Esercizio UDP e Shell Remote

Laboratorio di Programmazione di Rete A

Esercitazione di Laboratorio

24/10/2007

Esercizio su UDP

- Scrivere un'applicazione composta da un processo Sender, il client, ed un processo Receiver, il server, in esecuzione su host diversi.
- Il client riceve da linea di comando una stringa, l'hostname e la porta di ascolto del server e invia la stringa al server tramite protocollo UDP. Il server rimane in ascolto su una porta UDP (nota al client), riceve il messaggio e lo visualizza.

Il Client: il File ClientUDP.java

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

public class ClientUDP
{
   public static void main(String[] args)
   {
      String sendString = "";
      String hostname = "";
      int port;

      if(args.length > 2)
      {
         sendString = args[0];
         hostname = args[1];
         port = Integer.parseInt(args[2]);
      }
```

Il Client: il File ClientUDP.java

```
    else
    {System.out.println("java ClientUDP string hostname port");
     return;}
    try
    {
       System.out.println("Inizio client");
       InetAddress server = InetAddress.getByName(hostname);
       DatagramSocket socket = new DatagramSocket();
       byte[] data = sendString.getBytes();
       DatagramPacket packet = new DatagramPacket(data,
                               data.length, server, port);
       socket.send(packet);
       System.out.println("Inviato:" + sendString +
                     " al server " + server + ":" + port);
    }catch(UnknownHostException ex){System.out.println(ex);}
    catch(SocketException ex){System.out.println(ex);}
    catch(IOException ex){System.out.println(ex);}
    System.out.println("Fine client");
}}
```

Il Client: il File ClientUDP.java

Osservazioni:

- `DatagramSocket socket = new DatagramSocket()`: crea un socket su una porta anonima (una porta libera). Si utilizza lato client, poiche' non interessa assegnare il socket ad una porta prefissata.
  - il sistema operativo restituira' una porta non assegnata.
  - il numero di porta e' inserito in ogni datagramma inviato: il server inviera' le risposte a questa porta.
- `byte[] data = sendString.getBytes()`: codifica la stringa `sendString` in un array di byte.

Il Client: il File ClientUDP.java

- `DatagramPacket packet = new DatagramPacket(data, data.length, server, port)`: costruttore per creare un datagramma UDP lato client. Specifica anche il server e la porta di destinazione
  - il socket UDP non e' connesso a nessun server, per cui e' necessario specificare la destinazione nel `DatagramPacket`.
- `socket.send(packet)`: invia il datagramma al server/porta specificati nel `DatagramPacket` tramite il socket.

Il Server: il File ServerUDP.java

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

public class ServerUDP
{
   public static void main(String[] args)
   {
      final int MAX_SIZE = 65507;
      int port;
      if(args.length > 0)
      {
         port = Integer.parseInt(args[0]);
      }
      else
      {System.out.println("java ServerUDP port");
       return;
      }
}
```

Il Server: il File ServerUDP.java

```
System.out.println("Inizio Server, porta in ascolto:" + port);
try{ DatagramSocket server = new DatagramSocket(port);
     byte[] buffer = new byte[MAX_SIZE];
     DatagramPacket packet = new DatagramPacket(buffer,
                                        buffer.length);
     while(true)
     {  try{
          server.receive(packet);
          String s = new String(packet.getData(), 0,
                                packet.getLength());
          System.out.println("Ricevuto da client " +
                          packet.getAddress() + ":" +
                         packet.getPort()+ ": " + s);
          packet.setLength(buffer.length);
          }catch(IOException e){System.out.println(e);}
     }
  }catch(SocketException e)System.out.println(e);
  System.out.println("Fine Server");
}}
```

Il Server: il File ServerUDP.java

Osservazioni:

- `DatagramSocket server = new DatagramSocket(port)`: crea un socket in ascolto sulla porta indicata (la porta deve essere nota ai client!!).
- `byte[] buffer = new byte[MAX_SIZE]`: quando il socket ricevera' un datagramma, salvera' i dati ricevuti nel buffer allocato.
  - La dimensione massima di un datagramma UDP e' di 65535 bytes (infatti, il campo *datagram length* dell'header IP e' di 2 bytes) meno 8 bytes (dimensione header UDP) meno 20 bytes (dimensione minima header IP), cioe' 65507 bytes. La dimensione suggerita e' di 8Kbytes (8192 bytes).

