Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2018-2019

Pietro Frasca

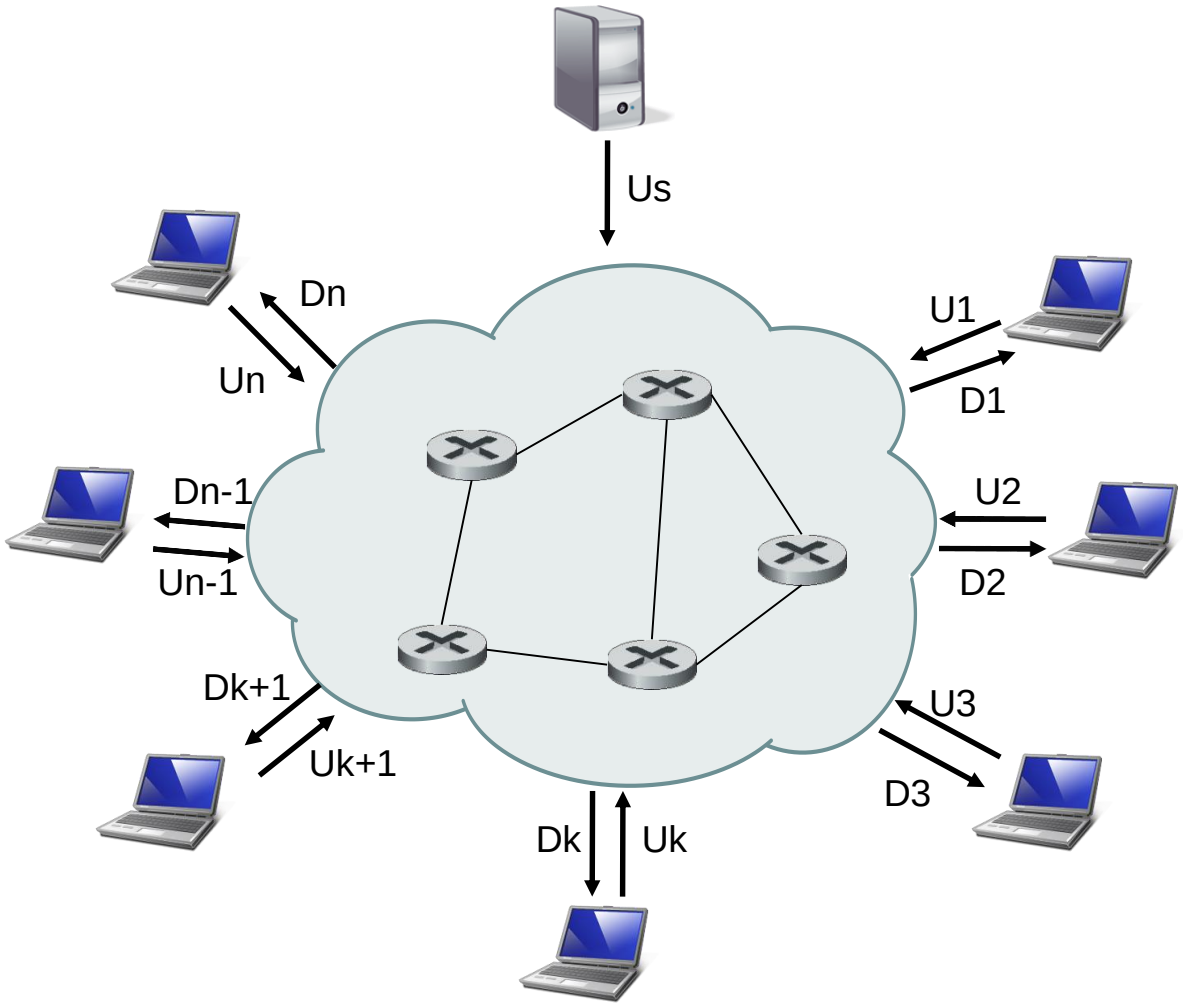
Parte II: Reti di calcolatori

Lezione 12 (36)

Giovedì 11-04-2019

Confronto architetture C/S e P2P .

