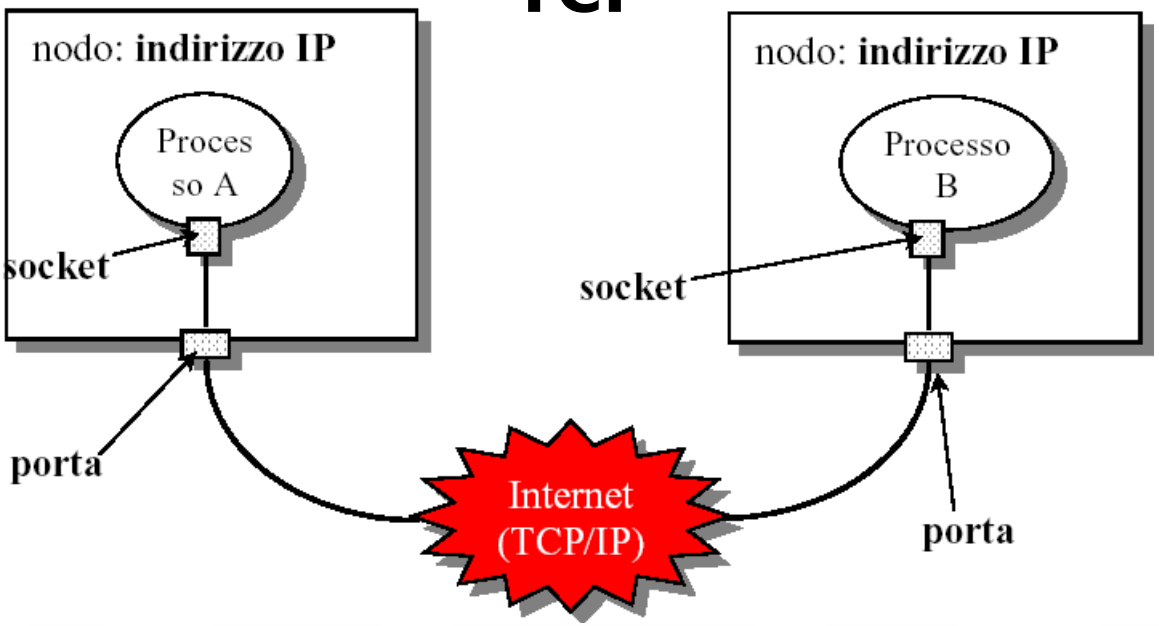
Università degli Studi della Calabria
Corso di Laurea in Ingegneria Informatica
A.A. 2010/2011