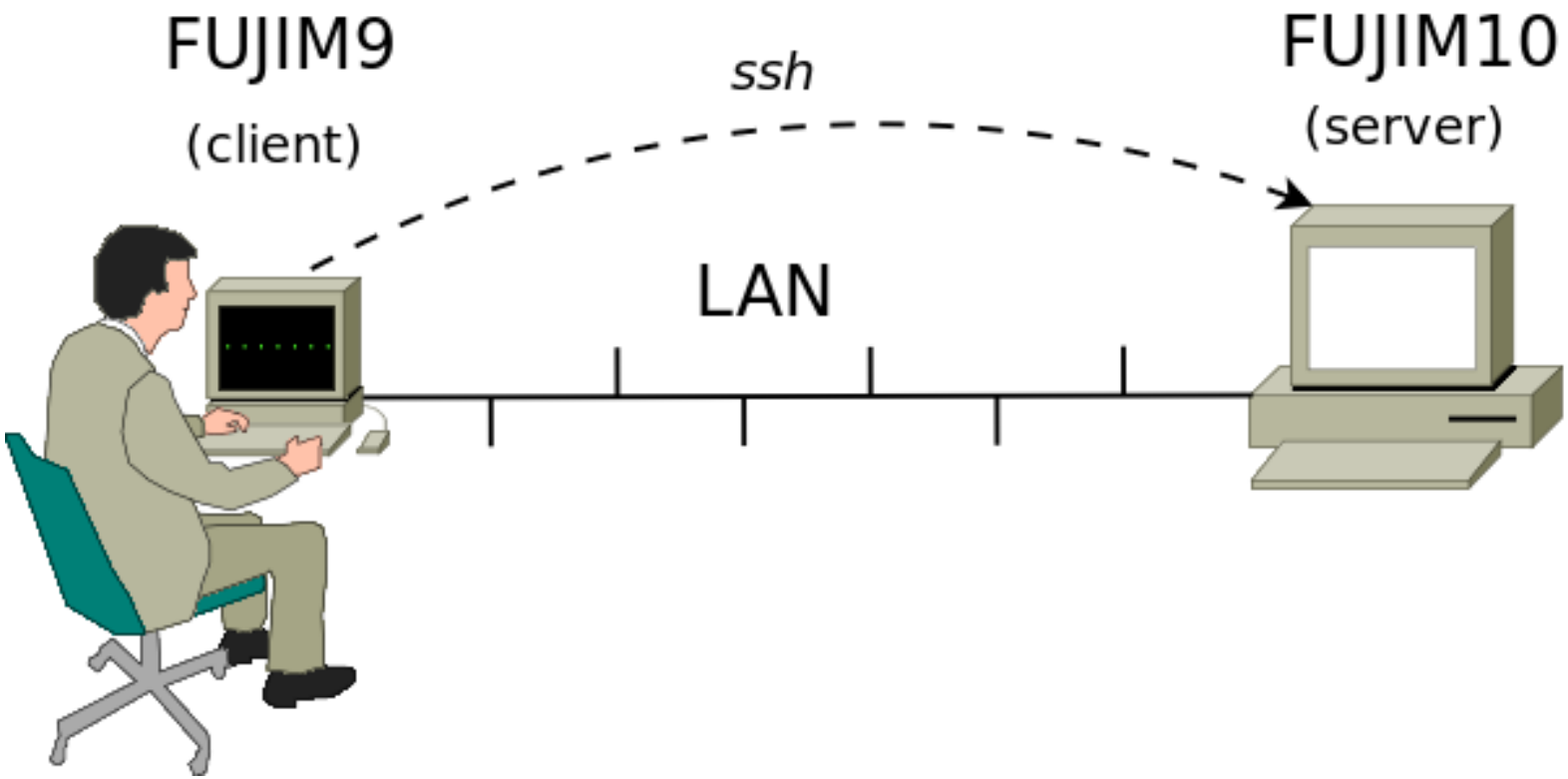
Il Server: il File ServerUDP.java

- `DatagramPacket packet = new DatagramPacket(buffer, buffer.length)`: creato per contenere il datagramma ricevuto dal socket. I dati sono salvati nel buffer a partire dalla posizione 0 fino ad aver ricevuto tutto il datagramma o alla massima lunghezza del buffer.
- `server.receive(packet)`: il server rimane in attesa di ricevere un datagramma UDP dalla rete salvandolo nel `DatagramPacket` creato. La `receive` e' bloccante.
- `String s = new String(packet.getData(), 0, packet.getLength())`: costruisce la stringa decodificando l'array di byte presenti nel datagramma ricevuto. Gli ultimi due parametri indicano l'indice del primo byte da decodificare e il numero di byte da decodificare.
- `packet.setLength(buffer.length)`: quando un datagramma viene ricevuto, automaticamente la lunghezza del buffer interno viene settata alla lunghezza dei dati presenti nel datagramma. Quindi, la lunghezza originale deve essere ripristinata.

