Lezione n.6
LPR- Informatica Applicata
Multicast

20/3/2006

Laura Ricci

Riassunto della presentazione

- Comunicazione tra gruppi di processi
- Gruppi di multicast
- Multicast affidabile
- Java Multicast API

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
- *DNS:* 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 G*
- 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

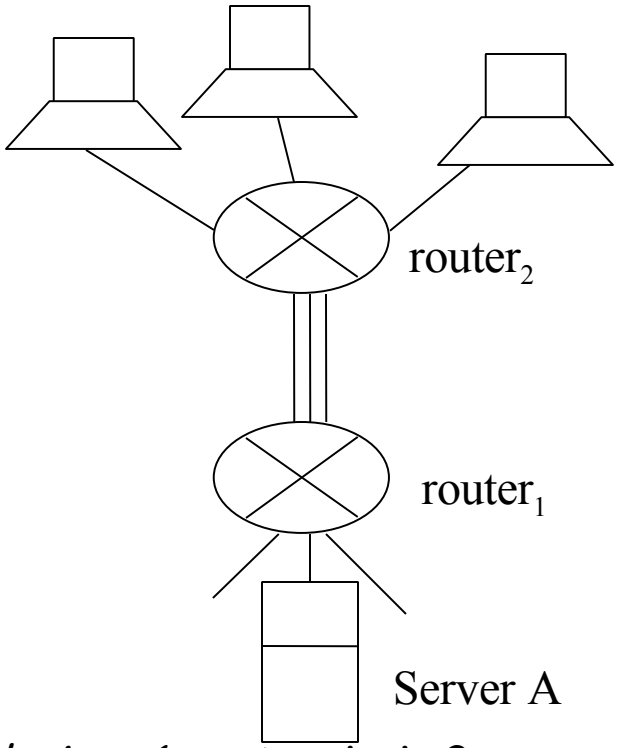
COMUNICAZIONE TRA GRUPPI DI PROCESSI: IMPLEMENTAZIONE

L'implementazione del multicast richiede:

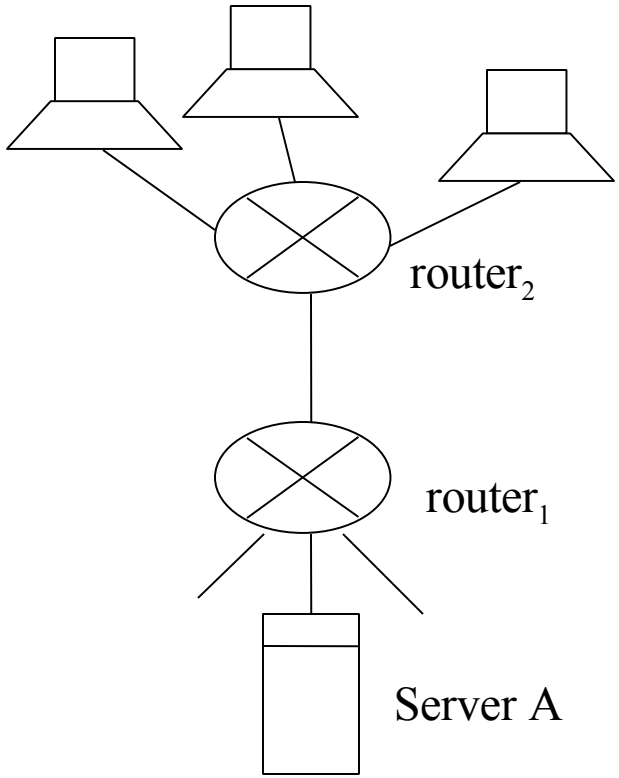
- uno schema di indirizzamento dei gruppi
- un supporto che registri la corrispondenza tra un gruppo ed i partecipanti
- un'implementazione che ottimizzi l'uso della rete nel caso di invio di pacchetti ad un gruppo di multicast