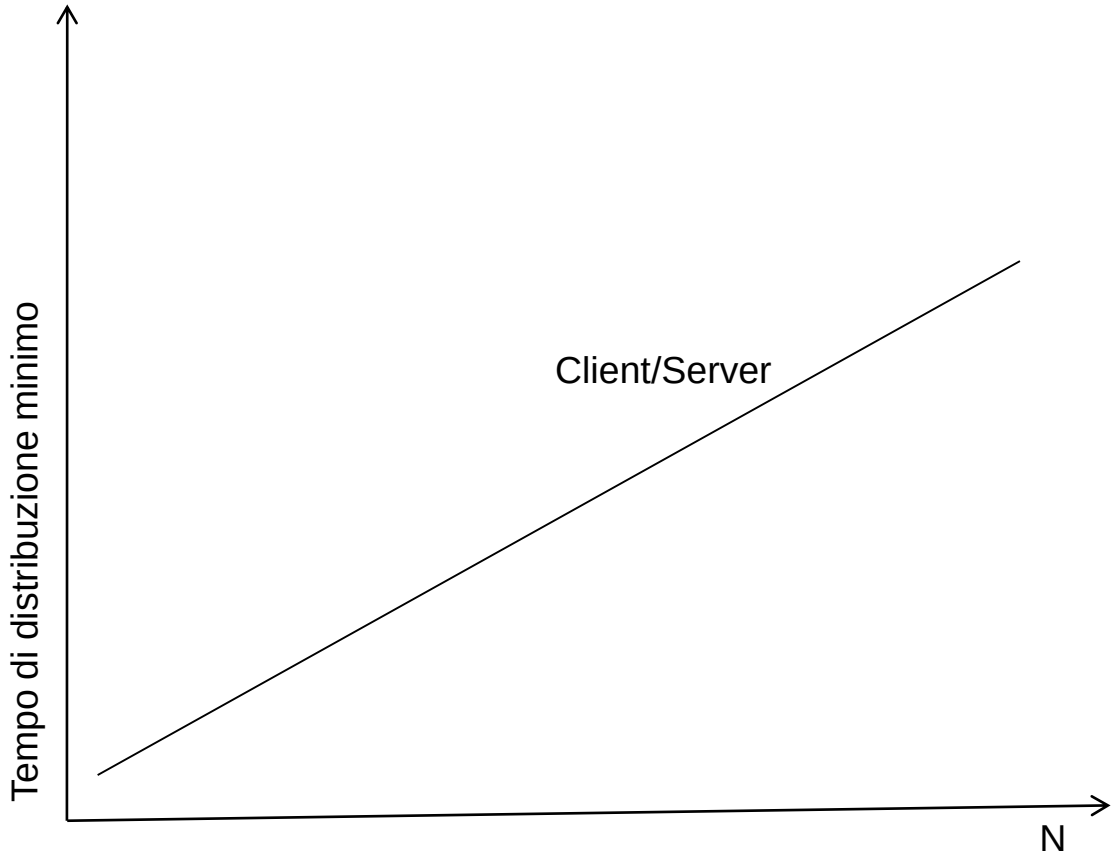
- Per confrontare le architetture client/server e P2P, faremo un calcolo semplificato del tempo necessario per distribuire un file di **F** bit (inclusi i bit di controllo dei protocolli) a un insieme **N** di client (o pari), per entrambe le architetture.
- Il tempo di distribuzione è il tempo necessario affinché tutti gli *N client (pari)* ottengano una copia del file.
- Supponiamo che server e client (o pari) siano connessi a Internet con tecnologie a larghezza di banda alquanto inferiore rispetto alle larghezze di banda dei collegamenti del resto della rete.
- Indichiamo con ***Us*** la velocità di upload del collegamento di accesso a Internet del server, con ***Ui e Di*** le velocità di upload e di download del collegamento di accesso dello *i-esimo* pari.


Us
Dn
Un
U1
D1
Dn-1
Un-1
U2
D2
Dk+1
Uk+1
U3
D3
Dk
Uk

- Calcoliamo prima il tempo di distribuzione $\mathbf{TD_{cs}}$ del file per l'architettura client/server. Facciamo le seguenti considerazioni:
  - Il server deve trasmettere una copia del file a ciascuno degli **N** client, cioè **N·F** *bit.* Dato che la velocità di upload del server è $\mathbf{U_s}$ il tempo per distribuire il file deve essere almeno $\mathbf{N \cdot F/U_s}$.
  - Indichiamo con $\mathbf{D_{min} = min\{D1,D2,..,Dn\}}$ la velocità di download del client con valore minimo. Il client con la velocità di download più bassa può ricevere il file in almeno $\mathbf{F/D_{min}}$ secondi. Quindi il tempo minimo di distribuzione è $\mathbf{F/D_{min}}$.
- Considerando queste due relazioni, otteniamo che:

$$\mathbf{TD_{cs} = max\{N \cdot F/U_s, F/D_{min}\}}$$

- Vediamo che la relazione dipende linearmente da *N*, e quando N assume valori grandi, il tempo di distribuzione client/server è dato da $\mathbf{N \cdot F/U_s}$.


Tempo di distribuzione minimo
Client/Server
N

- Calcoliamo ora il tempo di distribuzione per l'architettura P2P, tenendo presente che quando un pari riceve il file F, può rinviare parti dello stesso agli altri pari che lo richiedono, contribuendo così alla distribuzione del file.
- Il calcolo del tempo di distribuzione per un'architettura P2P è piuttosto complicato poiché il tempo di distribuzione dipende, oltre che dal numero N di pari, anche dalle dimensioni delle parti che ciascun pari distribuisce agli altri pari.
- Tuttavia, possiamo calcolare una semplice espressione approssimata del tempo minimo di distribuzione. Al riguardo, facciamo le seguenti osservazioni:
  1. All'inizio della distribuzione solo il server possiede il file. Per trasmetterlo agli altri pari, il server di *origine* deve inviare ciascuna parte del file almeno una volta. Quindi, il tempo minimo di distribuzione è almeno:

     $\mathbf{F/U_s}$.

Diversamente dal modello client/server, una parte del file inviata una volta dal server può non dover essere inviata di nuovo, in quanto i pari possono rinviarsela tra loro.

2. Come per l'architettura client/server, il pari con velocità di download più bassa non può ottenere il file completo in meno di $\mathbf{F/D_{min}}$ secondi.