Come Eseguire l'applicazione

- Come eseguire l'applicazione in maniera distribuita?
- Il client ed il server devono essere eseguiti su due host diversi!
- Per utilizzare un computer remoto occorre una shell remota: utilizzo di ssh.
  - Es.: il client e' eseguito localmente sull'host fujim9 e tramite ssh il server e' avviato su fujim10.
- Il percorso completo dei due file e':
  `~/java/UDPHello/client/ClientUDP.java`,
  `~/java/UDPHello/server/ServerUDP.java`.
  - Il simbolo ~ (tilde) identifica l'home directory.

Esempio

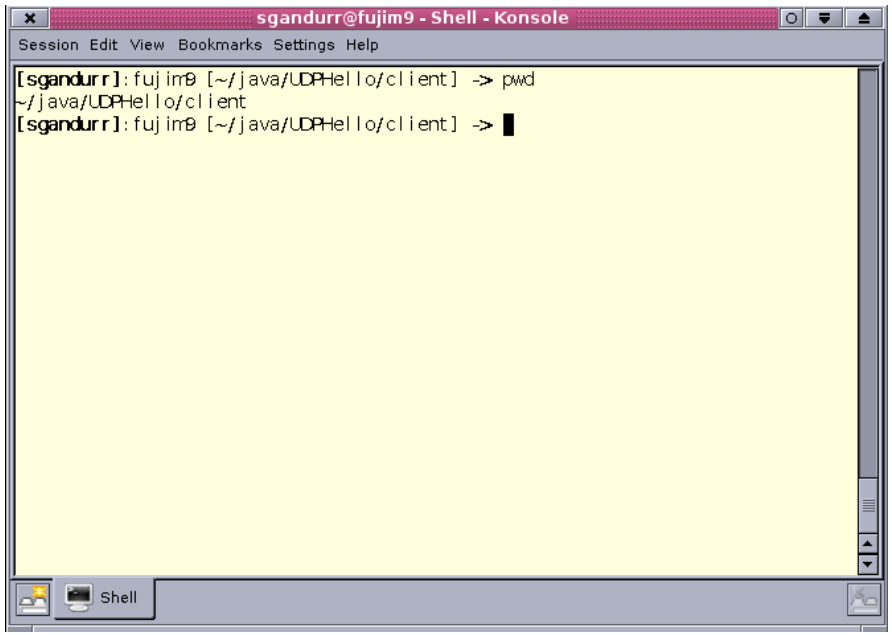


Ricapitolando...

1. Sono sul computer fujim9. Qui eseguiro' il client.
2. Da fujim9 eseguo il comando `ssh fujim10` per loggarmi remotamente su fujim10.
3. I comandi che voglio inviare a fujim10 sono digitati da una console su fujim9 e tramite ssh arrivano al computer remoto fujim10.
4. Avvio ServerUDP su fujim10 che rimane in ascolto su un socket UDP.
5. Avvio ClientUDP su fujim9: il client invia una stringa tramite socket UDP al server.
6. Il server riceve la stringa e la stampa a video.

