Laboratorio di Reti – A (matricole pari)

Autunno 2021,
instructor: Laura Ricci

laura.ricci@unipi.it

Lezione 10

UDP CHANNELS MULTICAST, URL

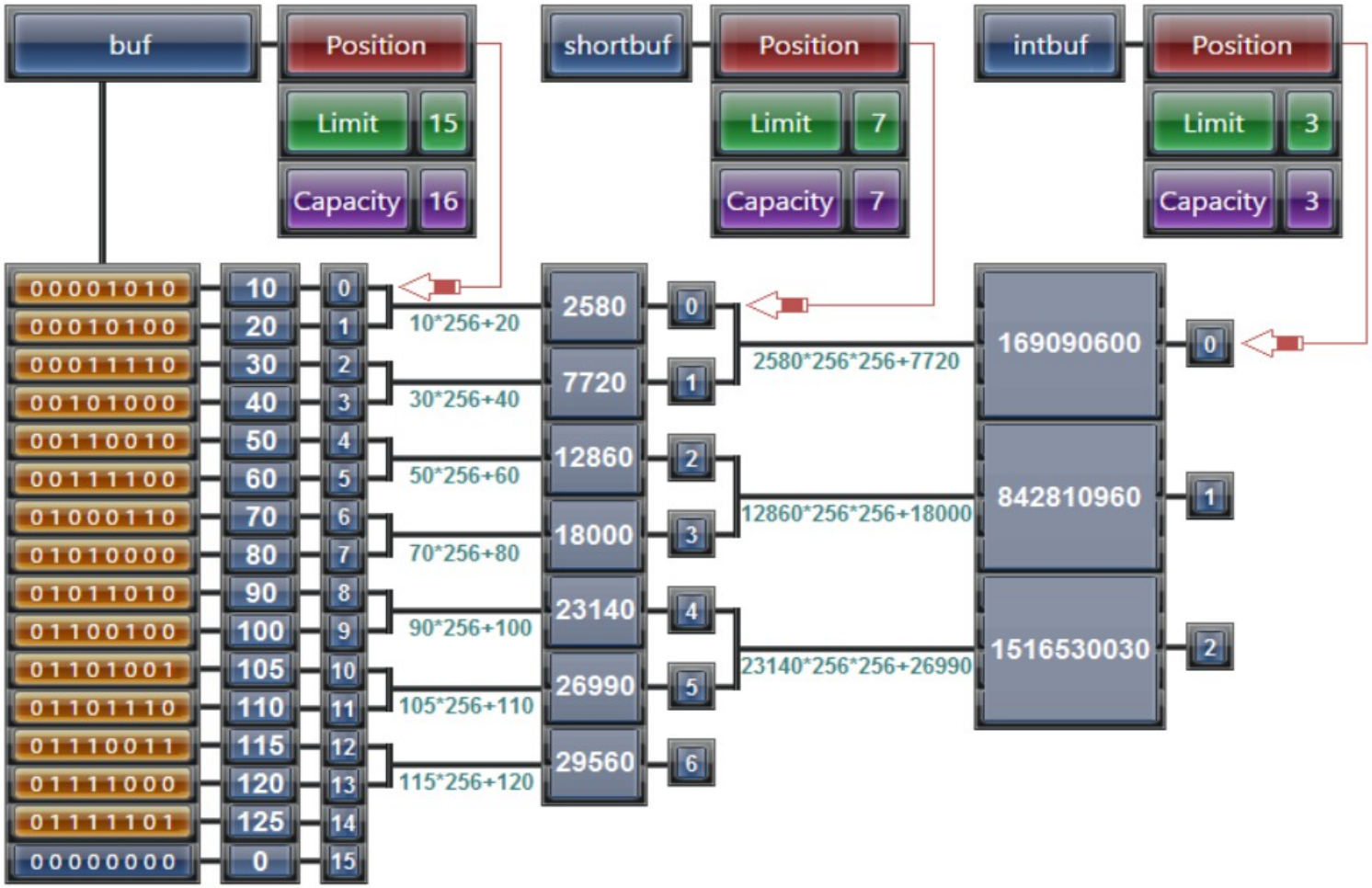
23/11/2021

BYTE-BUFFER MAPPING TO PRIMITIVE TYPES

- come elaborare i dati che si trovano in un `ByteBuffer`?
- possibilità di utilizzare view buffer
  - creazione di "viste" che mappano i byte del buffer in altri tipi di dato primitivi
  - viene creato un nuovo buffer
    - capacità = numero di elementi significativi del buffer originario (restituito dal metodo `remaining`) diviso il numero di byte necessari per la rappresentazione del tipo di dato primitivo
    - ogni byte rimanente, alla fine del buffer, non risulta visibile nel view buffer
    - `position`, `limit` e `capacity` sono aggiornati nel nuovo view buffer