MULTICAST: IMPLEMENTAZIONE

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



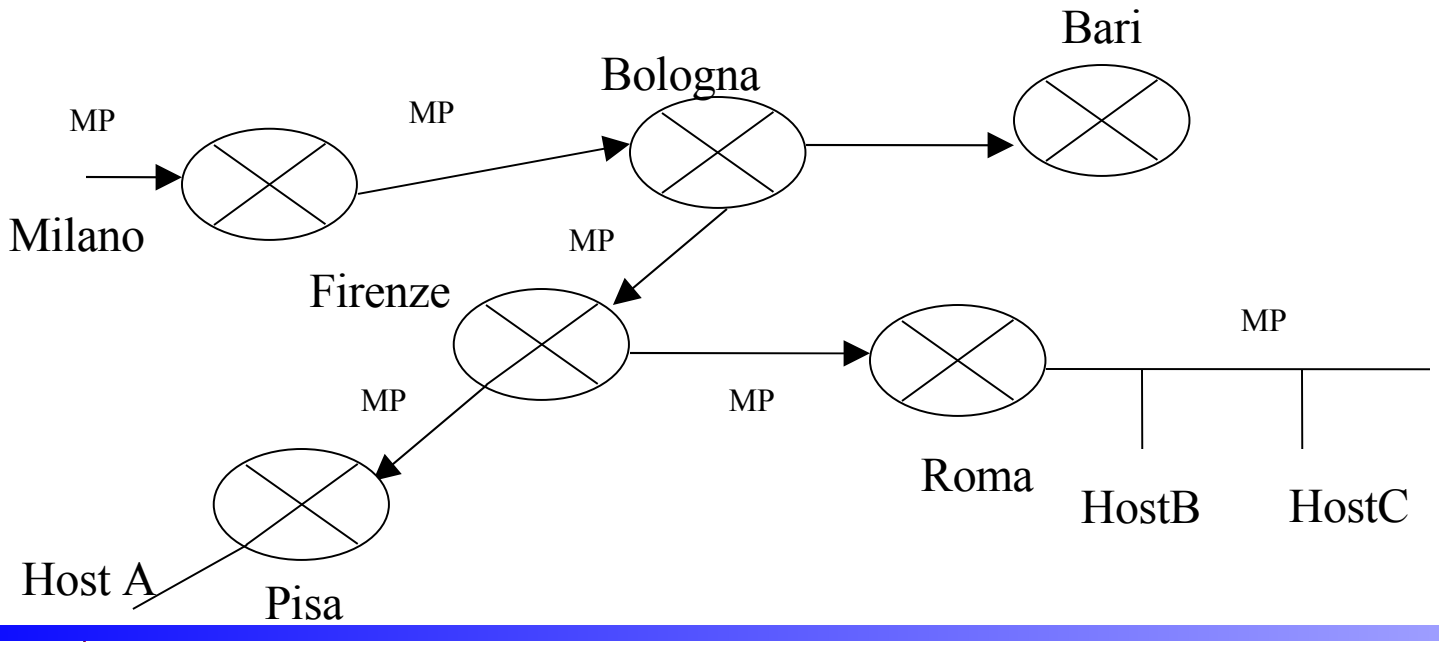
Soluzione 1: $router_1$ invia 3 messaggi distinti con collegamenti di tipo unicast



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

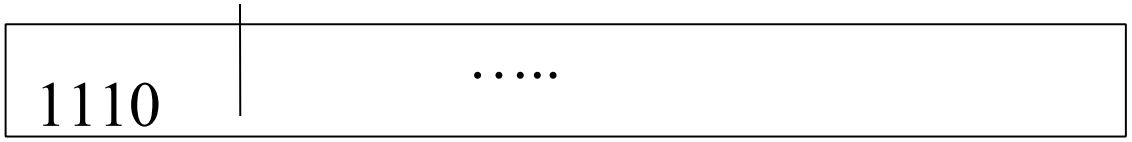
MULTICAST: IMPLEMENTAZIONE

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 DEI GRUPPI DI MULTICAST

- Indirizzo di un gruppo di *multicast*: indirizzo IP di classe D
- 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

- Gli indirizzi di multicast possono essere assegnati

  - in *modo permenante* (assegnati da IANA).

  - lista completa degli indirizzi di multicast riservati:
    http://www.iana.org/assignments/multicast-addresses

  - in modo *temporaneo.*

INDIVIDUAZIONE GRUPPI DI MULTICAST

- gli indirizzi di multicast appartenenti all'intervallo

  224.0.0.0 - 224.0.0.255

- consentono di limitare la diffusione di un pacchetto alla sottorete locale.

- alcuni di questi indirizzi sono riservati

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
PIM-ROUTERS.MCAST.NET 224.0.0.12 tutti i PIM (Protocol Independent Multicasting) routers della rete locale

……

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:
  sono necessarie *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.