3. Infine, osserviamo che la velocità totale di upload del sistema P2P nel suo complesso è uguale alla velocità di upload del server di origine (U_s) più quella degli altri pari,

$$\mathbf{U_{tot} = U_s + U_1 + U_2 + \ldots + U_n}$$

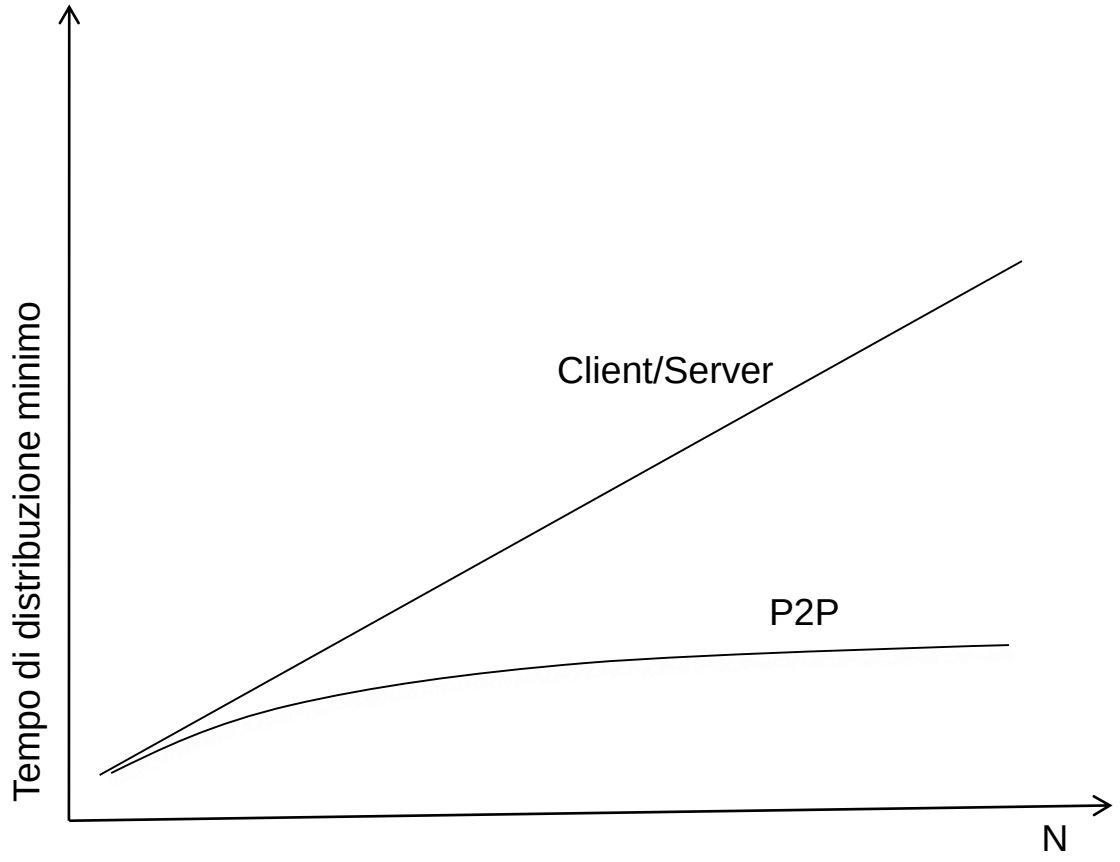
cioè il sistema P2P deve trasmettere **F** bit a ciascuno degli **N** pari, inviando quindi un totale di **F·N** *bit.* Questo non può essere fatto a una velocità superiore a $\mathbf{U_{tot}}$. Quindi, il tempo di distribuzione minimo è almeno

$$\mathbf{N \cdot F/(U_s + U_1 + U_2 + \ldots + U_n).}$$

- Mettendo insieme queste tre relazioni otteniamo il **tempo di distribuzione minimo** $\mathbf{TD_{P2P}}$ per l'architettura P2P:

  $$\mathbf{TD_{P2P} = \max\{F/U_s,\ F/D_{min},\ N \cdot F/(U_s + \sum U_i)\}}$$

  Questa espressione fornisce il valore minimo del tempo di distribuzione per l'architettura **peer-to-peer**, nell'ipotesi che ciascun pari possa rinviare un bit appena lo riceve. In realtà, vengono rinviate ampie parti del file (ad esempio di 256 KB). L'espressione costituisce una buona approssimazione dell'effettivo tempo minimo di distribuzione.

