersi, il Client manda una richiesta di disconnessione al Server e aspetta un ack, quindi termina l'esecuzione.
- Quando il Server riceve una richiesta di disconnessione interrome la trasmissione delle linee, manda un messaggio di ack, e si rimette in attesa di una richiesta di connessione.
- Il Client e il Server devono scambiarsi unicamente oggetti della classe TalkMsg, il cui codice è pubblicato sulla pagina del corso. Si devono usare i metodi della classe per crearne istanze e per ispezionare i messaggi arrivati.

SERVER PROGRAMMING

- Server Web, Mail Server, File Server, Database Server sono caratterizzati dalla seguente struttura:

```
ServerSocket socket = new ServerSocket(80);
    while (true)
    { Socket connection = socket.accept( );
          handleRequest(connection);}
```

- La gestione di ogni servizio (handleRequest) è indipendente dalla gestione degli altri
- Esempio: la gestione di un messaggio inviato ad un mail server non dipende da quella degli altri messaggi elaborati contemporaneamente
- E' naturale associare un thread diverso alla gestione di ogni richiesta di servizio.

SERVER PROGRAMMING

```
importjava.net.*;import java.util.concurrent.*;import java.io.*;
public class server_pool {
private static final int NTHREADS = 100;
private static final Executor
   exec=Executors.newFixedThreadPool(NTHREADS);
    public static void main(String [ ] args) throws IOException{
    ServerSocket socket = new ServerSocket(80);
    while (true) {
        final Socket connection = socket.accept();
        Runnable task = new Runnable( )
       {public void run( ){handleRequest(connection);}}};
       exec.execute(task);   }    }    }
```

THREAD POOL: POLITICHE DI SATURAZIONE

- Politica di saturazione:
  - utilizzata nel caso si utilizzino thread pool con code di dimensione limitata
  - indica le azioni che devono essere effettuate nel caso in cui il pool è saturo, cioè la coda è piena ed i threads del pool sono tutti attivi
- Strategia di default: abort il task viene rifiutato e viene sollevata una RejectedExecutionException
- E' possibile definire una politica 'ad hoc' mediante un Rejected Execution Handler
- JAVA mette a disposizione diversi Rejected Execution Handler, ognuno dei quali implementa una diversa politica di saturazione
- Selezione della politica: setRejectedExecutionHandler

THREAD POOL:POLITICHE DI SATURAZIONE

Politiche di saturazione: quando viene sottomesso un task T e la coda è piena, possono essere adottate le seguenti politiche

- abort T viene scartato e viene sollevata un'eccezione
- discard rifiuta T, ma non solleva alcun tipo di eccezione
- discard oldest scarta il primo task della coda (quello che avrebbe dovuto essere eseguito successivamente) e inserisce T in coda
- caller-runs
  - non scarta il task e non solleva eccezioni.
  - cerca di rallentare (throttling, to throttle=strozzare) il flusso dei tasks restituendo alcuni task al chiamante per l'esecuzione. Il task non viene eseguito in un thread del pool, ma nel thread che ha invocato la execute

THREAD POOL: POLITICHE DI SATURAZIONE

```
import java.util.concurrent.*;
public class prova{
public static void main (String args[ ])
{ThreadPoolExecutor executor=new
   ThreadPoolExecutor(10,11,0L,TimeUnit.MILLISECONDS,
              new LinkedBlockingQueue<Runnable>(100));
  executor.setRejectedExecutionHandler    ( new
                ThreadPoolExecutor.CallerRunsPolicy());}
}
```

THREAD POOL:POLITICHE DI SATURAZIONE

- Consideriamo un Web Server
  - riceve richieste di servizio da parte dei clients
  - ogni richiesta corrisponde ad un diverso task
- Supponiamo che il Web Server definisca un thread pool che utilizzi una coda a dimensione limitata ed una politica di saturazione caller-runs.
- Se tutti i threads sono occupati e la coda è piena, il task successivo (la successiva richiesta di servizio) viene eseguita nel thread che ha invocato la execute (il main thread del server)
- Poiché l'esecuzione del servizio richiesto richiederà un intervallo di tempo ΔT, il main thread non sottometterà ulteriori taks per l'esecuzione al pool, durante ΔT
- Alcuni thread del pool possono terminare l'esecuzione del task assegnato durante ΔT, rendendosi disponibili a nuove sottomissioni