BYTE-BUFFER MAPPING TO PRIMITIVE TYPES

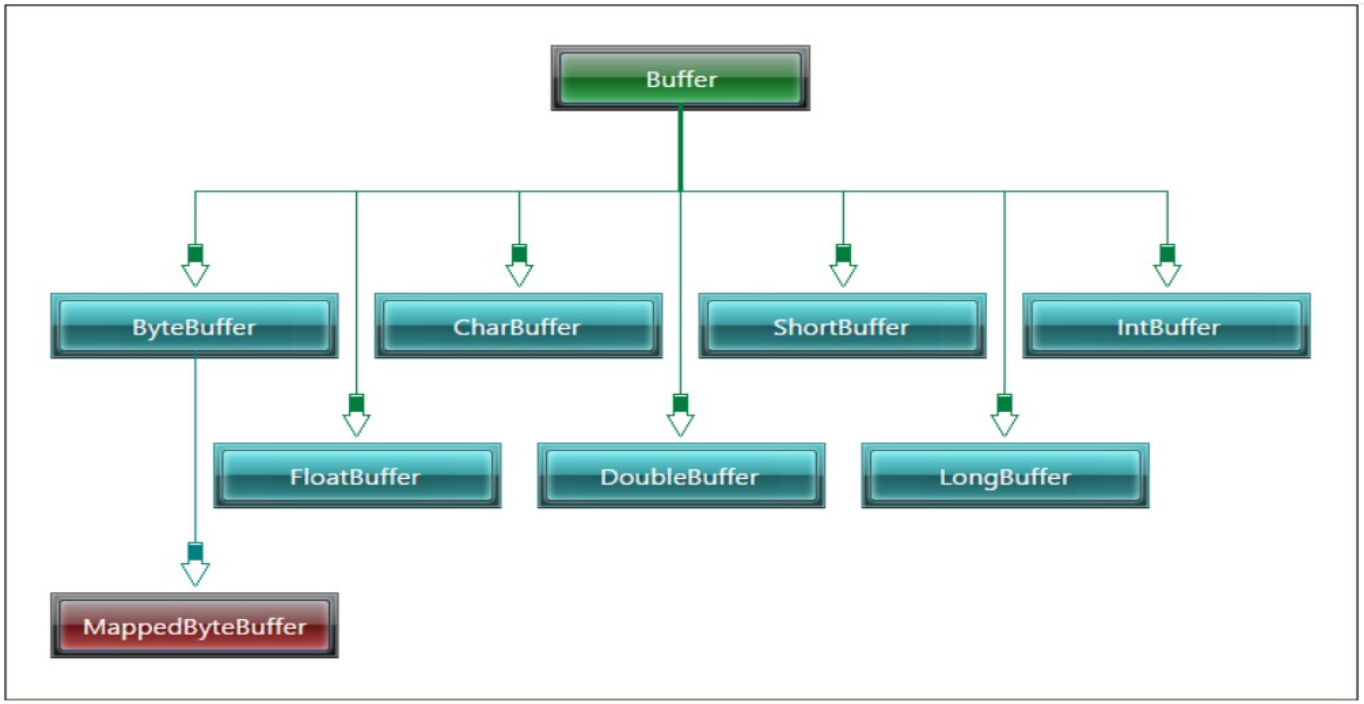
```
ByteBuffer buf = ByteBuffer.allocate(16);
buf.put(new byte[]{10,20,30,40,50,60,70,80,90,100,105,110,115,120,125});
buf.flip();
ShortBuffer shortbuf = buf.asShortBuffer();
IntBuffer intbuf = buf.asIntBuffer();
```



BYTE-BUFFER MAPPING TO PRIMITIVE TYPES

- diversi utilizzi
  - caricare dati che non sono di tipo byte in un `ByteBuffer` prima di inviarlo al dispositivo periferico
  - accedere ai dati letti da un file come valori diversi da byte grezzi

```
ByteBuffer buffer = ByteBuffer.allocate(1024);
IntBuffer intBuf = buf.asIntBuffer();
```



DATAGRAMCHANNEL

- la classe `DatagramChannel` supporta operazioni UDP non bloccanti
  - i metodi ritornano il controllo al chiamante se la rete non è immediatamente "pronta" a ricevere o a spedire dati
    - poichè UDP è intrinsecamenet "più asincrono" di TCP, l'effetto è minore
  - i `DatagramChannel` possono essere registrati su un `Selector`
    - oossibile monitorare lo stato di più socket UDP ed invocare/ricevere dati da quelli pronti a ricevere/spedire
- `Java 7+`: non necessario usare `DatagramSocket` o `DatagramPacket`
  - recive/read send/write byte buffer come si fa per `SocketChannel`

UDP CHANNELS: RECEIVE

```
DatagramChannel channel = DatagramChannel.open();
channel.socket().bind(new InetSocketAddress(9999));
ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
// Prepare buffer for reading
channel.receive(buf);
...
```

- apertura del `DatagramChannel`
- receive si blocca fino a che non è stato ricevuto un `DatagramPacket`
- `buf.clear( )`
  - svuota il buffer: re-inizializza `Position` e `Limit`
- come la receive di `DatagramSocket`: copia il pacchetto ricevuto nel buffer. Se il pacchetto ricevuto contiene più dati di quanti possano essere contenuti nel `Buffer`, scarta i dati rimanenti.

UDP CHANNELS: SEND

```
DatagramChannel channel = DatagramChannel.open();
channel.socket().bind(new InetSocketAddress(9999));
String newData = "New String to write to file..."
ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
buf.put(newData.getBytes());
buf.flip();
int bytesSent = channel.send(buf, new InetSocketAddress("www.google.it", 80));
```

- apertura del `DatagramChannel`
- `buf.flip()` prepara il buffer alla lettura: quello che ha scritto l'applicazione viene letto dal supporto
- se il canale è in modalità non bloccante e c'è sufficiente spazio nel buffer del SO o se è in modalità bloccante e si è liberato sufficiente spazio nel buffer del SO, il contenuto del `ByteBuffer` viene inviato in un unico pacchetto
- restituisce il numero di bytes inviati

SEND/RECEIVE NON BLOCCANTI

```
ByteBuffer buffer = ByteBuffer.allocate(8192);
DatagramChannel channel= DatagramChannel.open();
channel.configureBlocking(false);
channel.socket().bind(new InetSocketAddress(9999));
SocketAddress address = new InetSocketAddress("localhost",7);
buffer.put(…);
buffer.flip();
while (channel.send(buffer, address) == 0);
        //do something useful …
buffer.clear();
while ((address = channel.receive(buffer))==null);
    //do something useful …
```

- se la send restituisce 0, il buffer del SO non ha sufficiente spazio
- se la receive restituisce null, non è stato ricevuto alcun Datagram
- il programma può eseguire altre operazioni, nell'attesa che il canale sia disponible per l'operazione

DATAGRAMCHANNEL: MULTIPLEXING

- possibilità di registrare il canale con un selettore
- invocazione del metodo `Selector.select` per testare la "`readibility`" o "`writability`" del canale.

Operation	Meaning
OP_READ	Data is present in the socket receive-buffer or an exception is pending.
OP_WRITE	Space exists in the socket send-buffer or an exception is pending. In UDP, OP_WRITE is almost always ready except for the moments during which space is unavailable in the socket send-buffer. It is best only to register for OP_WRITE once this buffer-full condition has been detected, i.e. when a channel write returns less than the requested write length, and to deregister for OP_WRITE once it has cleared, *i.e.* a channel write has fully succeeded.