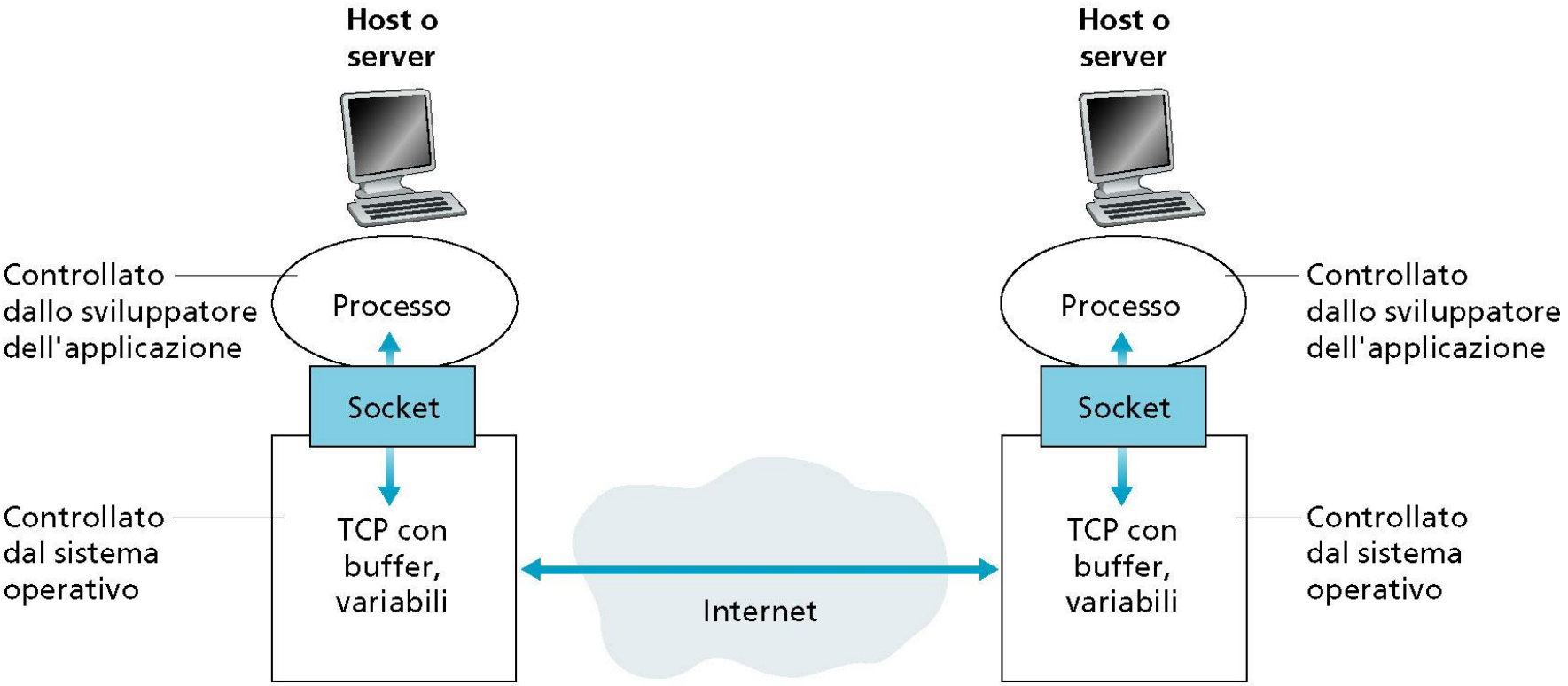

Tempo di distribuzione minimo
Client/Server
P2P
N

Programmazione delle socket

- Un applicazione di rete con modello client/server consiste di due programmi: un programma client e uno server. Quando i due programmi sono avviati si creano un processo client e uno server, che comunicano tra loro mediante le socket.
- Le socket costituiscono l'interfaccia fra il processo applicativo e i protocolli dello strato di trasporto (TCP o UDP).
- Le socket hanno un ruolo fondamentale nelle applicazioni di rete, e pertanto la realizzazione di applicazioni di rete viene spesso indicata con il termine **programmazione delle socket** *(**socket programming**).*
- Durante la fase di progetto, una delle prime scelte da fare riguarda il protocollo di trasporto da utilizzare: TCP o UDP. Ricordiamo che il TCP è orientato alla connessione e fornisce un *trasferimento affidabile*. L'UDP è senza connessione e non garantisce la consegna di tutti i dati trasmessi. Inoltre, si deve fare attenzione a non usare uno dei numeri di porta riservati *(well-known port number)* definiti nelle RFC.