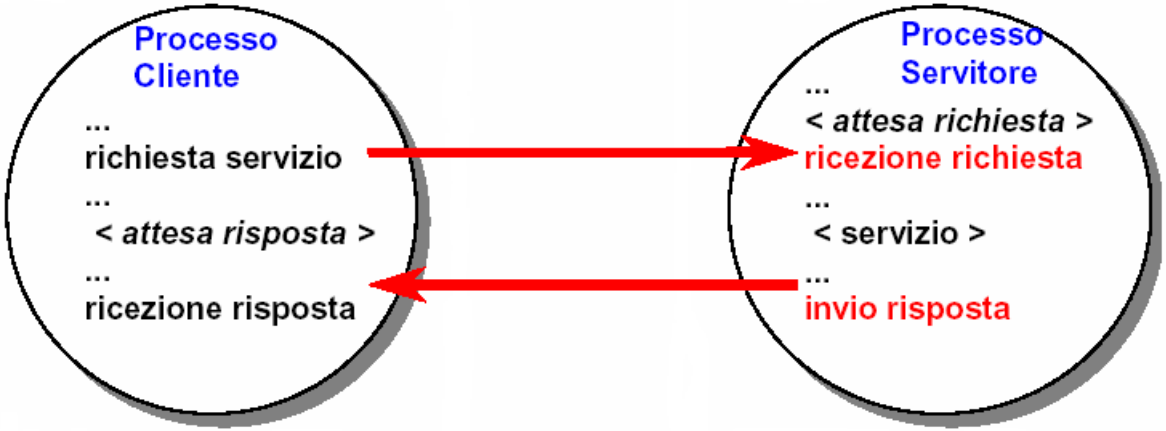
Corso di Reti di Calcolatori

Lucidi delle Esercitazioni

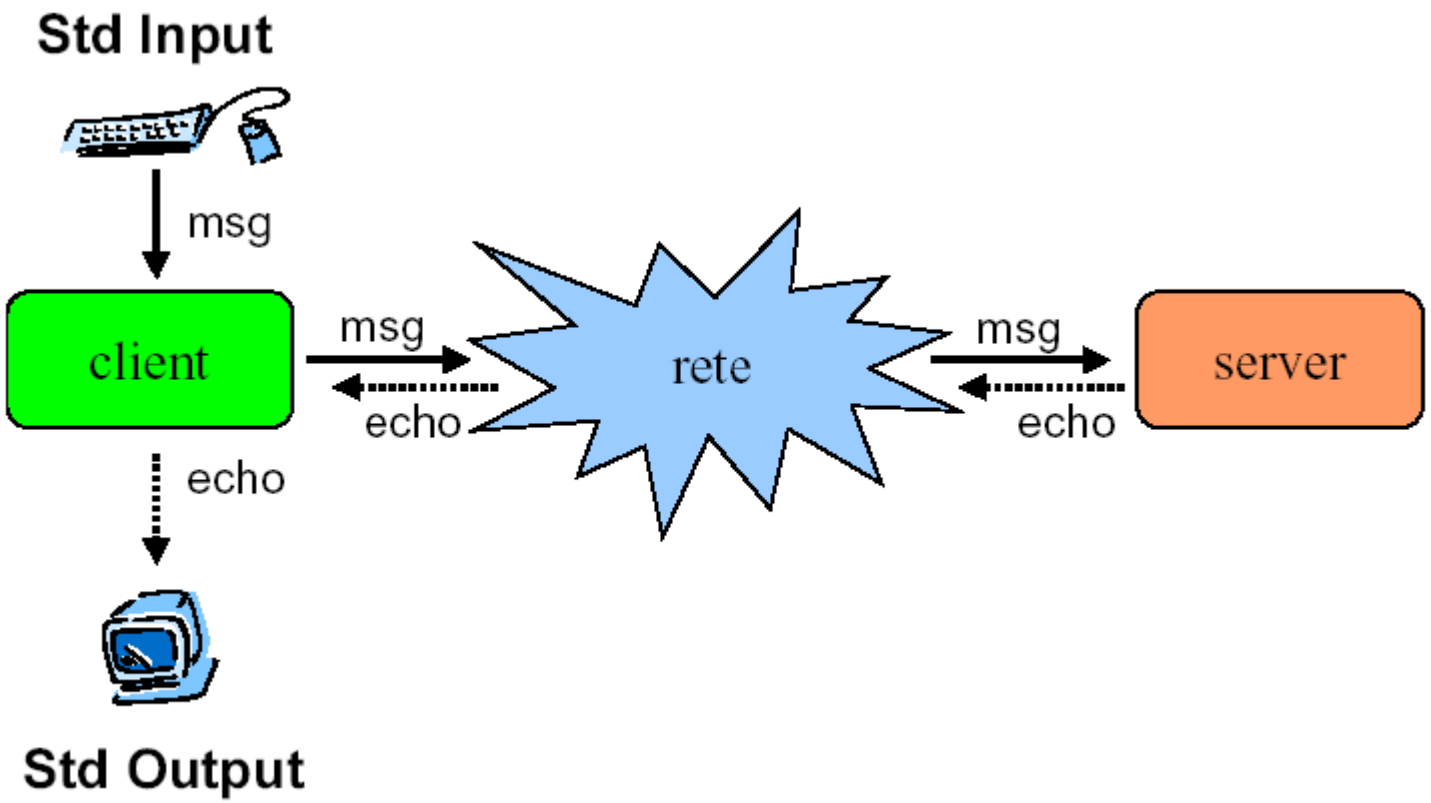
Raffaele Giordanelli

Socket TCP


nodo: indirizzo IP
Proces so A
socket
porta
nodo: indirizzo IP
Processo B
socket
porta
Internet (TCP/IP)


Processo Cliente
...
richiesta servizio
...
< attesa risposta >
...
ricezione risposta
Processo Servitore
...
< attesa richiesta >
ricezione richiesta
...
< servizio >
...
invio risposta

Echo Application



Echo Client (1)

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

public class EchoClient {
    public static void main(String[] args) throws IOException {

      /* Lanciando il programma senza argomenti si ottiene il local loopback IP address,
         per testarlo in locale (client e server sulla stessa macchina), altrimenti
         si possono passare da linea di comando l'indirizzo o il nome della macchina remota */

      InetAddress addr;
      if (args.length == 0) addr = InetAddress.getByName(null);
      else addr = InetAddress.getByName(args[0]);
      Socket socket=null;
      BufferedReader in=null, stdIn=null;
      PrintWriter out=null;
      try {
     // creazione socket
     socket = new Socket(addr, EchoServer.PORT);
     System.out.println("EchoClient: started");
     System.out.println("Client Socket: "+ socket);
```

Echo Client (2)

```
// creazione stream di input da socket
    InputStreamReader isr = new
InputStreamReader( socket.getInputStream());
    in = new BufferedReader(isr);

// creazione stream di output su socket
    OutputStreamWriter osw = new
OutputStreamWriter(socket.getOutputStream());
    BufferedWriter bw = new BufferedWriter(osw);
    out = new PrintWriter(bw, true);

// creazione stream di input da tastiera
    stdIn = new BufferedReader(new InputStreamReader(System.in));
    String userInput;
```

Echo Client (3)