NON BLOCKING MULTIPLEXED UDP ECHO SERVER

```
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
public class UDPEchoServerWithChannels {
  public final static int PORT = 7;
  public final static int MAX_PACKET_SIZE = 65507;
  public static void main(String[] args) {
    try {
        DatagramChannel channel = DatagramChannel.open();
        DatagramSocket socket = channel.socket();
        SocketAddress address = new InetSocketAddress(PORT);
        socket.bind(address);
        System.out.println("I am ready");
        ByteBuffer buffer = ByteBuffer.allocateDirect(MAX_PACKET_SIZE);
```

NON BLOCKING MULTIPLEXED UDP ECHO SERVER

```
while (true) {
          SocketAddress client = channel.receive(buffer);
          buffer.flip();
          channel.send(buffer, client);
          buffer.clear();
         }
} catch (IOException ex) {
  System.err.println(ex); } } }
```

- nota: il servizio è molto semplice, per cui ha senso implementare un server sequenziale.

UDP ECHO CLIENT WITH CHANNELS

- invia al server i numeri interi da 0 a 99
- stampa i valori restituiti
- controlla se qualche pacchetto è perso
- vantaggio di usare i channel in modalità non bloccante
  - invio e ricezione non bloccante
  - uso di un selettore: invio o ricezione sui canali "pronti"
  - il programma non si ferma, per ogni valore inviato, a ricevere l'eco
  - può inviare più valori prima di ricevere una risposta

UDP ECHO CLIENT WITH CHANNELS

```
import java.io.*; import java.net.*; import java.nio.*;
import java.nio.channels.*; import java.util.*;
public class UDPEchoClientWithChannels {
  public  final static int PORT = 7; private final static int LIMIT = 100;
  public static void main(String[] args) {
    SocketAddress remote;
    try {
      remote = new InetSocketAddress(args[0], PORT);
    } catch (RuntimeException ex) {
          System.err.println("Usage: java UDPEchoClientWithChannels host"); return; }
     try (DatagramChannel channel = DatagramChannel.open()) {
          channel.configureBlocking(false);
          channel.connect(remote);
          Selector selector = Selector.open();
          channel.register(selector, SelectionKey.OP_READ | SelectionKey.OP_WRITE);
```

Creazione del socket e registrazione del canale al selettore

UDP ECHO CLIENT WITH CHANNELS

```
ByteBuffer buffer = ByteBuffer.allocate(4);
int n = 0;
int numbersRead = 0;
while (true) {
        if (numbersRead == LIMIT) break;
        // wait one minute for a connection
        selector.select(60000);
        Set<SelectionKey> readyKeys = selector.selectedKeys();
        if (readyKeys.isEmpty() && n == LIMIT) {
          // All packets have been written and it doesn't look like any
          // more are will arrive from the network
          break;
        }
        else {
          Iterator<SelectionKey> iterator = readyKeys.iterator();
          while (iterator.hasNext()) {
            SelectionKey key = (SelectionKey) iterator.next();
            iterator.remove();
```

Loop per il controllo delle chiavi del selettore

UDP ECHO CLIENT WITH CHANNELS

```
if (key.isReadable()) {
    buffer.clear();
    channel.receive(buffer);
    buffer.flip();
    int echo = buffer.getInt();
    System.out.println("Read: " + echo);
    numbersRead++;
}
if (key.isWritable()) {
    buffer.clear();
    buffer.putInt(n);
    buffer.flip();
    channel.send(buffer, remote);
    System.out.println("Wrote: " + n);
    n++;
```

Conta e stampa il valore ricevuto

invia, stampa e conta il valore spedito

UDP ECHO CLIENT WITH CHANNELS

```
            n++;
              if (n == LIMIT) {
                // All packets have been written; switch to read-only mode
                key.interestOps(SelectionKey.OP_READ);
              } } } } }
```

stampa delle statistiche

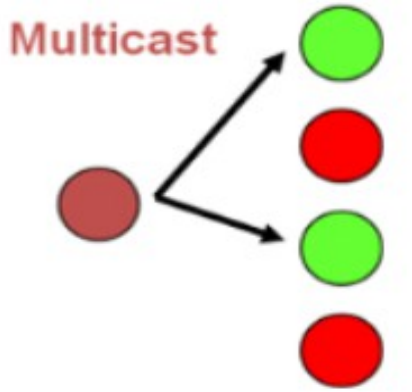
```
  System.out.println("Echoed " + numbersRead + " out of " + LIMIT + " sent");
  System.out.println("Success rate: " + 100.0 * numbersRead / LIMIT + "%");
} catch (IOException ex) {
  System.err.println(ex);
   } } }
```

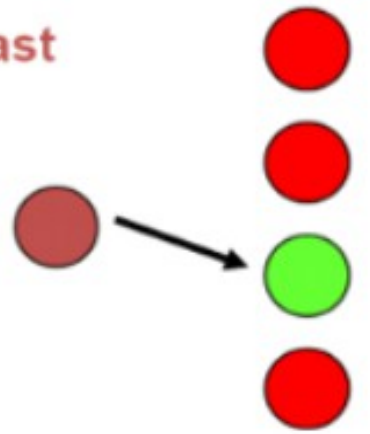
```
$ Java UDPEchoClientWithChannels
Wrote: 0
Read: 0
Wrote: 1
Wrote: 2
Read: 1
Wrote: 3
….....
Read: 98
Read: 99
Echoed 100 out of 100 sent
Success rate: 100.0%
```