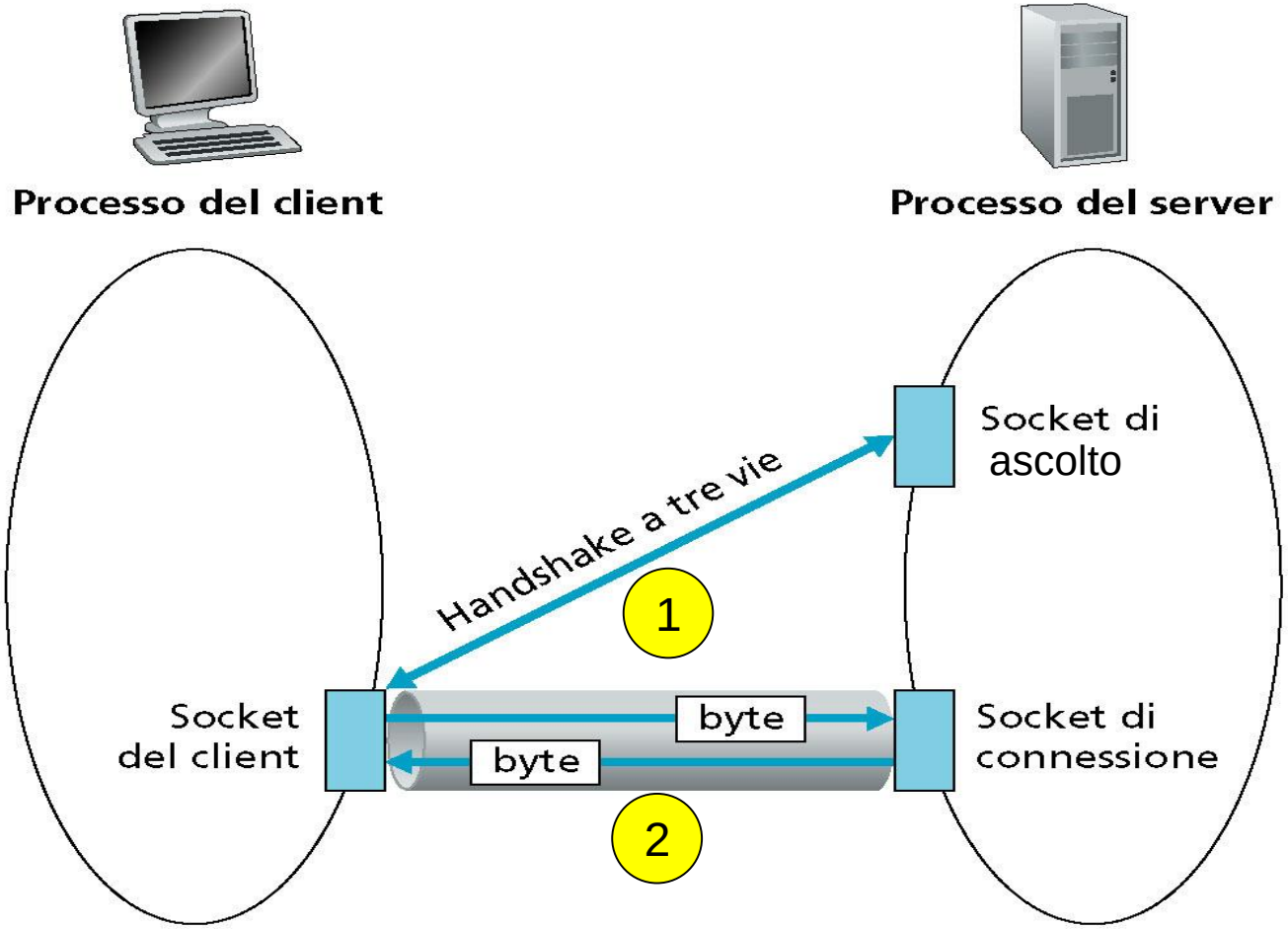
Programmazione delle socket con TCP

- Processi che girano su macchine diverse comunicano tra loro inviando (ricevendo) messaggi nelle (dalle) socket.



Processi comunicanti attraverso socket TCP

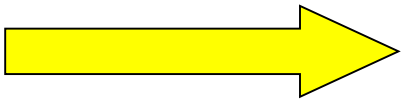
- Il client ha il compito di iniziare la comunicazione con il server.
- Affinché il server possa rispondere, deve essere pronto. Questo implica che:
  - il programma server deve essere avviato prima che il client tenti di stabilire la connessione.
  - il programma server deve usare un numero di porta che non sia già usato da altre applicazioni attive (che usano il TCP), per ricevere le richieste di connessione iniziali di un client.



Socket del client, socket di ascolto e socket di connessione.

- Dopo aver avviato il processo server, il processo client può instaurare una connessione TCP col server. Tale connessione si realizza nel programma client creando un **oggetto socket**.
- Il client crea il suo oggetto **socket** specificando, tramite il costruttore, **l'indirizzo IP** del server e il **numero di porta** del processo server.
- Dopo la creazione di un oggetto socket, il TCP nel client esegue la fase di handshake a tre vie con il server per stabilire una connessione TCP. La procedura di handshake, essendo eseguita dal TCP, è trasparente per l'applicazione.
- Per l'applicazione, la connessione TCP è come un canale logico che collega la socket del client alla **socket di connessione** del server.
- Instaurata la connessione, sia il client che il server possono inviare e ricevere dati attraverso le rispettive socket di connessione.

