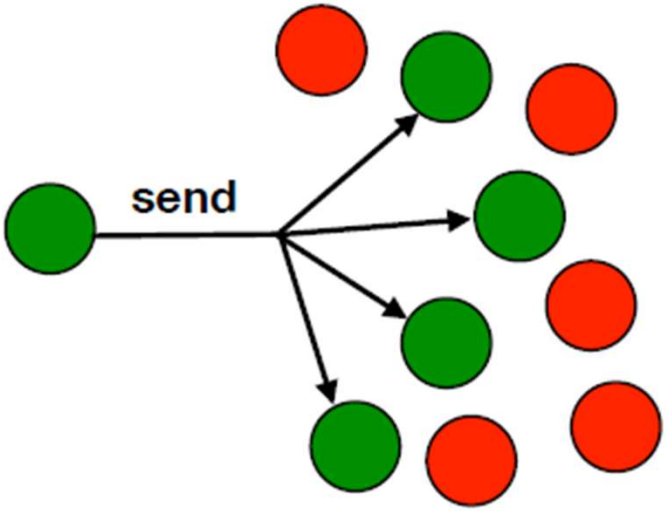
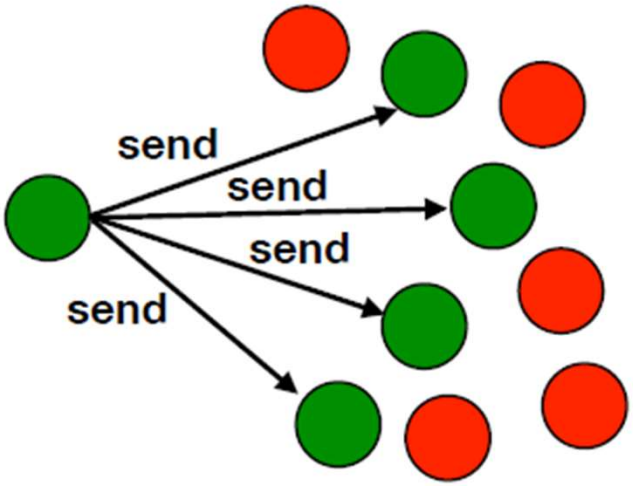
Multicast e MulticastDatagramSocket