```
// ciclo di lettura da tastiera, invio al server e stampa risposta
    while (true){
          userInput = stdIn.readLine();
          out.println(userInput);
          if (userInput.equals("END")) break;
          System.out.println("Echo: " + in.readLine());
     }
  }catch (UnknownHostException e) {
    System.err.println("Don't know about host "+ addr);
    System.exit(1);
  } catch (IOException e) {
      System.err.println("Couldn't get I/O for the connection to: " + addr);
      System.exit(1);
  }
  System.out.println("EchoClient: closing...");
  out.close();
  in.close();
  stdIn.close();
  socket.close();
  }
} //EchoClient
```

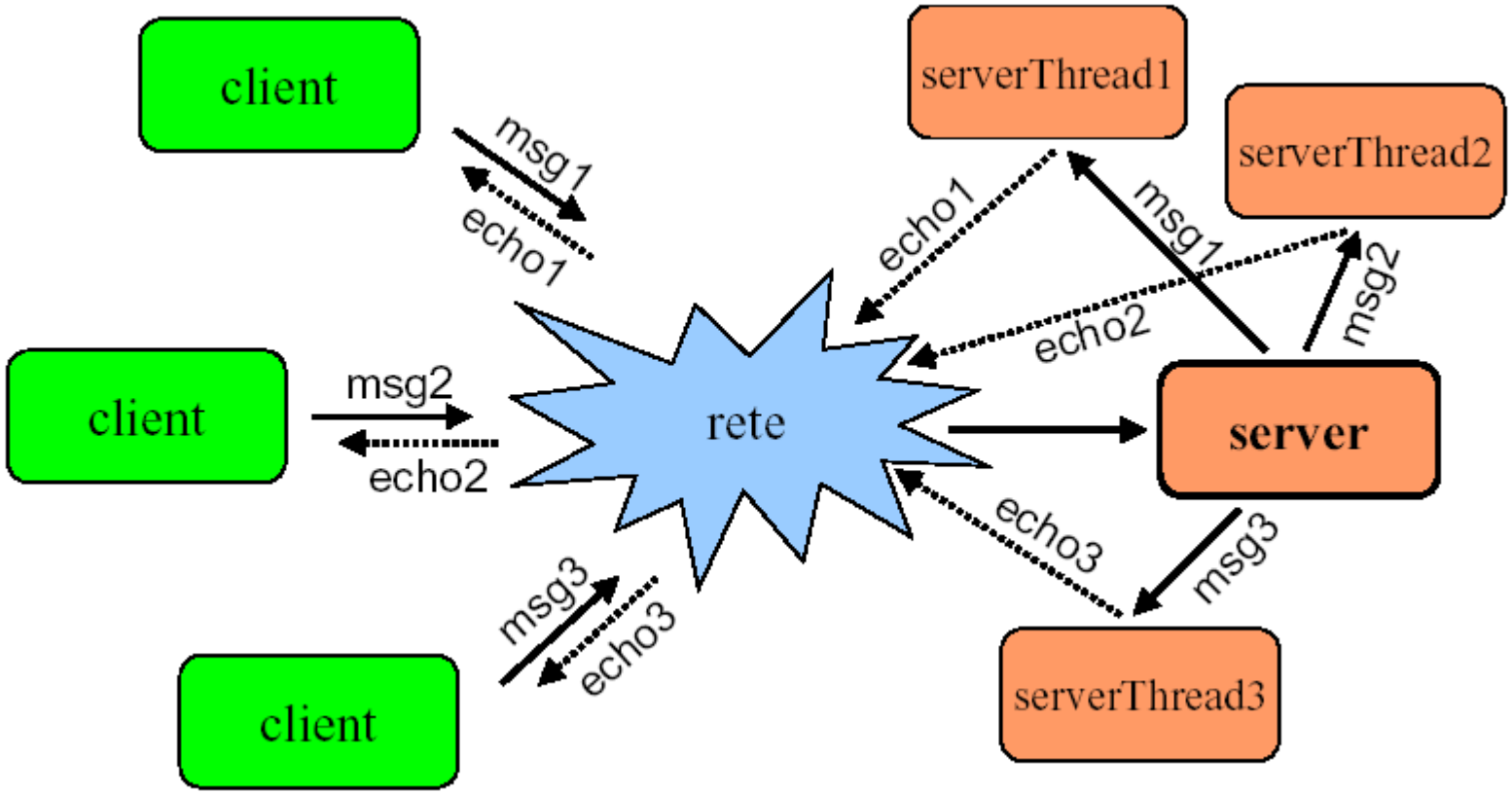
Echo Server (1)

```
public class EchoServer {
public static final int PORT = 1050; // porta al di fuori del range 1-1024 !

public static void main(String[] args) throws IOException {
  ServerSocket serverSocket = new ServerSocket(PORT);
  System.out.println("EchoServer: started ");
  System.out.println("Server Socket: " + serverSocket);
  Socket clientSocket=null;
  BufferedReader in=null;
  PrintWriter out=null;
  try {
   // bloccante finchè non avviene una connessione
   clientSocket = serverSocket.accept();
   System.out.println("Connection accepted: "+ clientSocket);
   // creazione stream di input da clientSocket
    InputStreamReader isr = new
InputStreamReader(clientSocket.getInputStream());
    in = new BufferedReader(isr);
   // creazione stream di output su clientSocket
    OutputStreamWriter osw = new
OutputStreamWriter(clientSocket.getOutputStream());
    BufferedWriter bw = new BufferedWriter(osw);
```

Echo Server (2)