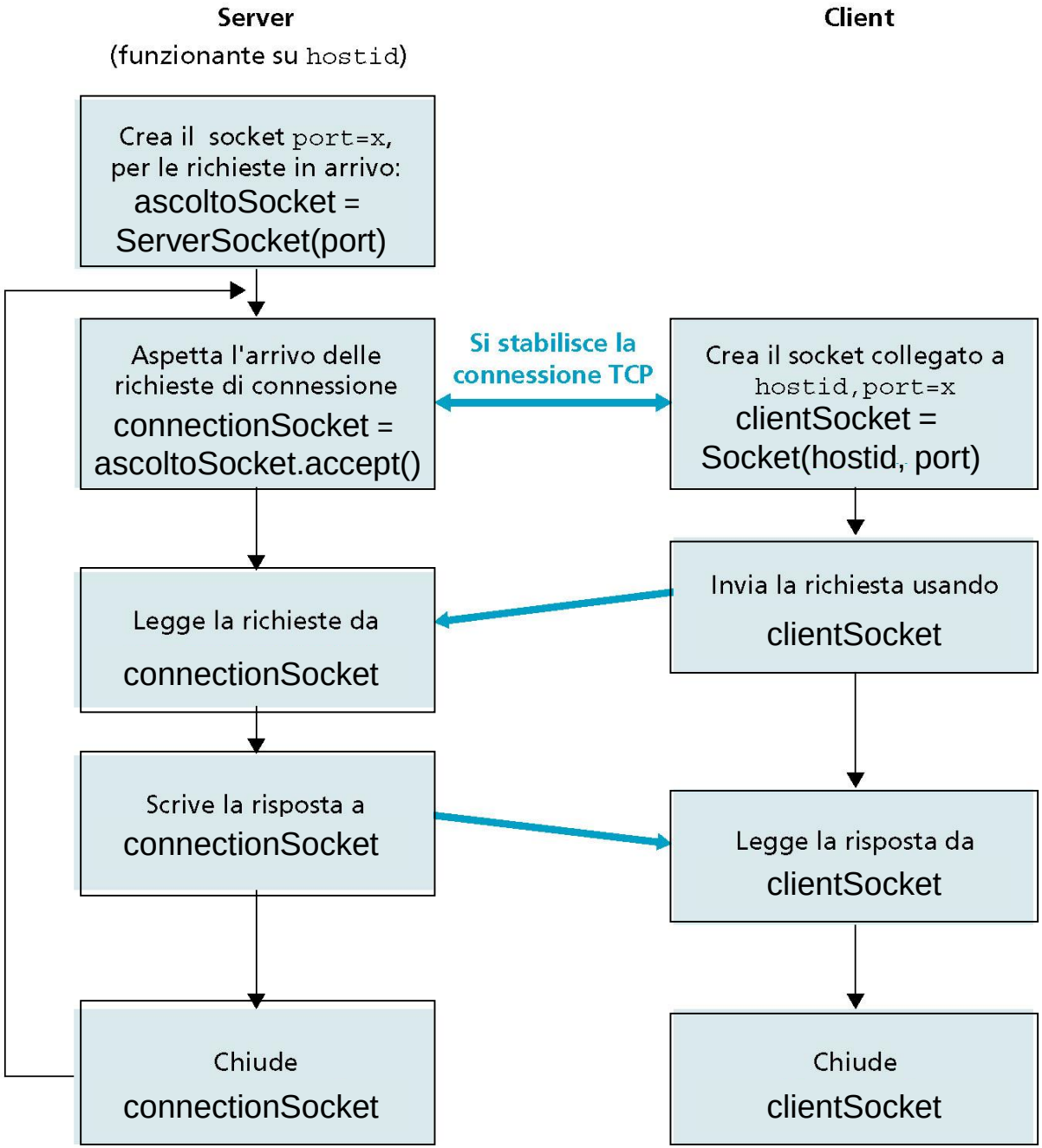
- Prima di descrivere l'applicazione client/server, è necessario introdurre il concetto di stream (flusso).
- Uno stream è una sequenza di caratteri che è trasmessa verso l'interno o verso l'esterno di un processo. Uno stream può essere di ingresso *(**input stream**)* o di uscita *(**output stream**)* per il processo.
- Uno stream di ingresso è collegato a sorgenti d'ingresso per il processo, come ad esempio l'ingresso standard (la tastiera), un file su disco o a una socket nella quale i dati provengono dalla rete. Uno stream di uscita è collegato a sorgenti di uscita del processo, come ad esempio l'uscita standard (il monitor), un file su disco o una socket attraverso la quale i dati sono trasmessi sulla rete.