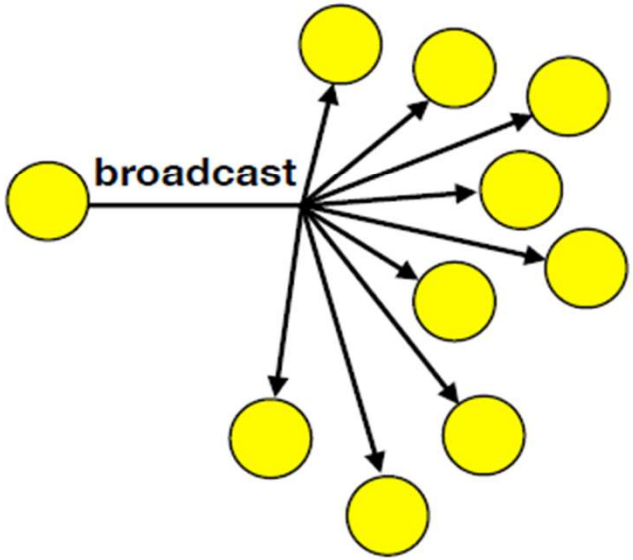
PARADIGMI DI COMUNICAZIONE



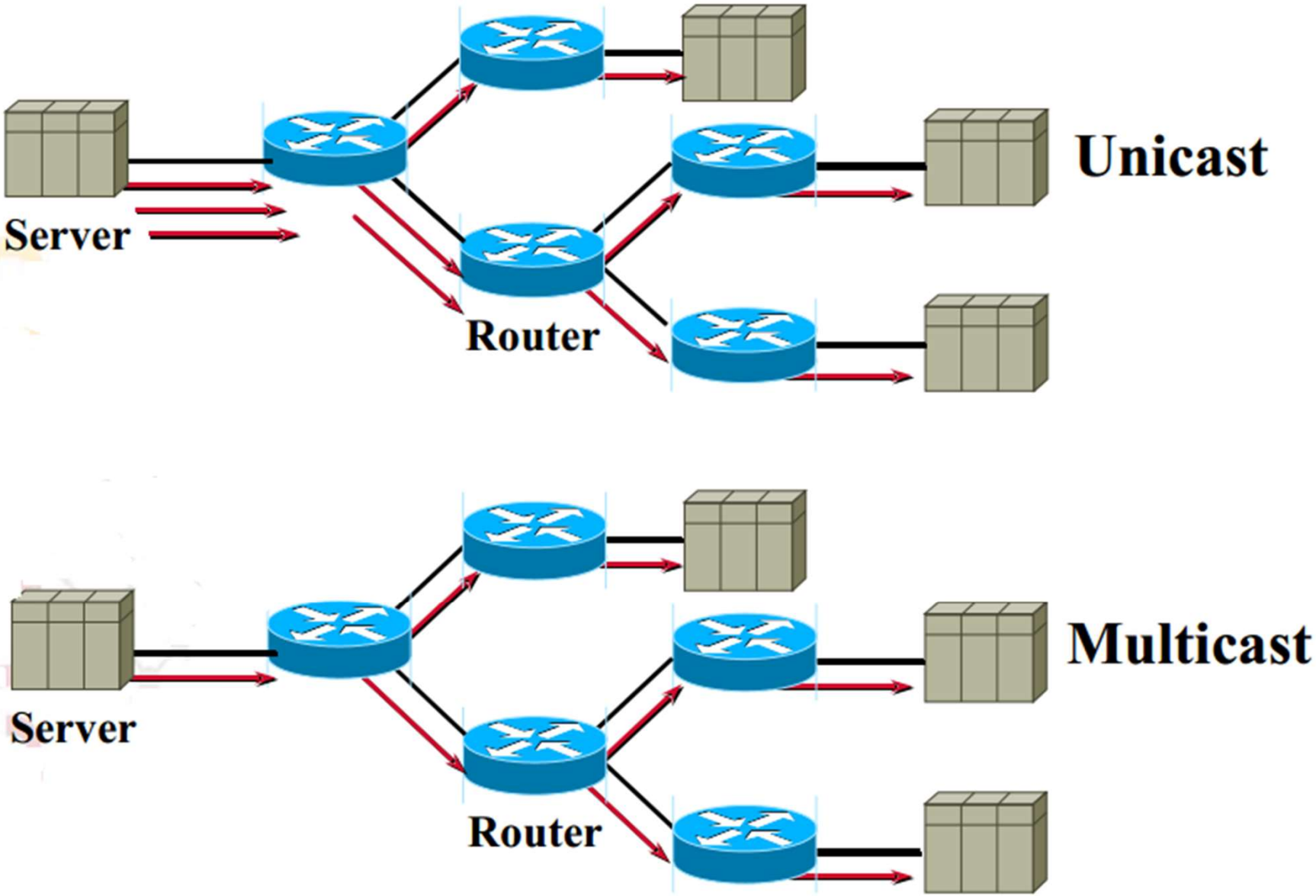
Multicast



Unicast

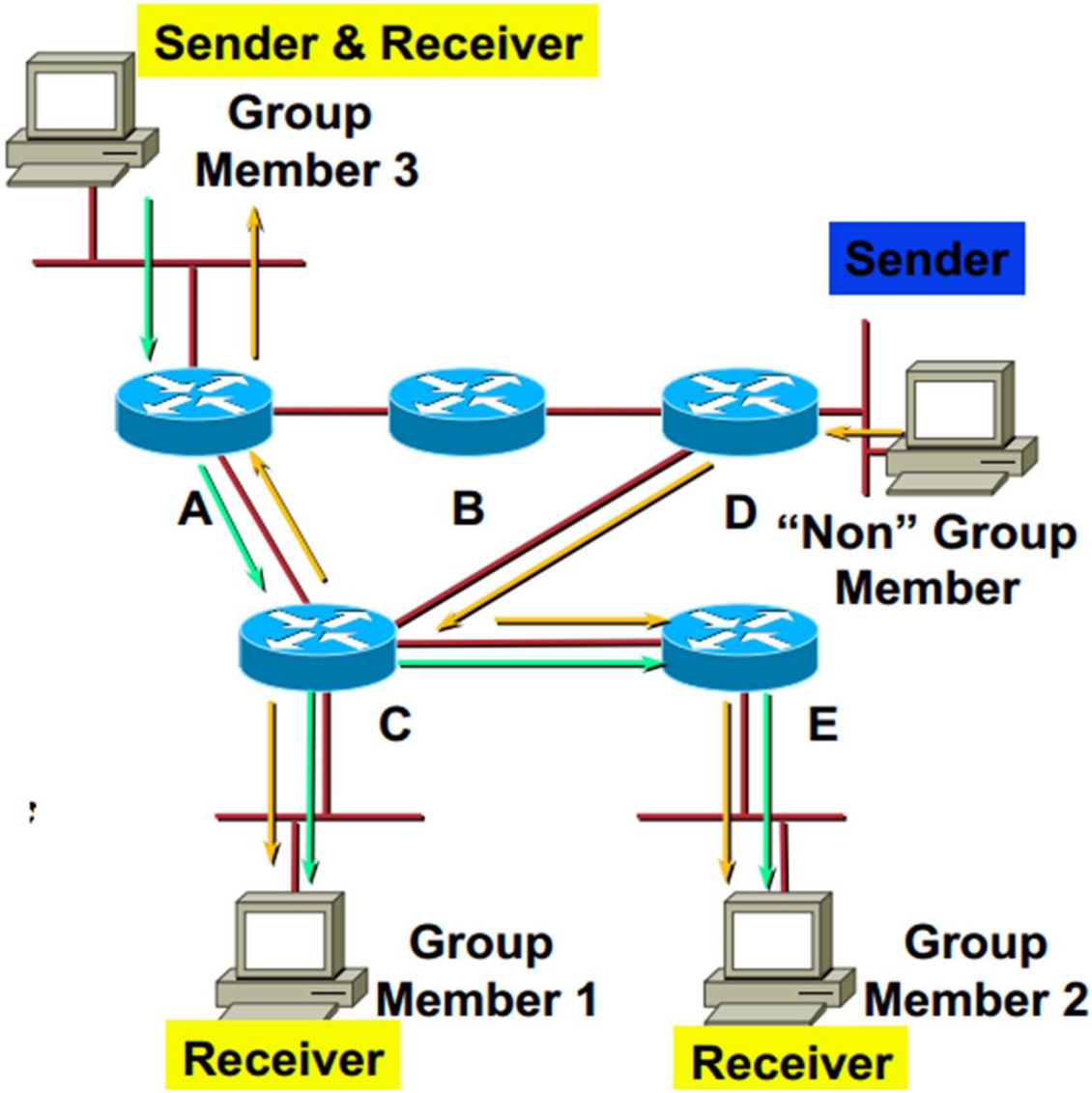


UNICAST VERSO MULTICAST



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
- non occorre essere membri del gruppo per inviare i messaggi su di esso