```
//ciclo di ricezione dal client e invio di risposta
   while (true) {
     String str = in.readLine();
     if (str.equals("END")) break;
     System.out.println("Echoing: " + str);
     out.println(str);
   }
 }
 catch (IOException e) {
   System.err.println("Accept failed");
   System.exit(1);
 }
 // chiusura di stream e socket
 System.out.println("EchoServer: closing...");
 out.close();
 in.close();
 clientSocket.close();
 serverSocket.close();
 }
} // EchoServer
```

Multithreaded Echo Application



EchoMultiClient (1)

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

class ClientThread extends Thread {
  private Socket socket;
  private BufferedReader in;
  private PrintWriter out;
  private static int counter = 0;
  private int id = counter++;
  private static int threadcount = 0;

  public static int threadCount() {
    return threadcount;
  }
  public ClientThread(InetAddress addr) {
    threadcount++;
    try {
      socket = new Socket(addr, 1050);
      System.out.println("EchoClient n° "+id+": started");
      System.out.println("Client Socket: "+ socket);
    } catch(IOException e) {}
    // Se la creazione della socket fallisce non è necessario fare nulla
```

EchoMultiClient (2)

```
try {
      InputStreamReader isr = new
InputStreamReader(socket.getInputStream());
      in = new BufferedReader(isr);
      OutputStreamWriter osw = new
OutputStreamWriter(socket.getOutputStream());
      out = new PrintWriter(new BufferedWriter(osw), true);
      start();
    } catch(IOException e1) {
      // in seguito ad ogni fallimento la socket deve essere chiusa, altrimenti
    // verrà chiusa dal metodo run() del thread
      try {
        socket.close();
      } catch(IOException e2) {}
    }
  }
```

EchoMultiClient (3)

```
public void run() {
    try {
      for(int i =0;i <10; i++) {
        out.println("client "+id +" msg "+i);
        System.out.println("Msg sent: client "+id+" msg"+i);
        String str = in.readLine();
        System.out.println("Echo: "+str);
      }
      out.println("END");
     } catch(IOException e) {}
    try {
      System.out.println("Client "+id+" closing...");
      socket.close();
    } catch(IOException e) {}
    threadcount--;
  }

} // ClientThread
```

EchoMultiClient (4)

```
public class EchoMultiClient {
  static final int MAX_THREADS = 10;

  public static void main(String[] args) throws
IOException,InterruptedException {
    InetAddress addr;
    if (args.length == 0) addr = InetAddress.getByName(null);
    else addr = InetAddress.getByName(args[0]);

    while(true) {
      if (ClientThread.threadCount() < MAX_THREADS)
        new ClientThread(addr);
      Thread.sleep(1000);
    }
  }
} // EchoMultiClient
```

EchoMultiServer (1)

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

class ServerThread extends Thread {
  private static int counter = 0;
  private int id = ++counter;
  private Socket socket;
  private BufferedReader in;
  private PrintWriter out;

  public ServerThread(Socket s) throws IOException {
    socket = s;
    in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
    OutputStreamWriter osw = new
OutputStreamWriter(socket.getOutputStream());
    out = new PrintWriter(new BufferedWriter(osw), true);
    start();
    System.out.println("ServerThread "+id+": started");
    System.out.println("Socket: " + s);
  }
```

EchoMultiServer (2)

```
public void run() {
  try {
    while (true) {
      String str = in.readLine();
      if (str.equals("END")) break;
      System.out.println("ServerThread "+id+": echoing -> " + str);
      out.println(str);
    }
  System.out.println("ServerThread "+id+": closing...");
  } catch (IOException e) {}
  try {
    socket.close();
  } catch(IOException e) {}
 }
} // ServerThread
```

EchoMultiServer

```
public class EchoMultiServer {  (3)
public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket(1050);
    System.out.println("EchoMultiServer: started");
    try {
      while(true) {
        // bloccante finchè non avviene una connessione:
        Socket clientSocket = serverSocket.accept();
        System.out.println("Connection accepted: "+ clientSocket);
        try {
          new ServerThread(clientSocket);
        } catch(IOException e) { clientSocket.close(); }
      }
    }
    catch (IOException e) {
      System.err.println("Accept failed");
      System.exit(1);
    }
    System.out.println("EchoMultiServer: closing...");
    serverSocket.close();
  }
} // EchoMultiServer
```

Trasmissione di oggetti serializzati (1)

```
import java.io.*;
public class Studente implements Serializable {
  private int matricola;
  private String nome, cognome, corsoDiLaurea;
  public Studente (int matricola, String nome, String cognome,
                   String corsoDiLaurea) {
    this.matricola = matricola; this.nome = nome;
    this.cognome = cognome; this.corsoDiLaurea = corsoDiLaurea;
  }
  public int getMatricola () { return matricola; }
  public String getNome () { return nome; }
  public String getCognome () { return cognome; }
  public String getCorsoDiLaurea () { return corsoDiLaurea; }
}
```

Trasmissione di oggetti serializzati (2)

```
import java.io.*;
import java.net.*;
public class SendObject {
  public static void main (String args[]) {
    try {
      ServerSocket server = new ServerSocket (3575);
      Socket client = server.accept();
      ObjectOutputStream output =
        new ObjectOutputStream (client.getOutputStream ());
      output.writeObject("<Welcome>");
      Studente studente =new Studente (14520,"Leonardo","da Vinci","Ing.
Informatica");
      output.writeObject(studente);
      output.writeObject("<Goodbye>");
      output.close();
      client.close();
      server.close();
    } catch (Exception e) { System.err.println (e); }
  }
}
```

Trasmissione di oggetti serializzati (3)