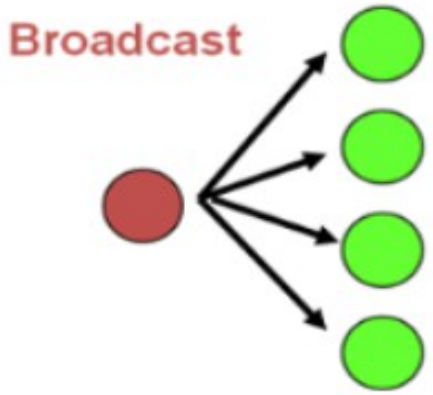
MULTICASTING

- 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

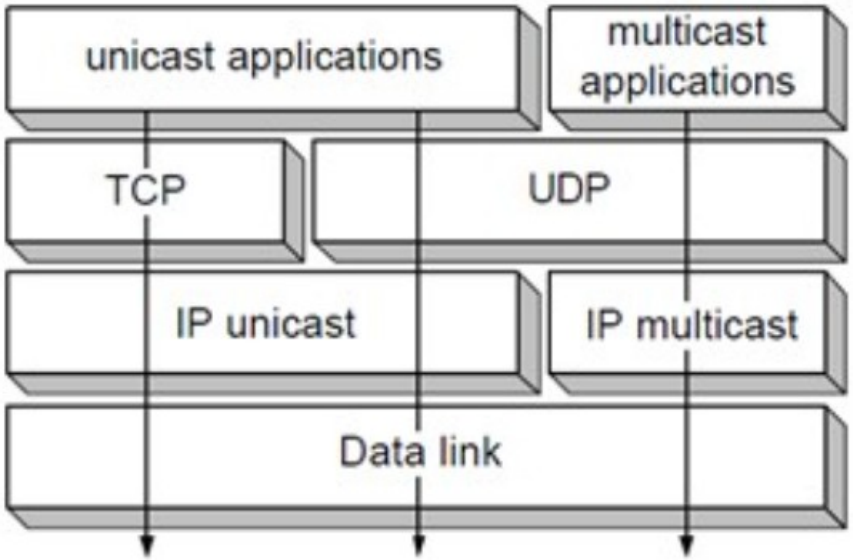
UNICASTING/MULTICASTING/BROADCASTING





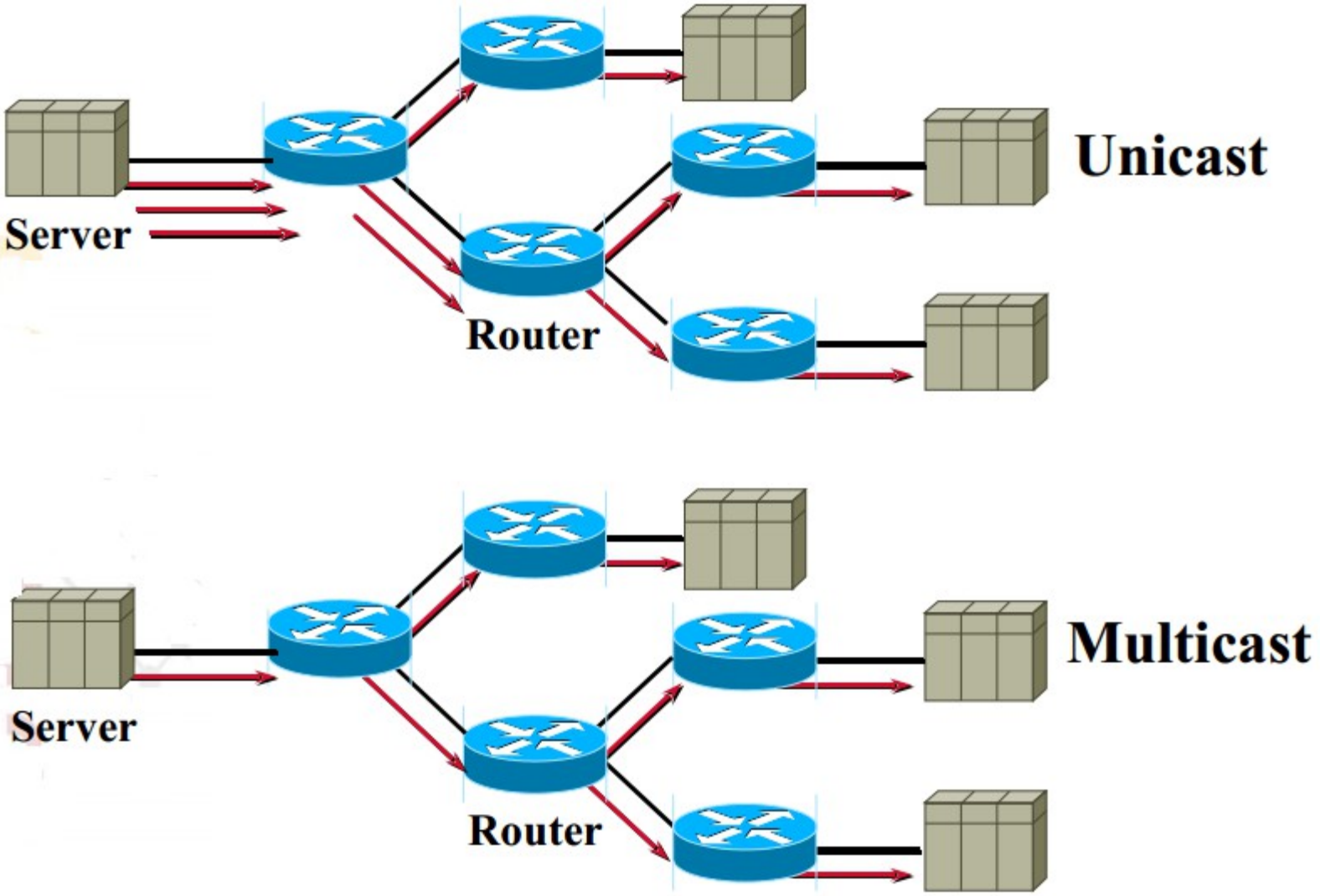


MULTICAST E PROTOCOL STACK



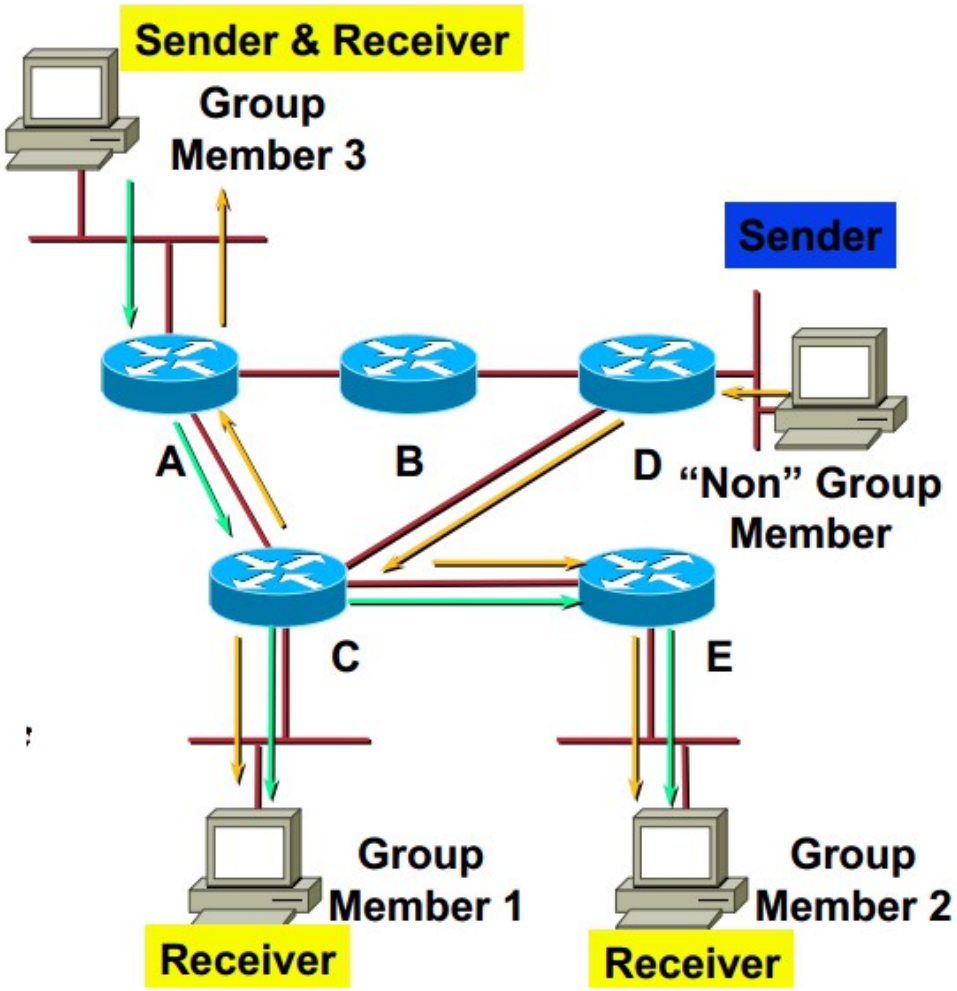
- IP multicasting utilizza UDP
- non esiste multicast TCP!

UNICAST VERSO MULTICAST



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
- mentre 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:

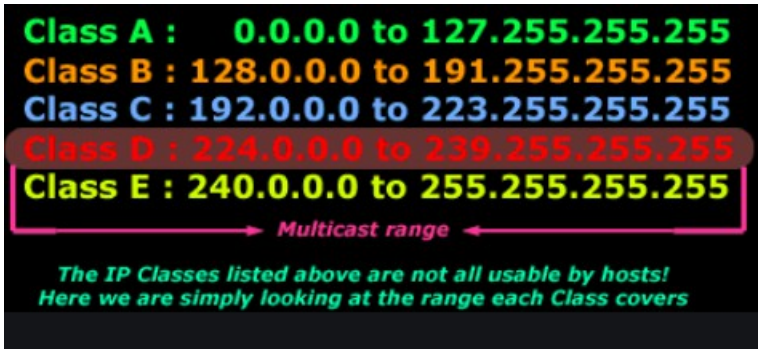
- unirsi ad un gruppo di multicast
- lasciare un gruppo