MULTICAST API

deve contenere almeno primitive per:

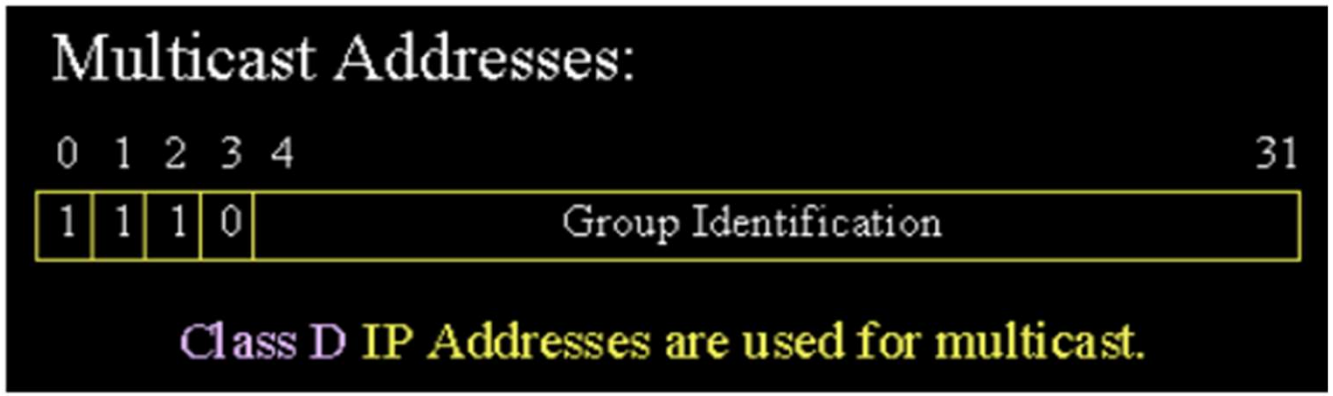
- unirsi ad un gruppo di multicast
- lasciare un gruppo
- spedire messaggi ad un gruppo. Il messaggio viene recapitato a tutti i processi che fanno parte del gruppo in quel momento
- ricevere messaggi indirizzati ad un gruppo