```
import java.io.*;
import java.net.*;
public class ReceiveObject {
  public static void main (String args[]) {
    try {
      Socket socket = new Socket ("localhost",3575);
      ObjectInputStream input =
        new ObjectInputStream (socket.getInputStream ());
      String beginMessage = (String)input.readObject();
      System.out.println (beginMessage);
      Studente studente = (Studente)input.readObject();
      System.out.print (studente.getMatricola()+" - ");
      System.out.print (studente.getNome()+studente.getCognome());
      System.out.print (studente.getCorsoDiLaurea()+"\n");
      String endMessage = (String)input.readObject();
      System.out.println (endMessage);
      socket.close();
    } catch (Exception e) { System.err.println (e); }
  }
}
```

Datagrammi

Le applicazioni che comunicano tramite socket possiedono un canale di comunicazione dedicato. Per comunicare, un client ed un server stabiliscono una connessione, trasmettono dati, quindi chiudono la connessione. Tutti i dati inviati sul canale sono ricevuti nello stesso ordine in cui sono stati inviati.

Le applicazioni che comunicano tramite *datagrammi* inviano e ricevono pacchetti di informazione completamente indipendenti tra loro. Queste applicazioni non dispongono e non necessitano di un canale di comunicazione punto-a-punto. La consegna dei *datagrammi* alla loro destinazione non è garantita, e neppure l'ordine del loro arrivo.

Il package java.net fornisce tre classi che consentono di scrivere applicazioni che usano datagrammi per inviare e ricevere pacchetti sulla rete: ***DatagramSocket***, ***DatagramPacket*** e ***MulticastSocket***. Una applicazione può inviare e ricevere *DatagramPacket* tramite un *DatagramSocket*. Inoltre, i *DatagramPacket* possono essere inviati in broadcast a più destinatari che sono in ascolto su un *MulticastSocket*.

NOTA: MulticastSocket estende DatagramSocket

java.net.DatagramSocket

- **DatagramSocket (int port)**
  Crea un DatagramSocket e lo collega alla porta specificata sulla macchina locale.
- **void receive (DatagramPacket p)**
  Riceve un DatagramPacket da questo socket.
- **void send (DatagramPacket p)**
  Invia un DatagramPacket su questo socket.
- **void close ()**
  Chiude questo DatagramSocket.

java.net.DatagramPacket

- **DatagramPacket (byte[] buf, int length)**
  Crea un DatagramPacket per ricevere pacchetti di lunghezza length.

- **DatagramPacket (byte[] buf, int length,**
  InetAddress address, int port)
  Crea un DatagramPacket per inviare pacchetti di lunghezza length all'host ed alla porta specificati.

- **InetAddress getAddress ()**
  Restituisce l'indirizzo IP della macchina alla quale questo DatagramPacket deve essere inviato o da cui è stato ricevuto.

- **int getPort ()**
  Restituisce la porta della macchina alla quale questo DatagramPacket
  deve essere inviato o da cui è stato ricevuto.

TimeClient

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

public class TimeClient {
   public static void main(String[] args) throws IOException {
          String hostname = "localhost";
          DatagramSocket socket = new DatagramSocket();
          // invia la richiesta
          byte[] buf = new byte[256];
          InetAddress address = InetAddress.getByName(hostname);
          DatagramPacket packet =
             new DatagramPacket(buf, buf.length, address, 3575);
          socket.send(packet);
           // riceve la risposta
          packet = new DatagramPacket(buf, buf.length);
          socket.receive(packet);
         // visualizza la risposta
          String received = new String(packet.getData());
          System.out.println("Response: " + received);
          socket.close();
    }
}
```