- spedire messaggi ad un gruppo
  - il messaggio viene recapitato a tutti i processi che fanno parte del gruppo in quel momento
- ricevere messaggi indirizzati ad un gruppo

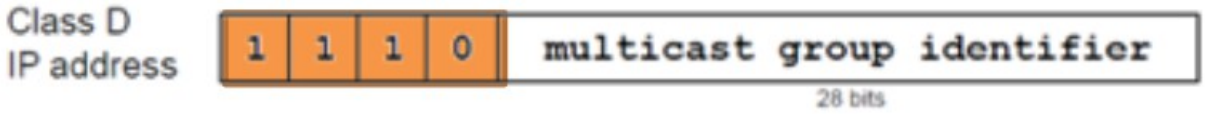
Il supporto deve fornire

- 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 ADDRESSING E SCOPING

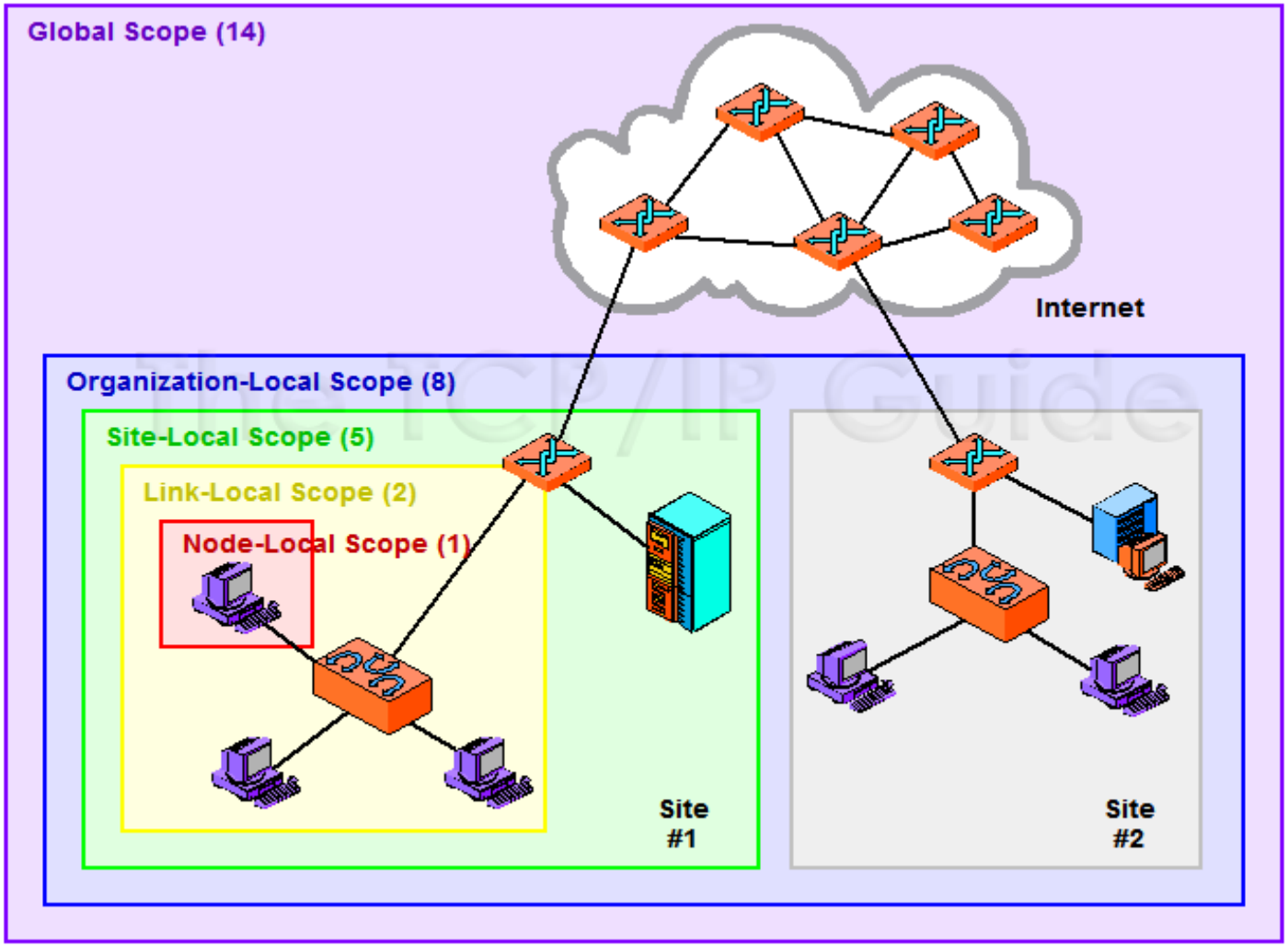
- Multicast addressing: come scegliere un indirizzo di multicast?
  - indirizzo multicast: deve essere noto "collettivamente" a tutti i partecipanti al gruppo
  - indirizzi statici: scelti da una autorità per un certo servizio
  - indirizzi dinamici: come evitare collisioni?