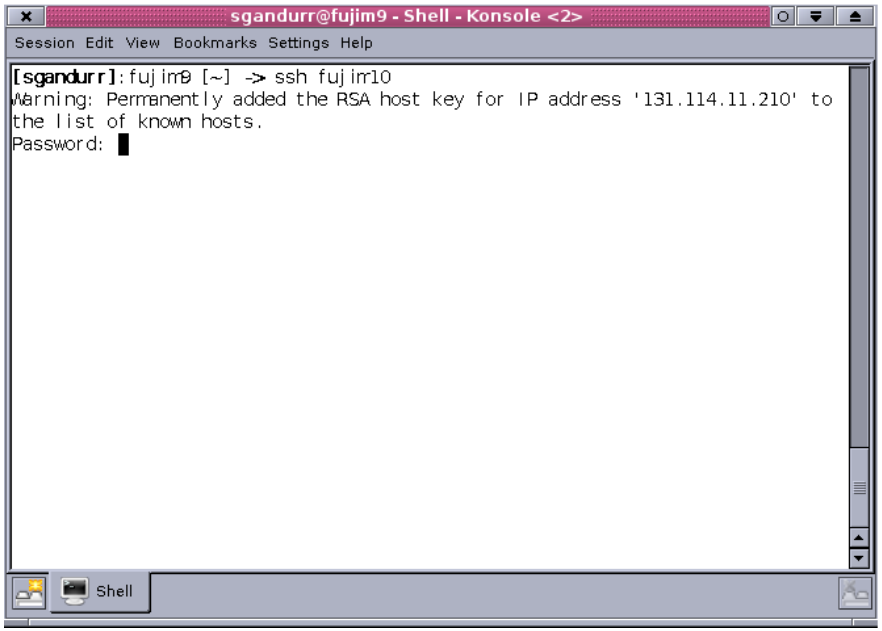
In Locale

- In locale sul fujim9 mi sposto nella directory del client: `cd ~/java/UDPHello/client` (cd = change directory).
- Il comando `pwd` (print working directory) mostra la directory corrente.