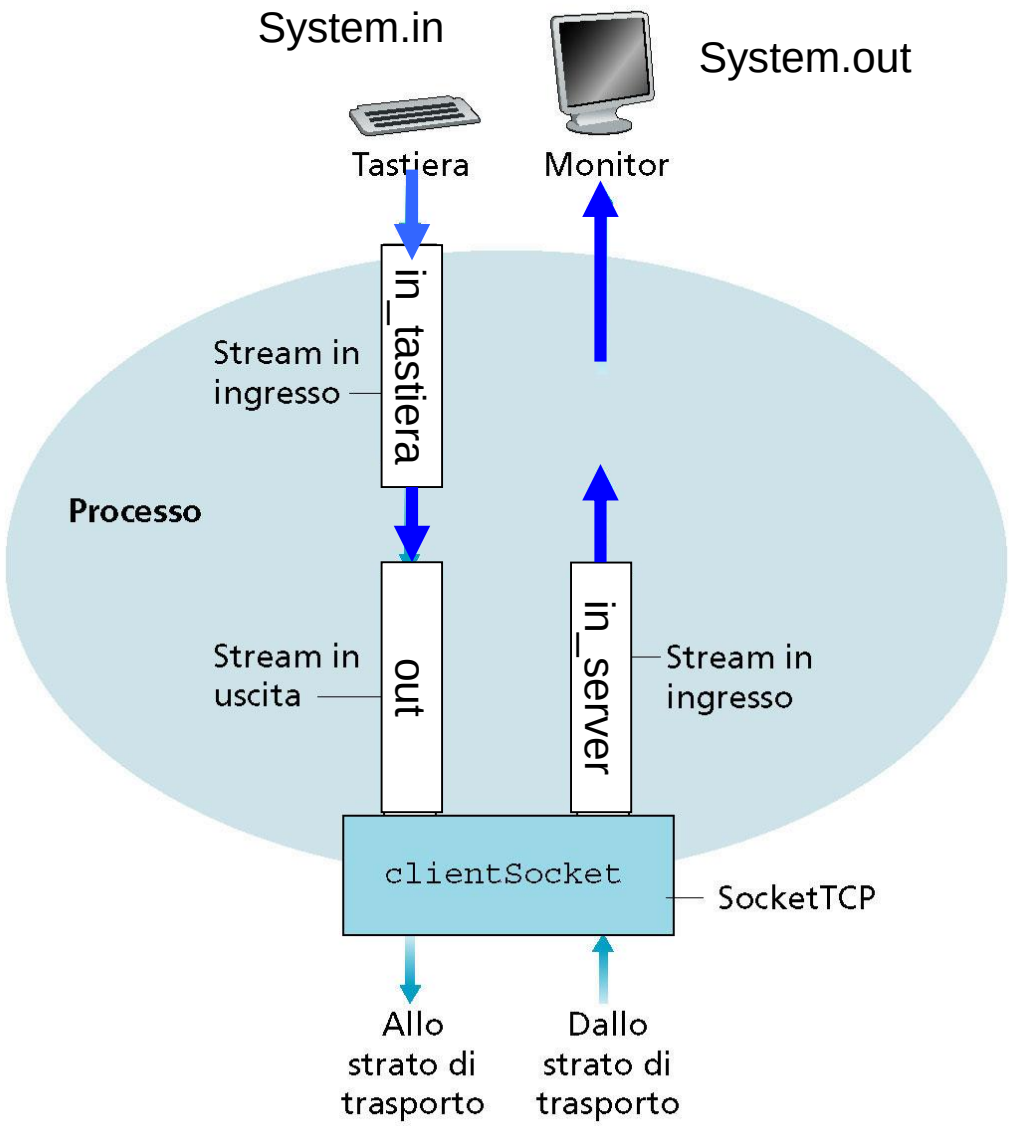


STREAM IN JAVA

Un esempio di applicazione client server in Java

- Useremo una semplice applicazione client/server per mostrare la programmazione delle socket sia per TCP sia per UDP:
  - Il client legge una frase da tastiera (standard input) scritta in lettere minuscole e la invia al server mediante la sua socket di connessione.
  - Il server legge i dati dalla sua socket di connessione.
  - Il server converte la frase in lettere maiuscole.
  - Il server invia al client la frase modificata, mediante la sua socket di connessione.
  - Il client legge la frase modificata dalla sua socket e la visualizza sul monitor.
- La figura seguente mostra uno schema di interazione tra client e server.


Server
(funzionante su hostid)
Client
Crea il socket port=x, per le richieste in arrivo:
ascoltoSocket = ServerSocket(port)
Aspetta l'arrivo delle richieste di connessione
connectionSocket = ascoltoSocket.accept()
Si stabilisce la connessione TCP
Crea il socket collegato a hostid,port=x
clientSocket = Socket(hostid, port)
Legge la richieste da
connectionSocket
Invia la richiesta usando
clientSocket
Scrive la risposta a
connectionSocket
Legge la risposta da
clientSocket
Chiude
connectionSocket
Chiude
clientSocket


System.in
System.out
Tastiera
Monitor
in_tastiera
Stream in ingresso
Processo
Stream in uscita
out
in_server
Stream in ingresso
clientSocket
SocketTCP
Allo strato di trasporto
Dallo strato di trasporto

Struttura delle applicazioni di esempio

```
class TCPClient {
   public static void main(String argv[]) throws Exception
   {
       // corpo del main
   }
}

class TCPClient {
   public static void main(String argv[])
   {
       try {
              // codice che potrebbe sollevare eccezioni
       } catch (Exception ex){
         // gestione delle eccezioni
       }


   }
}
```

```
1.    import java.io.*;
2.    import java.net.*;
3.    class TCPClient {
4.      public static void main(String arg[]) throws Exception{
5.        String nomeServer="localhost"; // indirizzo IP o nome del server
6.        int porta=1234; //porta di ascolto del server
7.        String richiesta="."; //messaggio di richiesta inviato dal client
8.        String risposta; //messaggio di risposta ricevuto dal server
9.        BufferedReader in_tastiera=new BufferedReader(new
        InputStreamReader(System.in));
10.       Socket clientSocket = new Socket(nomeServer, porta);
11.       System.out.println("porta remota “ + clientSocket.getPort());
12.       System.out.println("porta locale “ + clientSocket.getLocalPort());
13.       DataOutputStream out = new
        DataOutputStream(clientSocket.getOutputStream());
14.       BufferedReader in_server=new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));
15.       while (! richiesta.equals(“exit")){
16.         System.out.print("Scrivi una frase: ");
17.         richiesta = in_tastiera.readLine(); // metodo bloccante
18.         out.writeBytes(richiesta+"\n");
19.         risposta = in_server.readLine(); //metodo bloccante
20.         System.out.println("Risposta dal server: “ + risposta);
21.       }
22.       clientSocket.close()
23.     }
24.   }
```

- La socket per la connessione è clientSocket.
- Lo stream **in_tastiera** è uno stream di ingresso per il processo ed è collegato alla tastiera (input standard).
- Lo stream **in_server** è uno stream di ingresso al processo collegato alla socket che consente di prelevare i dati che arrivano dalla rete.
- Infine, lo stream **out** è uno stream di uscita dal processo collegato alla socket.
- Vediamo ora le varie linee del codice.