- Multicast scoping (scope: portata, raggio): come limitare la diffusione di un pacchetto?
  - TTL scoping: il TTL limita la diffusione del pacchetto
  - administrative scoping: a seconda dell'indirizzo in classe D scelto, la diffusione del pacchetto viene limitata ad una parte della rete i cui confini sono definiti da un amministratore di rete

TTL SCOPING

- utilizza il TTL
  - ad ogni pacchetto IP viene associato un valore rappresentato su un byte, riferito come `TTL (Time-To-Live)` del pacchetto
  - valori del `TTL` nell'intervallo `0-255`
  - indica il numero massimo di routers attraversati dal pacchetto
  - il pacchetto viene scartato dopo aver attraversato `TTL` routers
  - associazione valori di `TTL`-aree geografiche
    - `1, 16, 63,e 127`, rappresentano la sottorete locale, la rete della organizzazione di cui fa parte l'host, la rete regionale e la rete globale

ADMINISTRATIVE SCOPING



ADMINISTRATIVE SCOPING

Address type	IPv4	IPv6
Multicast	224.0.0.0 to 239.255.255.25	Begins with byte FF
MC global	224.0.1.0 to 238.255.255.255	FF0E or FF1E
Org MC	239.192.0.0/14	FF08 or FF18
MC site-local	N/A	FF05 or FF15
MC link-local	224.0.0.0	FF02 or FF12
MC node local	127.0.0.0	FF01 or FF11
Private	10.0.0.0 to 10.255.255.255 172.16.0.0 to 172.31.255.255 192.168.0.0 to 192.168.255.255	fd00::/8

ADMINISTRATIVE SCOPING

```
import java.net.*;
public class IndirizziIP {
public static void main (String args[]) throws Exception
    { InetAddress address=InetAddress.getByName(args[0]);
      if (address.isMulticastAddress()){
         if (address.isMCGlobal()){
            System.out.println("e' un indirizzo di multicast globale");}
        else if (address.isMCOrgLocal())
              {System.out.println ("e' un indirizzo organization local");}
        else if (address.isMCSiteLocal())
           {System.out.println ("e' un indirizzo site local");}
        else if(address.isMCLinkLocal())
              {System.out.println("e' un indirizzo link local");}; }
             else System.out.println("non e' un indirizzo di
                                                    Multicast");}}
```

INDIRIZZAMENTO GRUPPI DI MULTICAST

- l'allocazione degli indirizzi di multicast su Internet è una procedura molto complessa, che prevede un ampio numero di casi

- gli indirizzi possono essere
  - statici: assegnati da una autorità di controllo, utilizzati da particolari protocolli/ applicazioni.
    - l'indirizzo rimane assegnato a quel gruppo, anche se, in un certo istante non ci sono partecipanti
  - 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

    https://www.iana.org/assignments/multicast-addresses/multicast-addresses-xml
  - dall'amministratore di rete

- 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 più hosts
- assegnati dall'amministratore di una certa organizzazione
  - valgono per tutti gli host della rete amministrata

MULTICAST: CARATTERISTICHE

- utilizza il paradigma connectionless (UDP):
  - sarebbere richieste n×(n-1) connessioni per un gruppo di n applicazioni, se si usa 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 una animazione.
  - è più accettabile la perdita occasionale di un frame piuttosto che un ritardo costante tra la spedizione di due frames successivi
- si perde la 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 di provemulticast 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

SSDP SIMPLE SERVICE DISCOVERY PROTOCOL (SSDP)

- su una rete locale attivi molti protocolli che usano il multicast
- SSDP attivo sulla porta `1900` associato all'indirizzo di multicast `239.255.255.250`
- un protocollo di discovery usato per scoprire quali servizi sono disponibili in una rete
- ogni servizio è definito mediante specifiche UPnP
- il server SSDP invia pacchetti di advertisement sul gruppo di multicast
- formato di un pacchetto di advertisement

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=120
ST: urn:schemas-upnp-org:device:WANDevice:1
USN: uuid:fc4ec57e-b051-11db-88f8-
0060085db3f6::urn:schemas-upnp-org:device:WANDevice:1
EXT:
SERVER: Net-OS 5.xx UPnP/1.0
LOCATION: http://192.168.0.1:2048/etc/linuxigd/gatedesc.xml
```

SSDP SIMPLE SERVICE DISCOVERY PROTOCOL (SSDP)

Messaggio di advertisement

- USN, Unique Service Name
  - UUID (Universally Unique Identifier) che identifica una device o un servizio
- LOCATION
  - punta ad una URL
  - a quella URL l'utente può trovare una descrizione XML delle capability del servizio

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=120
ST: urn:schemas-upnp-org:device:WANDevice:1
USN: uuid:fc4ec57e-b051-11db-88f8-
0060085db3f6::urn:schemas-upnp-org:device:WANDevice:1
EXT:
SERVER: Net-OS 5.xx UPnP/1.0
LOCATION: http://192.168.0.1:2048/etc/linuxigd/gatedesc.xml
```