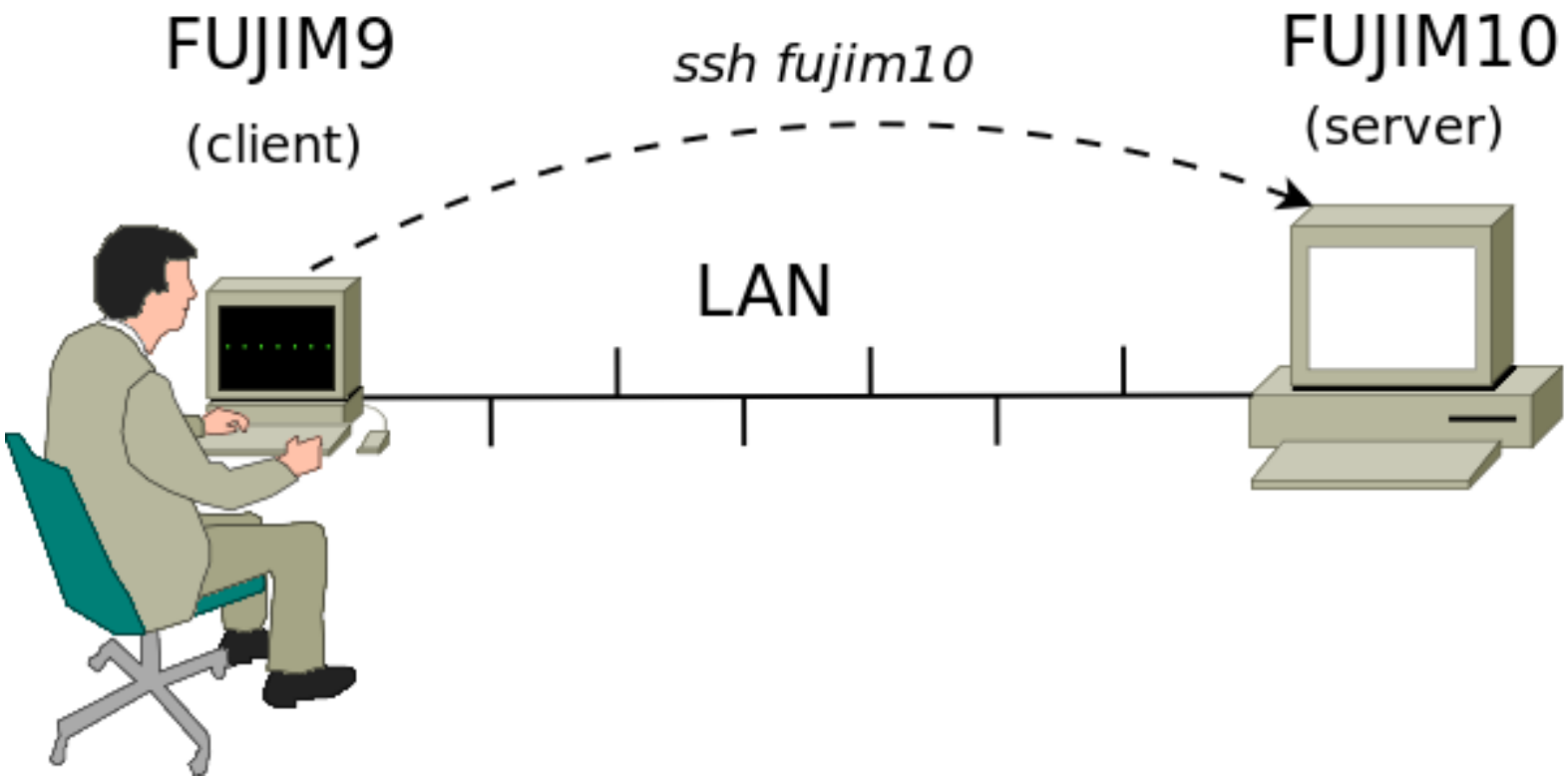


SSH

- Da un'altra shell su fujim9 eseguo il comando `ssh fujim10` per avere una shell remota su fujim10.
- ssh (secure shell) e' un protocollo utilizzato per stabilire una sessione remota cifrata con un host remoto. Analogo al telnet (DA EVITARE!!!), ma cifrato.