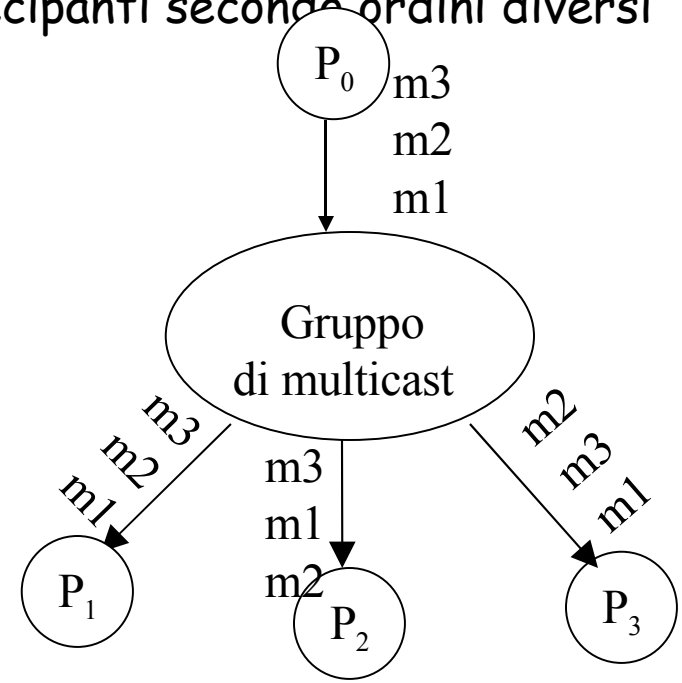
unico servizio offerto dalla multicast JAVA API standard (esistono package JAVA non standard che offrono qualche livello di affidabilità)

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

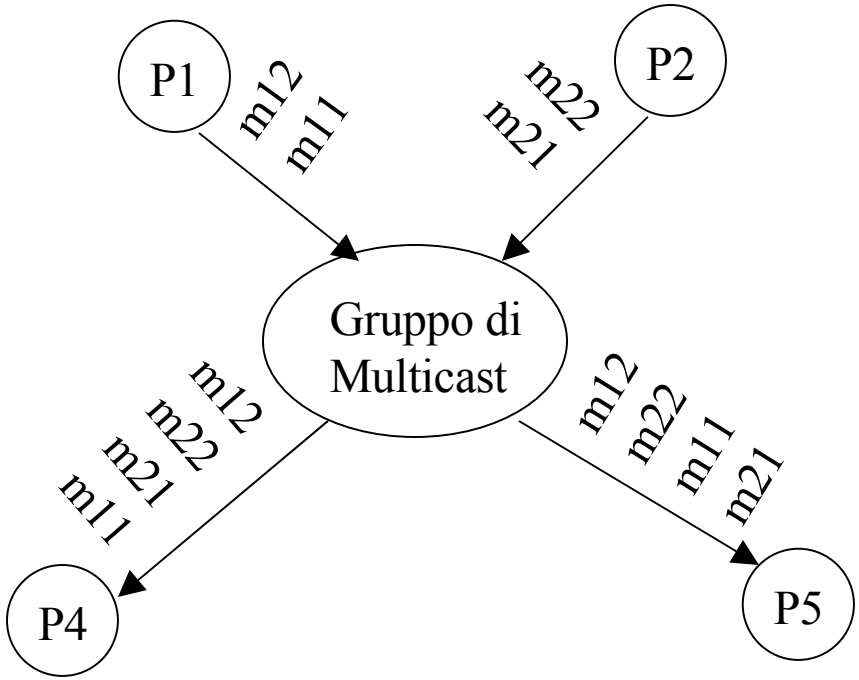
UNORDERED MULTICASTING

Unordered Multicasting: *nessuna garanzia* circa l'ordine con cui i messaggi vengono recapitati ai destinatari. Esempio: p_0 invia, *nell'ordine,* m_1,m_2,m_3 al gruppo di multicast. I messaggi vengono consegnati ai partecipanti secondo ordini diversi



FIFO MULTICASTING

Se un processo P invia, *nell'ordine*, i messaggi m_i ed m_j ad un gruppo di multicast, allora tutti i processi del gruppo ricevono i messaggi secondo quell'ordine. Non viene garantito nessun ordinamento per messaggi spediti da processi diversi



CAUSAL ORDER MULTICAST

Se la ricezione di un messaggio m_i precede la spedizione del messaggio m_j, allora m_i viene ricevuto da tutti i processi del gruppo prima del messaggio m_j

mi $\rightarrow$ mj mi ed mj sono in relazione *di ordinamento causale*

Causal-order multicast : mantiene l'ordinamento causale dei messaggi. Tutti i processi ricevono i messaggi spediti al gruppo di multicast secondo l'ordinamento causale

Esempio: P_1,P_2,P_3 appartengono allo stesso gruppo di multicast. P_1 invia un messaggio m_i a P_2. P_2, dopo aver ricevuto il messaggio di P_1, spedisce un messaggio m_j sul gruppo di multicast. Tutti gli appartenenti al gruppo di multicast ricevono mi prima di mj

ATOMIC ORDER MULTICAST

Atomic order multicast: garantisce che tutti i messaggi spediti ad un gruppo di multicast vengano consegnati ad ogni partecipante al gruppo secondo lo stesso ordine.