SNIFFING SSDP IN JAVA

```
import java.io.*; import java.net.*;
public class MulticastSniffer {
 public static void main(String[] args) {
    InetAddress group = null;
    int port = 0;
    // read the address from the command line
    try {
      group = InetAddress.getByName(args[0]);
      port = Integer.parseInt(args[1]);
    } catch (ArrayIndexOutOfBoundsException | NumberFormatException |
                                                    UnknownHostException ex)
        {
           System.err.println( "Usage: java MulticastSniffer multicast_address
                                                    port");
           System.exit(1); }
    MulticastSocket ms = null;
```

SNIFFING SSDP IN JAVA

```
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(), "8859_1");
        System.out.println(s); }
 } catch (IOException ex) { System.err.println(ex);
 } finally {
   if (ms != null) {
     try {
       ms.leaveGroup(group);
       ms.close(); } catch (IOException ex) { } } } } }
```

SNIFFING SSDP IN JAVA: ESECUZIONE

```
$ Java MulticastSniffer 239.255.255.250 1900

NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age=1801
NTS: ssdp:alive
LOCATION: http://192.168.1.1:49152/gKgq6lu34h/wps_device.xml
SERVER: Unspecified, UPnP/1.0, Unspecified
NT: urn:schemas-wifialliance-org:service:WFAWLANConfig:1
USN: uuid:03fef68b-319f-5ea7-80a8-2aea5bb7e216::urn:schemas-
wifialliance-org:service:WFAWLANConfig:1
…….
```

SPEDIRE PACCHETTI

per spedire messaggi ad un gruppo di multicast:

- creare un `DatagramSocket` su una porta anonima
- non è necessario collegare il 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
```

SPEDIRE PACCHETTI

```
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);
      DatagramSocket ms = new DatagramSocket(6789);
      ms.send(dp);
      Thread.sleep(80000);
    } catch(IOException ex){ System.out.println(ex);
    }}}
```

TIME TO LIVE

TTL 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

Valore impostato = 1 (i pacchetti di multicast non possono lasciare la rete locale)

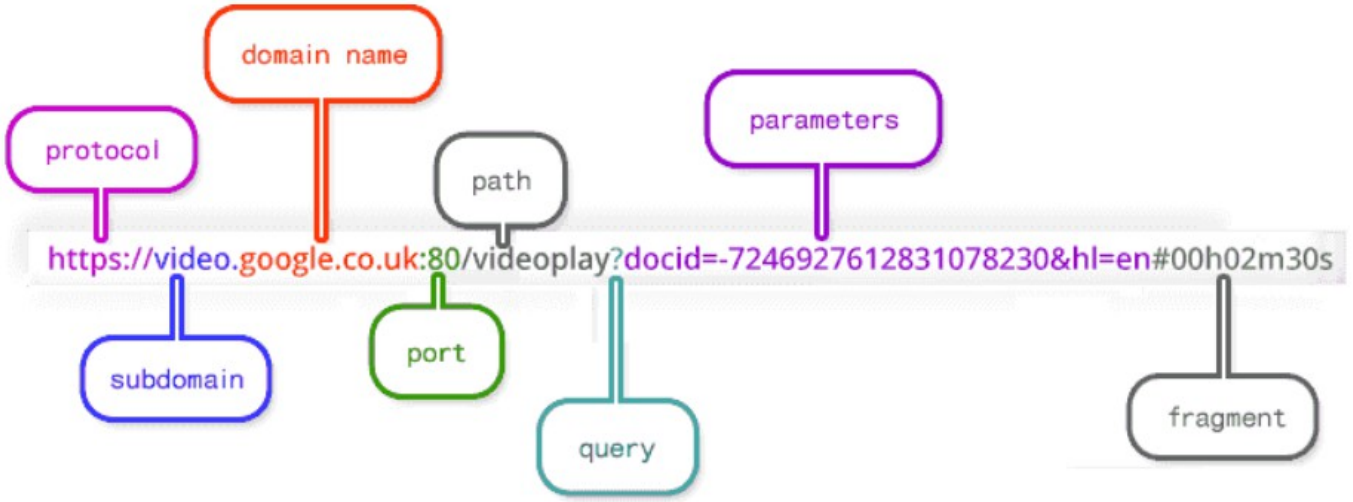
Per modificare il valore di default: posso associare il TTL al multicast socket

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

JAVA URL

- la definizione di un client che accede ad un server REST può essere effettuata in JAVA usando le seguenti classi:
  - `URL`
  - `URLConnection`
  - `HTTPURLConnection`
- più complessa la definizione di un server `REST`
- le classi `URL` ed `URLConnection` incapsulano la maggior parte della complessità relativa al ritrovamento di una informazione da un sito remoto

JAVA URL



- modellata in JAVA mediante oggetti URL

```
URL url= new URL("https://docs.oracle.com/javase/tutorial/
                                networking/urls/creatingUrls.html")
```

JAVA URL: COSTRUZIONE

```
import java.io.BufferedReader; import java.io.IOException;import java.net.URL;
import java.io.InputStreamReader; import java.net.MalformedURLException;
public class JavaNetURLExample {
  public static void main(String[] args) {
       try {// Generate absolute URL
            URL url1 = new URL("http://www.gnu.org");
            System.out.println("URL1: " + url1.toString());
            // stampa URL1: http://www.gnu.org
            // Generate URL for pages with a common base URL = www.gnu.org
            URL url2 = new URL(url1, "licenses/gpl.txt");
            System.out.println("URL2: " + url2.toString());
            // stampa URL2: http://www.gnu.org/licenses/gpl.txt
            // Generate URL from different pieces of data
            URL url3 = new URL("http", "www.gnu.org", "/licenses/gpl.txt");
            System.out.println("URL3: " + url3.toString());
            // stampa URL3: http://www.gnu.org/licenses/gpl.txt
```

LEGGERE CONTENUTO DA UNA URL

- se si deve solamente scaricare il contenuto della risorsa identificata dalla URL, è possibile usare il metodo `openStream` della classe URL.