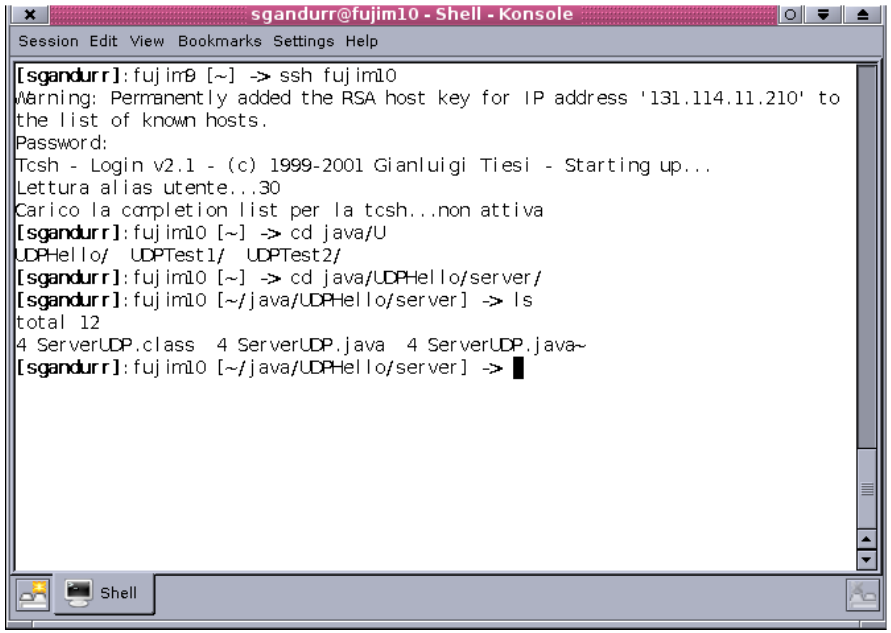


Esempio

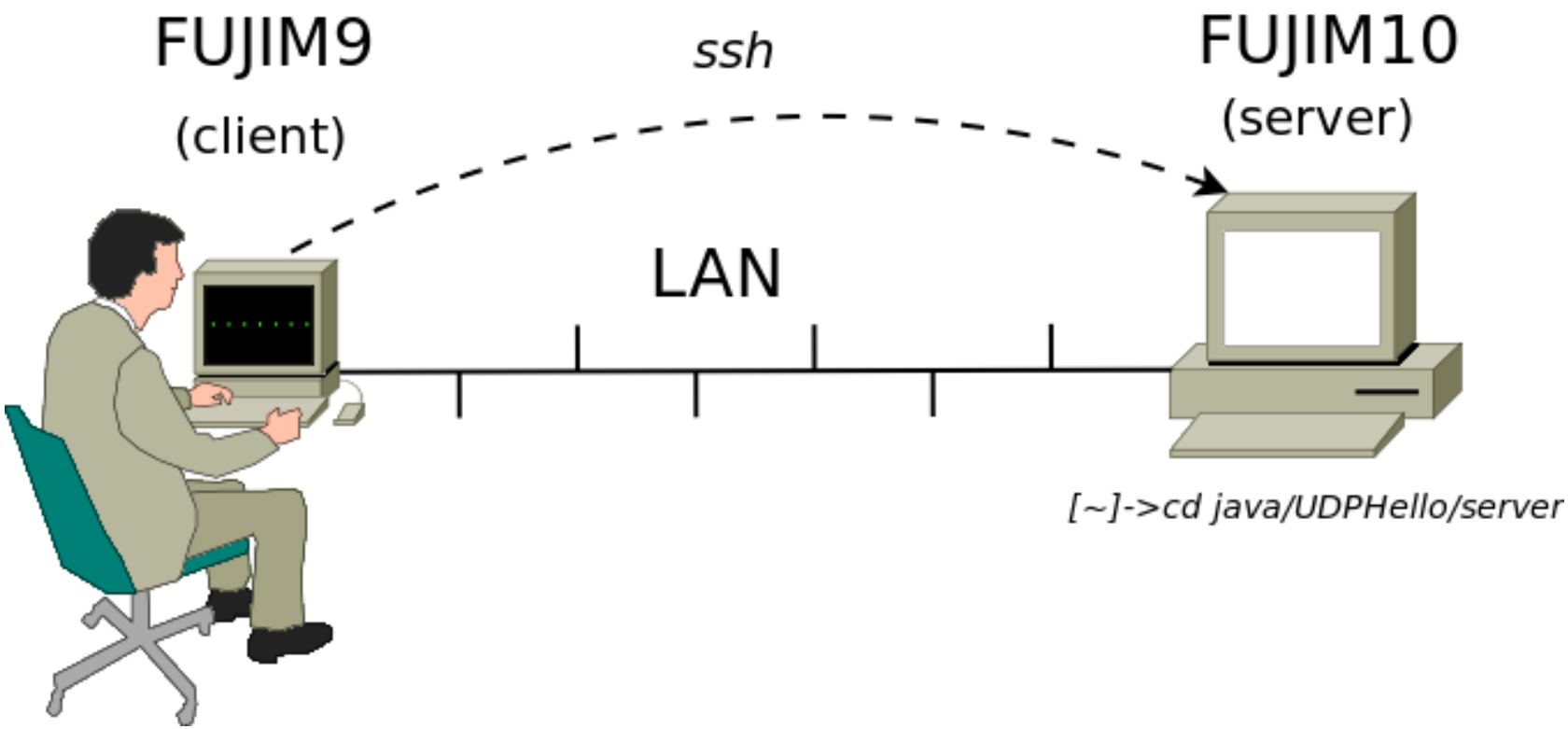


Shell Remota

- Adesso e' possibile eseguire comandi direttamente sull'host remoto fujim10.
- Eseguo il comando `cd ~/java/UDPHello/server`.