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
- un meccanismo che ottimizzi l'uso della rete nel caso di invio di pacchetti ad un gruppo di multicast

SCHEMA DI INDIRIZZAMENTO

- Basato sull'idea di riservare un certo insieme di indirizzi IP per il multicast
- IPV6: tutti gli indirizzi di multicast iniziano con FF
- IPV4: indirizzo di un gruppo è un indirizzo in classe D
  - [224.0.0.0 – 239.255.255.255]
  - riservati da IANA
  - i primi 4 bit del primo ottetto = 1110
  - i restanti bit identificano il particolare gruppo



INDIRIZZAMENTO GRUPPI DI MULTICAST

Multicast addressing: come scegliere un indirizzo di multicast?

- indirizzo multicast: deve essere noto “collettivamente” a tutti i partecipanti al gruppo
- L'allocazione degli indirizzi di multicast su Internet è una procedura molto complessa, che prevede un ampio numero di casi
- Gli indirizzi possono essere assegnati in modo
  - statico: 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
  - dinamico: si utilizzano 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) o 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

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 entro un unico dominio amministrativo

  Multicast Address Dynamic Client Allocation Protocol (MADCAP)

- Nell'ambito più generale della rete

  Multicast Address Set Claim (MASC), ecc.

MULTICAST SCOPING

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 di multicast scelto, la diffusione del pacchetto viene limitata ad una parte della rete i cui confini sono definiti da un amministratore di rete

TTL SCOPING

- IP Multicast Scoping: limita la diffusione di un pacchetto multicast

- ad ogni pacchetto IP viene associato un valore rappresentato su un byte, riferito come TTL (Time-To-Live) del pacchetto

- TTL scoping: meccanismo poco utilizzato
  - 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, ad es., limitano il pacchetto all'interno della la sottorete locale, la rete della organizzazione di cui fa parte l'host, la rete regionale e la rete globale, rispettivamente.

Administrative scoping

Range Start Address	Range End Address	Description
224.0.0.0	224.0.0.255	Reserved for special “well-known” multicast addresses.
224.0.1.0	238.255.255.255	Globally-scoped (Internet-wide) multicast addresses.
239.0.0.0	239.255.255.255	Administratively-scoped (local) multicast addresses.

1. **224.0.0.0/24**. I pacchetti con un indirizzo in questo intervallo hanno assegnato un **link-local scope** (pratica comune è di trasmettere i pacchetti con Time To Live (TTL) di 1 in modo che non vadano oltre la sottorete locale)

- Esempio:
- [224.0.0.1] `e il gruppo “all-hosts’. Se si invia un pacchetto di “ping” a questo gruppo, tutti gli host che supportano il multicast dovrebbero rispondere dato che un host deve fare il join a questo gruppo all’atto dell’inizializzazione delle interfacce che supportano il multicast;
- [224.0.0.2] `e il gruppo “all-routers”. Tutti i router devono fare il join a questo gruppo sulle interfacce che supportano il multicast.

http://www.tcpipguide.com/free/t_IPMulticastAddressing.htm

Administrative scoping

2. **Globally-scoped Internet-wide multicast address range**, contiene vari indirizzi
   - Indirizzi che possono essere inoltrati in Internet
   - Es. [224.0.1.1] `e l'indirizzo per NTP (Network Time Protocol).
3. **Administrative scoping** : intervallo di indirizzi da 239.0.0.0 a 239.255.255.255
   - «analogo» degli indirizzi IP unicast privati
   - I pacchetti indirizzati a indirizzi multicast con ambito amministrativo non attraversano i confini amministrativi.
   - Gli indirizzi multicast con administrative scoping sono assegnati localmente e non è necessario che siano univoci oltre i confini amministrativi

https://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml

ADMINISTRATIVE SCOPING

```
import java.net.*;
public class MulticastScope {
 public static void main (String args[]) throws Exception {
 InetAddress address=InetAddress.getByName("224.0.1.1"); //multicast globale
//InetAddress address=InetAddress.getByName("224.0.0.1"); //link local
//InetAddress address=InetAddress.getByName("239.192.0.101"); //organiz.local
//InetAddress address=InetAddress.getByName("239.255.0.101"); //site local
 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"); }}
```

- Quale servizio di trasporto usare?