```
InputStream stream= url.openStream();
```

  - restituisce un oggetto `InputStream`
  - usando questo oggetto, si possono leggere i contenuti della risorsa

- `url.openStream()` è la forma abbreviata di `url.openConnection().getInputStream()`

```
InputStream uin  = url.openStream();
BufferedReader in = new BufferedReader(new InputStreamReader(uin));
String line;
while ((line = in.readLine()) != null) {
            //    process line;
```

- invia un richiesta `HTTP GET`, poi la risposta è disponibile nello stream

LEGGERE CONTENUTO DA UNA URL

```
URL url4 = new URL("http", "www.gnu.org", 80, "/licenses/gpl.txt");
System.out.println("URL4: " + url4.toString() + "\n");
//stampa URL4: http://www.gnu.org:80/licenses/gpl.txt
// Open URL stream as an input stream and print contents to command line
try (BufferedReader in = new BufferedReader(new
                                InputStreamReader(url4.openStream())))
    {String inputLine;
     // Read the "gpl.txt" text file from its URL representation
     System.out.println("/***** File content (URL4) *****/\n");
     while((inputLine = in.readLine()) != null) {
           System.out.println(inputLine);   }
     } catch (IOException ioe) { ioe.printStackTrace(System.err); }
  } catch (MalformedURLException mue) {
           mue.printStackTrace(System.err); }
   }}
```

GET GPL.TXT…

```
/***** File content (URL4) *****/
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007
 Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.
                            Preamble
  The GNU General Public License is a free, copyleft license for
software and other kinds of works.
  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.
```

URLCONNECTION

- se sono richieste ulteriori informazioni su una risorsa, oppure si vuole modificare una risorsa e crearne una nuova occorre usare la classe `URLConnection`
- invocazione del metodo `openConnection` per ottenere un oggetto `URLConnection`

```
URLConnection connection = url.openConnection();
```

- connettersi alla risorsa remota invocando connect `connection.connect();`
- possibile impostare proprietà della connessione come
  - `SetDoInput`
  - `SetDoOutput`
  - `SetIfModifiedSince`
  - `SetUseCaches`
  - `SetAllowUserInteraction`
  - `SetRequestProperty`

URLCONNECTION

```
try {
      URL u = new URL("http://www.bogus.com");
      URLConnection uc = u.openConnection();
      InputStream raw = uc.getInputStream();
       // read from URL ...
      }
catch (MalformedURLException ex) {
      System.err.println(ex);}
catch (IOException ex) {
              System.err.println(ex); }
```

- apre una connessione HTTP con quella URL
- vengono inviate un insieme di informazioni che descrivono il contenuto reperibile a quella URL
- dopo aver effettuato la connessione, si ricevono diverse informazioni che descrivono il contenuto

URLCONNECTION

- due metodi per reperire il valore degli header

```
public String getHeaderFieldKey (int n)
public String getHeaderField (int n)
```

- per convenienza, esistono inoltre dei metodi per reprire il valore di alcuni header standard:
  - `getContentType:` restituisce il `MIME media type` dei dati contenuti nel corpo del messaggio
    - `null`, se il content type non è disponibile
    - se il content type è `text/*`, l'header può anche contenere informazioni sul charset utilizzato,
  - `getContentLength`
  - `getContentEncoding`
  - `getDate`
  - `getExpiration`

STAMPA DEGLI HEADER

```
import java.io.*; import java.net.*;
public class AllHeaders {
   public static void main(String[] args) {
    for (int i = 0; i < args.length; i++) {
      try {
         URL u = new URL(args[i]);
         URLConnection uc = u.openConnection();
         boolean fine=false; int j=1;
         while (!(fine)) {
           String header = uc.getHeaderField(j);
           if (header == null) fine=true;
           else System.out.println(uc.getHeaderFieldKey(j) + ": " + header);
           j++;}
      } catch (MalformedURLException ex) {
          System.err.println(args[i] + " is not a URL I understand.");
      } catch (IOException ex) { System.err.println(ex);}
         System.out.println();}}}
```

HEADER: CON URL http://www.internic.net

```
Date: Wed, 11 Dec 2019 23:11:40 GMT
Server: Apache
Content-Security-Policy: upgrade-insecure-requests
Vary: Accept-Encoding
Last-Modified: Sun, 11 Jun 2017 00:01:00 GMT
ETag: "23ad-551a3e5c58700"
Accept-Ranges: bytes
Content-Length: 9133
Cache-Control: max-age=3600
Expires: Thu, 12 Dec 2019 00:11:40 GMT
X-Frame-Options: SAMEORIGIN
Referrer-Policy: origin-when-cross-origin
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
Content-Language: en
```

ENCODING AWARE CONTENT VIEWER

- accedere ad una risorsa tramite URL : ad esempio, un file HTML
- visualizzare il contenuto del file, tenendo presente della codifica utilizzata per i caratteri del file
- utilizza l'header HTTP `ContentType`
- HTTP `ContentType`
  - media type della risorsa reperita
  - immagini `image/png` `image/jpg`
  - testo `text/html; charset=UTF-8`

ENCODING AWARE CONTENT VIEWER

```
import java.io.*; import java.net.*;
public class EncodingAwareSourceViewer {
  public static void main (String[] args) {
    for (int i = 0; i < args.length; i++) {
      try {
        // set default encoding
        String encoding = "ISO-8859-1";
        URL u = new URL(args[i]);
        URLConnection uc = u.openConnection();
        String contentType = uc.getContentType();
        System.out.println("contenttype"+contentType);
        int encodingStart = contentType.indexOf("charset=");
        if (encodingStart != -1) {
            encoding = contentType.substring(encodingStart + 8);
        }
        System.out.println("encoding"+encoding); }
```

ENCODING AWARE CONTENT VIEWER

```
    InputStream in = new BufferedInputStream(uc.getInputStream());
    Reader r = new InputStreamReader(in, encoding);
    int c;
    while ((c = r.read()) != -1) {
        System.out.print((char) c);
        }
  r.close();
} catch (MalformedURLException ex) {
                System.err.println(args[0] + " is not a parseable URL");
} catch (UnsupportedEncodingException ex) {
                 System.err.println(
                "Server sent an encoding Java does not support: " +
                                                    ex.getMessage());
} catch (IOException ex) {
                    System.err.println(ex);
     }}}}
```

ASSIGNMENT

Definire un Server `TimeServer`, che

- invia su un gruppo di multicast dategroup, ad intervalli regolari, la data e l'ora.
- attende tra un invio ed il successivo un intervallo di tempo simulata mediante il metodo `sleep()`.
- l'indirizzo `IP` di dategroup viene introdotto da 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.