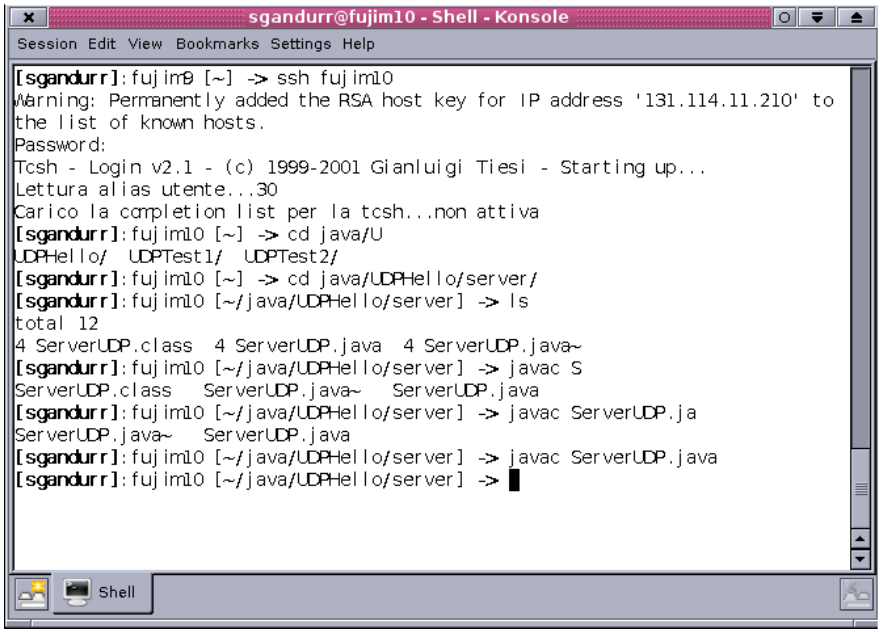


Esempio


FUJIM9
(client)
ssh
FUJIM10
(server)
LAN
[~]->cd java/UDPHello/server

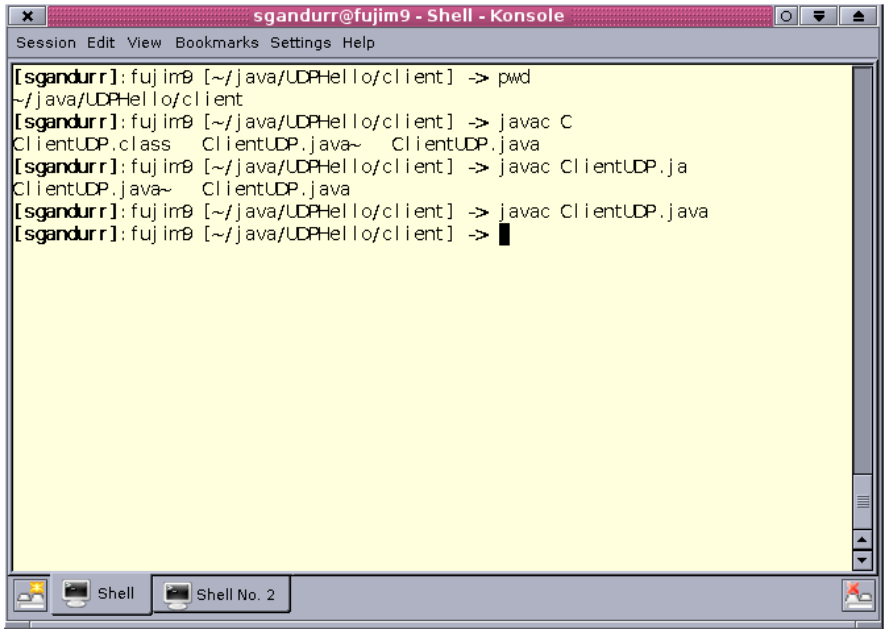
Compilare il Server Remotamente

- Eseguo il comando `javac ServerUDP.java` su fujim10.



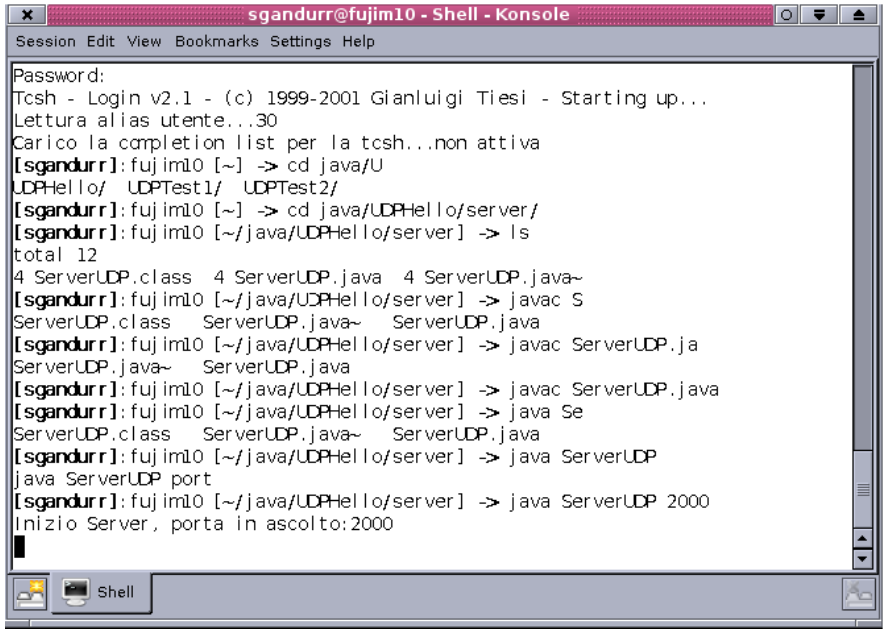
Compilare il Client Localmente

- Eseguo il comando `javac ClientUDP.java` su fujim9.