THREAD POOL: POLITICHE DI SATURAZIONE

- Inoltre, il main thread, impiegato nell'esecuzione del task rifiutato dal pool, non accetta ulteriori connessioni (non esegue l'accept sul ServerSocket) durante l'intervallo di tempo ΔT
- Le richieste di servizio vengono quindi accodate a livello TCP
- Se l'overload persiste, sarà il livello TCP che eventualmente inizierà a scartare richieste, quando la sua coda sarà, a sua volta, satura
- Conclusione: se il server è sovraccarico, si può implementare una graceful degradation, spostando gradualmente il sovraccarico
  - dai thread del pool alla coda associata al pool,
  - dalla coda alla applicazione,
  - dall'applicazione al livello TCP
  - Il livello TCP è in grado di bloccare il client, mediante il meccanismo di controllo del flusso

POLITICHE DI SATURAZIONE: ESEMPI

```
public void process(    )
{ ThreadPoolExecutor exec =  new ThreadPoolExecutor(5, 5,
        0L, TimeUnit.MILLISECONDS, new SynchronousQueue(0));
  exec.setRejectedExecutionHandler(new
                        ThreadPoolExecutor.CallerRunsPolicy());
   while(true)
   {Object work = acquireWork();
    if(work != null)
          exec.execute(new WorkerThread(work)); }}
```

- supponiamo che 6 nuovi tasks vengano sottomessi a process, i primi 5 vengono distribuiti ai worker threads, il sesto viene rifiutato ed eseguito da process
- se tutti i tasks richiedono approssimativamente lo stesso tempo, tutti i threads terminano l'esecuzione dei tasks approssimativamente nello stesso tempo
- se il sesto tasks richiede più tempo, il thread del pool rimangono inattivi

POLITICHE DI SATURAZIONE: ESEMPI

```
public void process()
{ ThreadPoolExecutor exec =  new ThreadPoolExecutor(5, 5,
        0L, TimeUnit.MILLISECONDS, new LinkedBlockingQueue(5));
  exec.setRejectedExecutionHandler(new
            ThreadPoolExecutor.CallerRunsPolicy());
   while(true)
   {Object work = acquireWork();
      if(work != null)
      {exec.execute(new WorkerThread(work));
      }}}
```

- Quando un task viene rifiutato, esistono alcuni task in coda
- Anche se process impiega molto tempo nella esecuzione del task rifiutato, esiste del lavoro extra che il therad pool distribuisce ai workers, mentre process elabora il task rifiutato

PROGRAMMAZIONE CONCORRENTE: IL PROBLEMA DEI 5 FILOSOFI

- cinque filosofi siedono ad una tavola rotonda con un piatto di spaghetti davanti, una forchetta (o bacchette cinesi, a seconda della versione)a destra e una forchetta a sinistra

- la vita di un filosofo alterna periodi in cui mangia a quelli in cui pensa

- ciascun filosofo ha bisogno di due forchette per mangiare, ma che le forchette vengano prese una per volta. Dopo essere riuscito a prendere due forchette il filosofo mangia per un pò, poi lascia le forchette e ricomincia a pensare

IL PROBLEMA DEI 5 FILOSOFI

- Problema proposto da E.Dijkstra nel '65: sviluppo di un algoritmo che impedisca lo stallo (deadlock) o la morte d'inedia (starvation).

- Il deadlock può verificarsi se ciascuno dei filosofi tiene in mano una forchetta senza mai riuscire a prendere l'altra. Il filosofo F1 aspetta di prendere la forchetta che ha in mano il filosofo F2, che aspetta la forchetta che ha in mano il filosofo F3, e così via in un circolo vizioso.

IL PROBLEMA DEI 5 FILOSOFI

- problema proposto da E.Dijkstra nel '65 come problema di sincronizzazione tra più attività concorrenti

- Starvation si verifica quando un processo non riesce mai ad acquisire le risorse di cui ha bisogno

- la situazione di starvation può verificarsi indipendentemente dal deadlock se uno dei filosofi non riesce mai a prendere entrambe le forchette.