- CONNECTION-ORIENTED O CONNECTIONLESS?
- TCP O UDP?

MULTICAST: CARATTERISTICHE

- Utilizza il paradigma connectionless:
  - una comunicazione connection oriented richiederebbe la gestione di un alto numero di connessioni
  - richieste n(n-1) connessioni per un gruppo di n applicazioni

- comunicazione connectionless adatta per il tipo di applicazioni verso cui è orientato il multicast (trasmissione di dati video/audio).
  - invio dei frame di una animazione. E' più accettabile la perdita occasionale di un frame piuttosto che un ritardo tra la spedizione di due frame successivi

JAVA API PER MULTICAST

Classe **MulticastSocket**:

- socket su cui ricevere i messaggi da un gruppo di multicast
- estende la classe **DatagramSocket**
- effettua overriding dei metodi esistenti
- È una DatagramSocket che offre metodi aggiuntivi per l'implementazione di funzionalità tipiche del multicast

- Per ricevere messaggi in un gruppo multicast
  1. Istanziare una `MulticastSocket`
  2. `joinGroup(SocketAddress mcastaddr, NetworkInterface netIf)`, unirsi al Gruppo su una certa interfaccia di rete
  3. `receive()` per attendere la ricezione del DatagramPacket

JAVA API PER MULTICAST

```
import java.net.*;
import java.io.*;
public class Multicast {
public static void main (String [ ] args) {
   int port = 6789;
   try {
         MulticastSocket ms = new MulticastSocket(port);
         InetSocketAddress group = new
                        InetSocketAddress("239.255.1.3", port);
         NetworkInterface netIf =
                        NetworkInterface.getByName("wlan1");
         ms.joinGroup(group, netIf);
     }
    catch (IOException ex) {System.out.println("errore"); }}}
```

`joinGroup(SocketAddress mcastaddr, NetworkInterface netIf)`
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

JAVA API PER MULTICAST

```
import java.io.*; import java.net.*;
public class ProvaMulticast {
public static void main (String args[]) throws Exception
{
     byte[] buf = new byte[10];
     InetSocketAddress ia = new InetSocketAddress("239.255.1.3", 4000);
     DatagramPacket dp = new DatagramPacket(buf,buf.length);
     MulticastSocket ms = new MulticastSocket(4000);
     NetworkInterface netIf = NetworkInterface
                                        .getByName("wlan1");
     ms.joinGroup(ia, netIf);
     ms.receive(dp);
     }
 }
```

JAVA API PER MULTICAST

Multiple MulticastSockets possono sottoscriversi contemporaneamente a un Gruppo multicast e ad una porta per ricevere I datagrammi

E' possible configurare la proprietà REUSE_ADDR. Se settata a false (prima di fare il binding!), impedisce di associare più instanze di socket multicast allo stesso SocketAddress

```
sock.setReuseAddress(false);
```

La proprietà è settata per default a true per le MulticastSocket

se attivo due istanze di ProvaMulticast sullo stesso host (con REUSE_ADD a 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

Multicast receiver

Dopo aver ricevuto in input 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 MulticastReceiver {
 public static void main (String[] args) {
     String addressName =  "239.255.1.3";
     int port = 6789;
     if (args.length > 0) {
         try {
             addressName = args[0];
             port = Integer.parseInt(args[1]);
         }   }
     catch(Exception e){
         System.out.println("Uso:java multicastsniffer multicast_address
         port"); }
     try (MulticastSocket ms = new MulticastSocket(port)){
         InetSocketAddress group = new
         InetSocketAddress(InetAddress.getByName(addressName), port);
         NetworkInterface netIf =     NetworkInterface
                                     .getByName("wlan1");
         ms.joinGroup(group, netIf);
```

Multicast receiver