Esempio:

P_1 invia m_1, P_2 invia m_2, P_3 invia m_3

l'ordine con cui i partecipanti riceveranno i messaggi può essere uno qualsiasi tra

m1-m2-m3, m1-m3-m2, m2-m1-m3, m2-m3-m1, m3-m1-m2, m3-m2-m1

ma l'ordine è lo stesso per ogni partecipante

MULTICAST: L'API JAVA

public class *MulticastSocket* ***extends*** *DatagramSocket*

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

```
try {
      MulticastSocket ms =new MulticastSocket( );
      .......
    }
catch (SocketException ex) {System.out.prinln (...) }
```

è possibile specificare la porta a cui collegare il socket.

MULTICAST: L'API JAVA

public void *joinGroup* (***InetAddress*** address) ***throws IOException***

```
try{
 MulticastSocket ms = new MulticastSocket (4000);
 InetAddress ia = InetAddress.getByName
      ("experiment.mcast.net");
 ms.joinGroup(ia);
 .....
  } catch (IOException ex) { System.out.println(ex) }
```

- 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

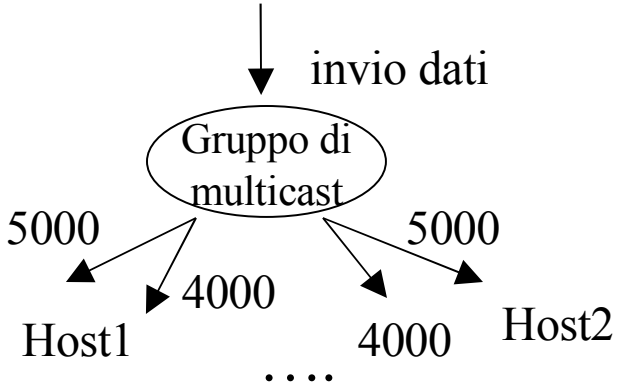
Uso delle porte per multicast sockets:

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

due processi, attivati sullo stesso host da due shell diverse possono collegarsi allo stesso gruppo di multicast sulla *stessa porta*

⇒

una porta non individua una *specifica applicazione* (*processo*)

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

se attivo due istanze di provamulticast da *shell diverse* della *stessa macchina*, non viene sollevata *BindException* (che viene invece sollevata se si sostituisce il MulticastSocket con un DatagramSocket)

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*

```
try   {
 InetAddress ia=
            InetAddress.getByName("experimental.mcast.net");
 byte [] data =…;.
 int port 4000;
 DatagramPacket dp = new DatagramPacket(data, data.length, ia ,
                                                    port);
 MulticastSocket ms = new MulticastSocket( );
 ms.send(dp);
 }
catch(IOException ex){ System.out.println(ex) }
```

MULTICAST: TIME TO LIVE

- *IP Multicast Scoping: m*eccanismo 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 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:

  - Associo il ttl al multicast socket

```
MulticastSocket s = new MulticastSocket( );
s.setTimetoLive(1);
```

  - Specifico il ttl nella send

```
DatagramPacket dp = new DatagramPacket(data, data.length);
MulticastSocket ms= new MulticastSocket( );
ms.send(dp,64)
```

MULTICAST: ESERCIZIO 1

Definire un Server *TimeServer*, che invia su un gruppo di multicast *dategroup*, ad intervalli regolari,la *data e l'ora*. Data ed ora possono essere ottenute mediante il metodo JAVA *calendar( )*, 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.

MULTICAST: ESERCIZIO 2

Si consideri una applicazione composta da *m* servers $Magazzino_1$,.... .,$Magazzino_m$ ed n clients $Client_1$, ...,$Client_n$. Ognuno dei client ed dei server è allocato all'interno di uno *spazio virtuale bidimensionale* e può essere individuato dalle sue *coordinate cartesiane* all'inteno di questo spazio. Ogni Magazzino possiede lo stesso insieme di articoli, ognuno caratterizzato da un codice e dalla quantità presente in quel magazzino (scorta). Le informazioni riguardanti gli articoli vengono caricate da un file.

Ogni magazzino *invia periodicamente* le sue coordinate ad un gruppo di multicast M a cui ogni client si collega all'inizio della propria esecuzione. Quando un client riceve una informazione da M, la scarta se l'ha ricevuta precedentemente, altrimenti la memorizza in una struttura dati opportuna.

MULTICAST: ESERCIZIO 2 (CONTINUA)

Ogni client riceve dall'utente, in modo interattivo, il *codice* e la *quantità* di un prodotto e richiede tale prodotto al magazzino *più vicino*, tramite una connessione UDP. Il Magazzino invia al client un *ack*, nel caso possieda tale prodotto in quantità sufficiente, altrimenti invia un *nack*. Il client visualizza la risposta e, nel caso di nack, provvede ad inviare un' ulteriore richiesta al Magazzino piu' vicino, *scelto tra i rimanenti.*

Il procedimento termina quando il Client riceve una risposta positiva da un magazzino, oppure quando sono stati consultati tutti i magazzini.