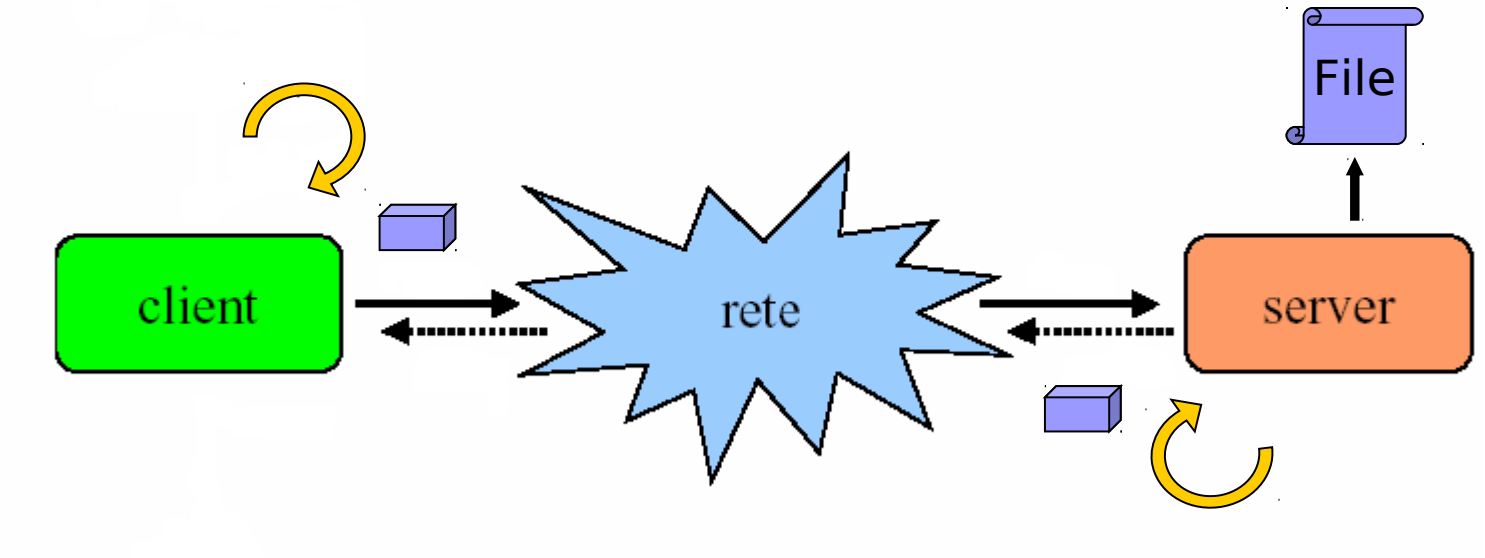
TimeServer (1)

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

public class TimeServer {

  public static void main(String[] args) {
    DatagramSocket socket = null;
    try {
      socket = new DatagramSocket(3575);
      int n = 1;
      while (n <= 10) {
        byte[] buf = new byte[256];
        // riceve la richiesta
        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
```

TimeServer (2)

```
        // produce la risposta
        String dString = new Date().toString();
        buf = dString.getBytes();
        // invia la risposta al client
        InetAddress address = packet.getAddress();
        int port = packet.getPort();
        packet = new DatagramPacket(buf, buf.length, address, port);
        socket.send(packet);
        n++;
      }
      socket.close();
    } catch (IOException e) {
      e.printStackTrace();
      socket.close();
    }
  }
}
```

Line application



- i client inviano al server pacchetti (vuoti) che vengono interpretati dal server come richiesta della linea corrente di un certo file di testo;

- il server invia a ciascun client un pacchetto contenente una linea (le linee vengono estratte in modo sequenziale) del file; se non riesce invia data e ora attuali

LineClient (1)

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

public class LineClient {
    public static void main(String[] args) throws IOException {
        // creazione della socket datagram con un timeout di 30s
        DatagramSocket socket = new DatagramSocket();
        socket.setSoTimeout(30000);
        System.out.println("LineClient: started");
        System.out.println("Socket: "+ socket);
        // creazione e invio del pacchetto di richiesta
        byte[] buf = new byte[256];
        InetAddress addr;
        if (args.length == 0) addr = InetAddress.getByName(null);
        else addr = InetAddress.getByName(args[0]);
        for (int i=0; i<200; i++){
           DatagramPacket packet = new DatagramPacket(buf, buf.length,
addr, 4444);
            socket.send(packet);
            System.out.println("Request sent to "+ addr);
```

LineClient (2)

```
        // pulizia del buffer
        for (int j=0; j<256; j++) buf[j]=' ';
        // ricezione e stampa della risposta
        packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
        // sospensiva solo per i millisecondi indicati, dopodichè solleva una SocketException
        String received = new String(packet.getData());
        System.out.println("Line got: " + received);
      }
      System.out.println("LineClient: closing...");
      socket.close();
    }
}
```

LineServer (1)

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

public class LineServer {
   public static final int PORT = 4444; // porta al di fuori del range 1-1024 !
   public static final String FILE = “file.txt";
   public DatagramSocket socket;
   public static BufferedReader in = null;
   public static boolean moreLines = true;
   public LineServer(){
     try {
          // creazione della socket datagram
          socket = new DatagramSocket(PORT);
          System.out.println("Socket: " + socket);
     } catch (SocketException e) {System.err.println("Unable to create the
socket");}
     try {
          // associazione di uno stream di input al file da cui estrarre le linee
          in = new BufferedReader(new FileReader(FILE));
          System.out.println("File "+FILE+" opened");
     } catch (FileNotFoundException e){System.err.println("Could not
```

LineServer (2)

```
public static void main(String[] args) throws IOException {
      System.out.println("LineServer: started");

      LineServer server=new LineServer();

      while (moreLines) {
          try {
              byte[] buf = new byte[256];
             // ricezione della richiesta
              DatagramPacket packet = new DatagramPacket(buf,
buf.length);
              server.socket.receive(packet);
             // preparazione della linea da inviare
            String dString = null;
            if (in == null) dString = new Date().toString();
            else dString = getNextLine();
```

LineServer (3)