```
        byte[] buffer = new byte[8192];
        for (int i=0;i<10; i++) {
            try {
                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);
            }
        }
    ms.leaveGroup(group,netIf);
    }
    catch (IllegalArgumentException ex1){}
    catch (IOException ex){}
    }
}
```

Per verificare che vi siete uniti al gruppo?

- windows
  - `netsh interface ip show joins`
- Linux
  - `netstat –g`
  - `ip show joins`

```
Interfaccia 9: Wi-Fi

Ambito       Riferimenti  Ultimo  Indirizzo
---------    -----------  ------  -------------------------------
0                      0  No      224.0.0.1
0                      2  SÌ      224.0.0.251
0                      1  SÌ      224.0.0.252
0                      1  SÌ      239.255.1.3
0                      0  SÌ      239.255.1.4
0                      1  SÌ      239.255.255.250
```

SPEDIRE PACCHETTI

- Per spedire messaggi ad un gruppo di multicast:
  - creare un DatagramSocket 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
```

SPEDIRE PACCHETTI

```
import java.io.*;
import java.net.*;
public class MulticastSender {
public static void main (String args[]) {
   try{
         InetAddress ia=InetAddress.getByName("239.255.1.3");
         byte[] data;
         data="hello".getBytes();
         int port= 6789;
         DatagramPacket dp = new
         DatagramPacket(data,data.length,ia,
                port);
         DatagramSocket ms = new DatagramSocket(6800);
         ms.send(dp);
         Thread.sleep(8000);
    }
    catch(IOException ex){ System.out.println(ex);}}}
```

TIME TO LIVE

TTL Scoping, implementazione

- il mittente specifica un valore per il 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));
```

Esercizio 10 – Welcome Multicast

- Definire un Server WelcomeServer, che
  - invia su un gruppo di multicast (*welcomegroup**), ad intervalli regolari, un messaggio di «welcome».
  - attende tra un invio ed il successivo un intervallo di tempo simulato mediante il metodo sleep( ).
- Definire un client WelcomeClient che si unisce a *welcomegroup* e riceve un messaggio di welcome, quindi termina.

* Ad esempio con indirizzo IP 239.255.1.3

Assignment 10 – Multicast Date Server

- 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 simulato 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.

URL, URLConnection, HttpURLConnection

UNIFORM RESOURCE LOCATOR

- URL è un acronimo per Uniform Resource Locator: specifica
  - la locazione (riferimento)di una risorsa su Internet e come reperire quella risorsa
  - la prima parte di una URL indica un protocollo, come HTTP o FTP
- La risorsa può essere un file su un host, ma può anche puntare ad altre risorse:
  - una query per un database;
  - l'output di un **comando**
- esempio di URL: http://java.sun.com
  - http: identificativo del protocollo.
  - java.sun.com: nome della risorsa.

URL Class

- Classe `java.net.URL` fornisce un'astrazione di una URL, es.
- http://www.unipi.it o ftp://ftp.redhat.com/pub/
- Oggetto con campi quali, ad es. scheme, hostname, port, path,etc.

- Vari costruttori (vedi slide seguente)
- `MalformedURLexception` se:
  - Il protocollo non è supportato
  - La URL non è sintatticamente corretta

URL: CREAZIONE

- oggetto URL che rappresenta una URL assoluta

```
URL cli = new URL("http://www.di.unipi.it/");
```

- URL relativa, con la forma

```
URL (URL baseURL, String relativeURL)
```

```
URL cli = new URL("http://www.di.unipi.it/");
URL faq = new URL(cli, "faq");
```

- URL costruita a partire dalle sue componenti:

```
try  {
    URL url = new URL("http", "didattica.di.unipi.it",
                                "/laurea-in-informatica/");}
catch (MalformedURLException e) {
    System.err.println(e); } }
```

WHAT PROTOCOL IS SUPPORTED BY THE JVM?