FILOSOFI A PRANZO: LE BACCHETTE

```
public class bacchetta {
        private boolean taken;
        private int bacid;
        public bacchetta (int bacid)
            {this.bacid=bacid;
             this.taken = false;    }
```

FILOSOFI A PRANZO: LE BACCHETTE

```
public synchronized void take(int filid) throws
                                        InterruptedException
        { while (taken)
            wait();
            taken=true;
            System.out.println(filid+"ho preso la
                                racchetta"+bacid);}

public synchronized void drop(int filid) {
            taken = false;
            System.out.println(filid+"ho lasciato la
                                racchetta"+bacid);
            notifyAll();  } }
```

FILOSOFI A PRANZO: IL FILOSOFO

```
import java.util.*;
public class filosofo implements Runnable {
        private bacchetta left;
        private bacchetta right;
        private int filid;
        public filosofo (bacchetta left, bacchetta right,
                         int filid)
              {this.filid=filid;
               this.left=left;
               this.right=right;
               }
```

FILOSOFI A PRANZO: IL FILOSOFO

```
public void run( )   {
       try {  for (int i=0; i<2; i++){
              System.out.println (filid+""+"thinking");
              Thread.sleep (500);
              right.take(filid);
              left.take(filid);
              System.out.println (filid+""+"Mangio");
              Thread.sleep(1000);
              right.drop(filid);
              left.drop(filid);
          }}catch (InterruptedException e){} }

                    }
```

FILOSOFI A PRANZO: INIZIALIZZAZIONE

```
import java.util.concurrent.*;
public class filosoficondeadlock {
public static void main (String [ ] args) throws Exception
{int size =5;
if (args.length >1)  size = Integer.parseInt(args[1]);
ExecutorService exec = Executors.newFixedThreadPool(size);
bacchetta [  ] vectbacchetta = new bacchetta[size];
for (int i=0; i < size; i++)   vectbacchetta[i]= new
  bacchetta(i);
for (int i= 0; i< size; i++)
    {exec.execute(new
  filosofo(vectbacchetta[i],vectbacchetta[(i+1)%size],i));} }
  }
```

FILOSOFI A PRANZO: ESEMPIO DI ESECUZIONE

1thinking

0thinking

2thinking

4thinking

3thinking

1ho preso la racchetta2

1ho preso la racchetta1

1Mangio

2ho preso la racchetta3

4ho preso la racchetta0

4ho preso la racchetta4

4Mangio

4ho lasciato la racchetta0

4ho lasciato la racchetta4

4thinking

FILOSOFI A PRANZO: ESEMPIO DI ESECUZIONE

3ho preso la racchetta4

1ho lasciato la racchetta2

1ho lasciato la racchetta1

2ho preso la racchetta2

2Mangio

2ho preso la racchetta2

2Mangio

```
0ho preso la racchetta1
0ho preso la racchetta0
0Mangio
1thinking
2ho lasciato la racchetta3
0ho lasciato la racchetta1
0ho lasciato la racchetta0
0thinking.................
```

In questo caso il deadlock non si verifica...

FILOSOFI A PRANZO: ESECUZIONE CON DEADLOCK

Per provocare il deadlock modificare il codice in modo che un thread si sospenda dopo che ha preso la bacchetta di destra

```
public void run(){
try { for (int i=0; i<2; i++){
    System.out.println (filid+""+"thinking");
    Thread.sleep(500);
    right.take(filid);
    Thread.sleep(2500);
    left.take(filid);
    System.out.println (filid+"  "+"Mangio");
    Thread.sleep(1000);
    right.drop(filid);  left.drop(filid);  }}catch
      (InterruptedException e){ } } }
```

FILOSOFI A PRANZO: ESECUZIONE CON DEADLOCK

1thinking

2thinking

0thinking

3thinking

4thinking

1ho preso la racchetta2

2ho preso la racchetta3

0ho preso la racchetta1

4ho preso la racchetta0

3ho preso la racchetta4

FILOSOFI A PRANZO

Compito per casa:

Come evitare il deadlock??