```
if (dString!=null) {
            buf = dString.getBytes();
           // "impacchettamento" e invio della risposta
           InetAddress address = packet.getAddress();
           int port = packet.getPort();
           packet = new DatagramPacket(buf, buf.length, address,
port);
           server.socket.send(packet);
           System.out.println("Sending packet to "+address+",
"+port);
         }
       } catch (IOException e) {
            e.printStackTrace();
            moreLines = false;
         }
    }
     System.out.println("No more lines. Goodbye.");
     System.out.println("LineServer: closing...");
     server.socket.close();
  }
```

LineServer (4)

```
static String getNextLine() {
        String returnValue = null;
        try {
            if ((returnValue = in.readLine()) == null) {
                in.close();
                moreLines = false;
                returnValue = null;
                //returnValue = "No more lines. Goodbye.";
            }
        } catch (IOException e) {
            returnValue = "IOException occurred.";
        }
        return returnValue;
    }

}
```

MulticastSocket

Un Multicast Datagram socket è utilizzato per spedire/ricevere messaggi a/da destinatari/mittenti multipli.

Un MulticastSocket è un UDP DatagramSocket con funzionalità aggiuntive per aggregarsi e abbandonare “gruppi” di host.

Un gruppo multicast è identificato da un indirizzo IP di **classe D**. Gli indirizzi di classe D appartengono al range 224.0.0.0 – 239.255.255.255 inclusi.

Tutti i messaggi spediti ad un gruppo multicast verranno ricevuti da tutti i partecipanti al gruppo. Il mittente non deve essere necessariamente un appartenente al gruppo. Le operazioni di adesione e abbandono a/di un gruppo sono regolate dal protocollo **igmp**.

È possibile prenotare permanentemente indirizzi di gruppo multicast o assegnarli e utilizzarli temporaneamente quando necessario sulla propria rete. Per prenotare un indirizzo IP di gruppo permanente per l'utilizzo su Internet, è necessario registrarlo con Internet Assigned Numbers Authority (IANA).
http://www.iana.org/assignments/multicast-addresses

java.net.MulticastSocket

- **MulticastSocket (int port)**
  Crea un MulticastSocket e lo collega alla porta specificata sulla macchina locale.

- **void joinGroup (InetAddress mcastaddr)**
  Si aggiunge ad un multicast group.

- **void leaveGroup (InetAddress mcastaddr)**
  Abbandona un multicast group.

- **void receive (DatagramPacket p)**
  Riceve un DatagramPacket da questo socket.

- **void send (DatagramPacket p)**
  Invia un DatagramPacket su questo socket.

- **void close ()**
  Chiude il MulticastSocket.

MulticastTimeServer (1)

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

public class MulticastTimeServer {

  public static void main(String[] args) {
    MulticastSocket socket = null;
    try {
      socket = new MulticastSocket(3575);
      int n = 1;
      while (n <= 100) {
        byte[] buf = new byte[256];
        // non aspetta la richiesta
        String dString = new Date().toString();
        buf = dString.getBytes();
```

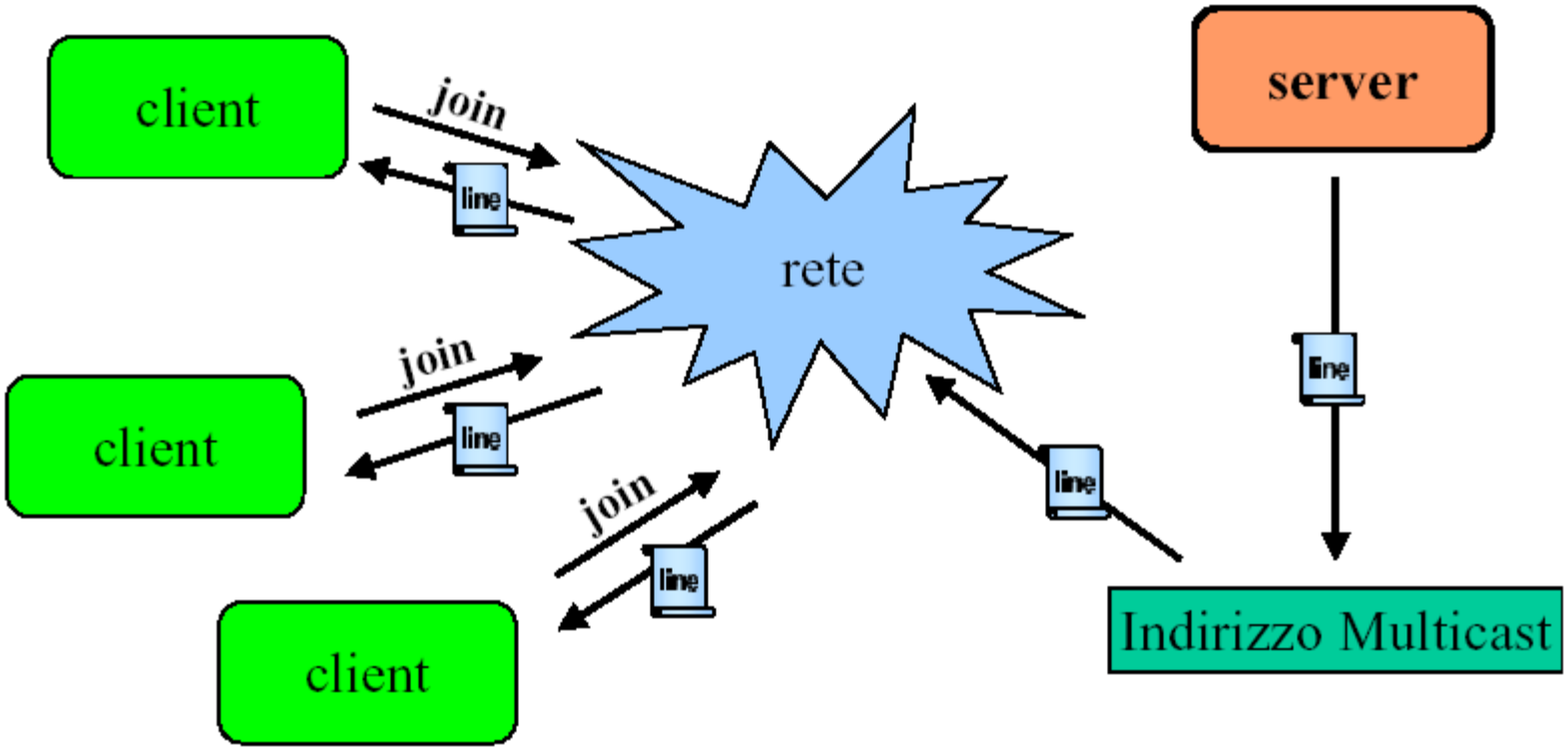
MulticastTimeServer (2)

```
    // invia il messaggio in broadcast
    InetAddress group = InetAddress.getByName("230.0.0.1");
        DatagramPacket  packet  =new  DatagramPacket(buf,  buf.length,
group, 3575);
    socket.send(packet);
    System.out.println ("Broadcasting: "+dString);
    Thread.sleep(1000);
     n++;
     }
     socket.close();
   }
   catch (Exception e) {
   e.printStackTrace();
   socket.close();
   }
  }
}
```