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

  java.io e **java.net** sono package Java.

  Il package **java.io** contiene le classi per gli stream di ingresso e di uscita.

  Il package **java.net** fornisce le classi per la rete. In particolare, contiene le classi **Socket** e **ServerSocket**.
  ```

La stringa **richiesta** conterrà i caratteri digitati da tastiera attraverso lo stream **in_tastiera**.

- ```
  String richiesta;
  String risposta;
  ```

  La variabile **richiesta** conterrà la frase che l'utente digiterà sulla tastiera; **risposta** è la stringa che il client riceverà dal server.

- ```
  BufferedReader in_tastiera = new BufferedReader(new
  InputStreamReader(System.in));
  ```

  crea l'oggetto stream **in_tastiera** della classe **BufferedReader**. Al costruttore di InputStreamReader è passato System.in, che collega lo stream alla tastiera (ingresso standard). A sua volta un oggetto della classe InputStreamReader è passato al costruttore della classe BufferedReader, realizzando in tal modo un flusso

bufferizzato tra tastiera e processo.

- `Socket clientSocket = new Socket("hostname",1234);`

  crea un oggetto clientSocket della classe Socket che inizia la connessione TCP tra client e server. La stringa "nomeServer" deve essere sostituita con l'hostname o dal numero IP del server (per esempio, "reti.uniroma2.it" oppure 160.80.1.16).

  Nel caso si utilizzasse l'hostname, prima che la connessione TCP inizi realmente, il client esegue una ricerca DNS sull'hostname per ottenere l'indirizzo IP dell'host.
- Il numero 1234 è il numero della porta che deve essere lo stesso usato dal lato server dell'applicazione (porta di ascolto).
- Come già detto, l'indirizzo IP dell'host e il numero di porta identificano il processo del server.

- `DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());`

  `BufferedReader in_server = new BufferedReader(new inputStreamReader(clientSocket.getInputStream()));`

  sono oggetti stream che sono collegati alla socket:
  - lo stream **out** è l'uscita del processo verso la socket.
  - lo stream **in_server** è l'input dalla socket verso il processo.

- `richiesta = in_tastiera.readLine();`

- `out.writeBytes(richiesta + '\n');`

  invia la stringa **richiesta** allo stream **out**. La stringa passa attraverso la socket TCP del client. Il client aspetta quindi un messaggio di risposta dal server.

- `risposta = in_server.readLine();`

  La sequenza di caratteri che arriva dal server, attraverso lo stream **in_server** viene assegnata alla stringa **risposta**, finché la linea termina con un ritorno carrello (‘\n’).

- `System.out.println(“Risposta dal server: " + risposta);`

  stampa sul monitor la stringa **risposta** inviata dal server.

- `clientSocket.close();`

  chiude la socket di connessione e, quindi, la connessione TCP tra client e server.

```
1. import java.io.*;
2. import java.net.*;
3. class TCPServer {
4.   public static void main(String arg[]) throws Exception {
5.     String richiesta, risposta;
6.     int porta = 1234; // porta di ascolto
7.     ServerSocket ascoltoSocket = new ServerSocket(porta);
8.     System.out.println("Server in ascolto sulla porta "+porta);
9.     Socket connSocket=ascoltoSocket.accept();//metodo bloccante
10.    BufferedReader in_client = new BufferedReader(new
          InputStreamReader(connSocket.getInputStream()));
11.    DataOutputStream out = new
          DataOutputStream(connSocket.getOutputStream());

12.    while (! richiesta.equals("exit")) {
         richiesta = in_client.readLine(); //metodo bloccante
13.      risposta = richiesta.toUpperCase()+"\n";
14.      out.writeBytes(risposta);
15.    }
16.  }
17.}
```

- Vediamo ora il codice di TCPServer.java.

- `ServerSocket ascoltoSocket=new ServerSocket(1234);`

  crea l'oggetto **ascoltoSocket** della classe **ServerSocket** che il server usa per ricevere le richieste dei client. L'indirizzo IP del server e la porta numero 1234 identificano il processo server.

- `Socket connSocket = ascoltoSocket.accept();`

  il metodo **accept** della classe **ServerSocket** è bloccante; crea una nuova socket, **connSocket**, quando un messaggio di richiesta di un client giunge ad **ascoltoSocket**. la nuova socket creata, connSocket ha lo stesso numero di porta di ascoltoSocket.

- Il TCP crea allora un canale logico punto-punto tra clientSocket nel lato client e connSocket nel lato server. Da questo momento, client e server si possono scambiare dati attraverso il canale logico.
- Il programma ora crea oggetti stream, analogamente agli oggetti stream creati in clientSocket.

- `risposta = richiesta.toUpperCase() + '\n';`

  converte la stringa **richiesta** in maiuscolo.