```
import java.net.*;
public class ProtocolTester {
  public static void main(String[] args) {
    String host = "didattica.di.unipi.it";
    String file = "laurea-in-informatica/";
    String[] schemes = {"http", "https", "ftp", "mailto",
                        "telnet", "file", "ldap", "gopher",
                        "jdbc", "rmi", "jndi", "jar",
                        "doc", "netdoc", "nfs", "verbatim",
                         "finger", "daytime", "systemresource"};
    for (int i = 0; i < schemes.length; i++) {
            System.out.print(schemes[i] + " is ");
            try { URL u = new URL(schemes[i], host, file);
                  System.out.println("supported."); }
            catch (MalformedURLException e) {
                        System.out.println("not supported."); } } } }
```

Richiesta GET

- nella forma più semplice, l'invio di una richiesta con HTTP method GET utilizza il metodo openStream()

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

- invia un richiesta HTTP GET, la risposta è disponibile nello stream (restituisce un InputStream)

Recuperare contenuto da una URL

```
import java.io.*;
import java.net.*;
public class SourceViewer {
  public static void main (String[] args) {
    if (args.length > 0) {
      InputStream in = null;
      try {
        // Open the URL for reading
        URL u = new URL(args[0]);
        in = u.openStream();
        // buffer the input to increase performance
        in = new BufferedInputStream(in);
        // chain the InputStream to a Reader
        Reader r = new InputStreamReader(in);
        int c;
```

Recuperare contenuto da una URL

```
while ((c = r.read()) != -1) {
            System.out.print((char) c);
        }
    } catch (MalformedURLException ex) {
      System.err.println(args[0] + " is not a parseable URL");
    } catch (IOException ex) {
      System.err.println(ex);
    } finally {
      if (in != null) {
        try {
          in.close();
        } catch (IOException e) {
          // ignore
        }
      }
     }  }  }  }
```

URLCONNECTION

- classe astratta implementata da classi specializzate nel controllo di una connessione ad una URL
  - implementazioni diverse in base al protocollo

- si ottiene richiamando metodo openConnection() da un oggetto di tipo URL
- permette di
  - scegliere diversi parametri della connessione
  - analizzare gli header inviati da un server e settare i campi header del client
  - leggere ed anche di scrivere sulla connessione. Permette invio di dati ad un web server con metodi POST e PUT di HTTP

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);
      }
```

URLCONNECTION

- l'header possiede diverse informazioni, che possono essere reperite con metodi JAVA
  - content type, length, charset, expiration date, modification date
  - ed altri headers.....
- poiché non vi è limite al numero di header...

```
public String getHeaderField (String name)
public String getHeaderFieldKey (int i)
public String getHeaderFieldKey (int i)
```

restituiscono, rispettivamente chiave e valore dell'i-esimo header

Example: Print out entire HTTP header.

PRINT ALL HTTP 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();}}}
```

PRINT ALL HTTP HEADER

```
Date: Sat, 09 Dec 2017 13:32:12 GMT
Server: Apache
Set-Cookie: 384fa47606106959ed1254676261f6f6=1i8a58ukf3slev2rllpnuir414;
path=/; secure; HttpOnly
P3P: CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM", CP="NOI ADM DEV
PSAi COM NAV OUR OTRo STP IND DEM"
Expires: Wed, 17 Aug 2005 00:00:00 GMT
Last-Modified: Sat, 09 Dec 2017 13:19:31 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-
check=0, no-cache
Pragma: no-cache, no-cache
ETag: d8c221bea9852965e456cafe8772c509
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
```

URLCONNECTION

```
public String getContentType()
```

- restituisce il MIME media type dei dati contenuti nel corpo del messaggio, null, se il content type non è disponibile

- `getContentEncoding`
- `getContentLength`
- `getDate`
- `getExpiration`
- `getLastModifed`
- …

HttpURLCONNECTION

- sottoclasse astratta di URLConnection.
- fornisce metodi addizionali per lavorare con URL HTTP
  - metodi get/set. Ad es. setRequestMethod (GET, POST, HEAD, ecc.)
  - decidere se seguire redirect
  - reperire i response code

```
public int getResponseCode() throws IOException
public String getResponseMessage() throws  IOException
```

  - determinare se è utilizzato un proxy

```
public abstract boolean usingProxy();
```

```
URL u = new URL("http://www.di.unipi.it");
HttpURLConnection con =(HttpURLConnection)
                                    u.openConnection();
```

HttpURLCONNECTION

```
con.setRequestMethod("GET");
con.setRequestProperty("User-Agent", USER_AGENT);
int responseCode = con.getResponseCode();
System.out.println("GET Response Code :: " +
                                        responseCode);
if (responseCode == HttpURLConnection.HTTP_OK) {
    …
}
```