MulticastTimeClient

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

public class MulticastTimeClient {

  public static void main(String[] args) throws IOException {
    MulticastSocket socket = new MulticastSocket(3575);
    InetAddress group = InetAddress.getByName("230.0.0.1");
    socket.joinGroup(group);
    DatagramPacket packet;
    for (int i = 0; i < 100; i++) {
      byte[] buf = new byte[256];
      packet = new DatagramPacket(buf, buf.length);
      socket.receive(packet);
      String received = new String(packet.getData());
      System.out.println("Time: " + received);
    }
    socket.leaveGroup(group);
    socket.close();
  }
}
```

Multicast Line



- i client non inviano più nessun pacchetto di richiesta al server ma si associano al gruppo cui il server invia periodicamente le linee, ne ricevono alcune (per es. 5), le stampano, poi si dissociano dal gruppo;

- il server invia periodicamente (per es. ogni 5 secondi), una linea a tutti i client che si sono registrati all'indirizzo multicast

MulticastLineServer (1)

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

public class MulticastLineServer extends LineServer{
    Public static int OUTPUTPORT = 10000;

    public MulticastLineServer(){
      super();
    }

    public static void main(String[] args) throws IOException {

      long WAIT = 1000;
      int count=0;
      System.out.println("MulticastServer: started");

      MulticastServer server=new MulticastServer();
      // creazione del gruppo associato all'indirizzo multicast
      InetAddress group = InetAddress.getByName("230.0.0.1");
      System.out.println("Creating group "+ group);
```

MulticastLineServer (2)

```
while (moreLines) {
    try {
        count++;
        byte[] buf = new byte[256];
        String dString = null;
        if (in == null) dString = new Date().toString();
        else dString = getNextLine();
        if (dString!=null) {
           buf = dString.getBytes();
                DatagramPacket packet = new DatagramPacket(buf, buf.length,
group, OUTPUTPORT);
           server.socket.send(packet);
           System.out.println("Sending line # "+ count);
        }
        try {
          Thread.sleep((long)(Math.random() * WAIT));
        } catch (InterruptedException e) { }
    } catch (IOException e) {moreLines = false;}
  }
  System.out.println("No more lines. Goodbye.“+"MulticastServer: closing...");
  server.socket.close();
 }
}
```

MulticastLineClient (1)

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


public class MulticastClient {
    public static void main(String[] args) throws IOException {

        // creazione della socket multicast
                                        MulticastSocket socket = new
MulticastSocket(MulticastServer.OUTPUTPORT);
        socket.setSoTimeout(20000);
        InetAddress address = InetAddress.getByName("230.0.0.1");
        System.out.println("MulticastClient: started");
        System.out.println("Socket: "+ socket);
        // adesione al gruppo associato all'indirizzo multicast
        socket.joinGroup(address);
        System.out.println("Joining to "+ address);
```

MulticastLineClient (2)

```
DatagramPacket packet;

        // ricezione di alcune linee
        for (int i = 0; i < 5; i++) {
            byte[] buf = new byte[256];
            packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String received = new String(packet.getData());
            System.out.println("Line got: " + received);
        }
        // uscita dal gruppo e chiusura della socket
        System.out.println("Leaving group...");
        socket.leaveGroup(address);
        System.out.println("MulticastClient: closing...");
        socket.close();
    }
}
```