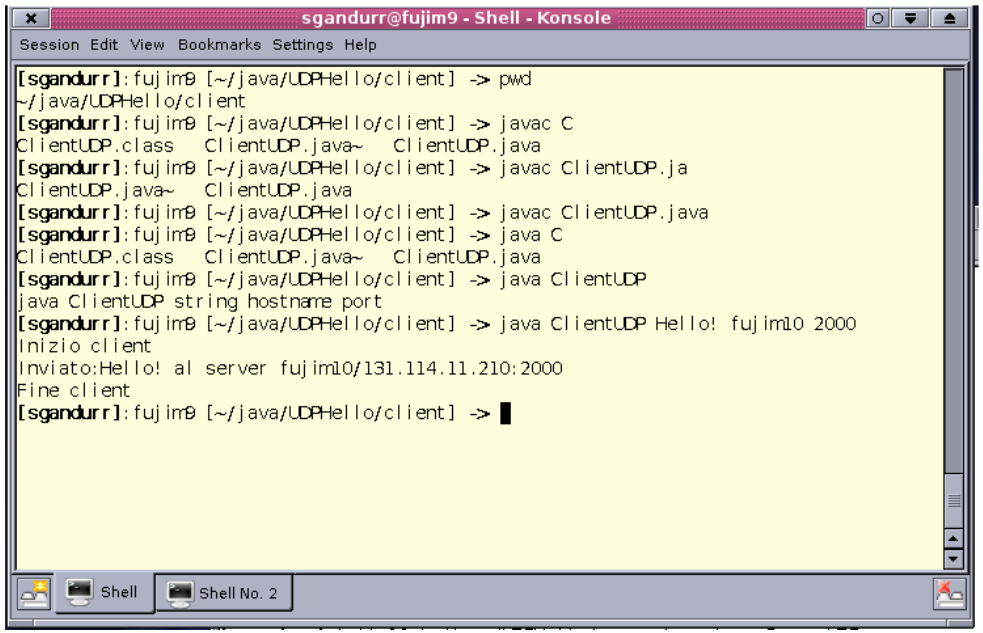
Avviare il Server

- Eseguo il comando `java ServerUDP 2000` su fujim10.
- Il server rimane in ascolto sulla porta `2000` UDP.



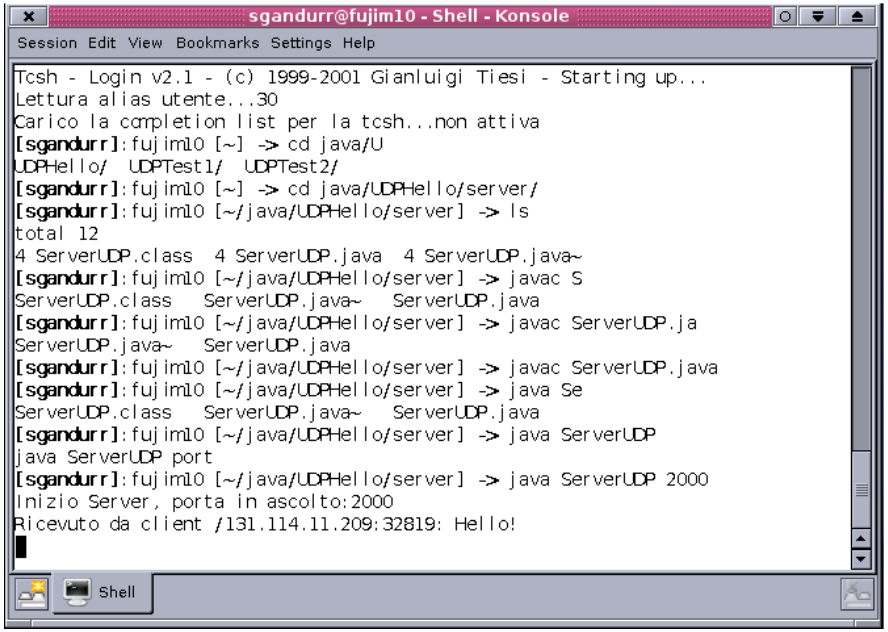
Avviare il Client

- Eseguo `java ClientUDP Hello! fujim10 2000` su fujim9.
- Il client invia la stringa `Hello!` a fujim10 sulla porta `2000`.



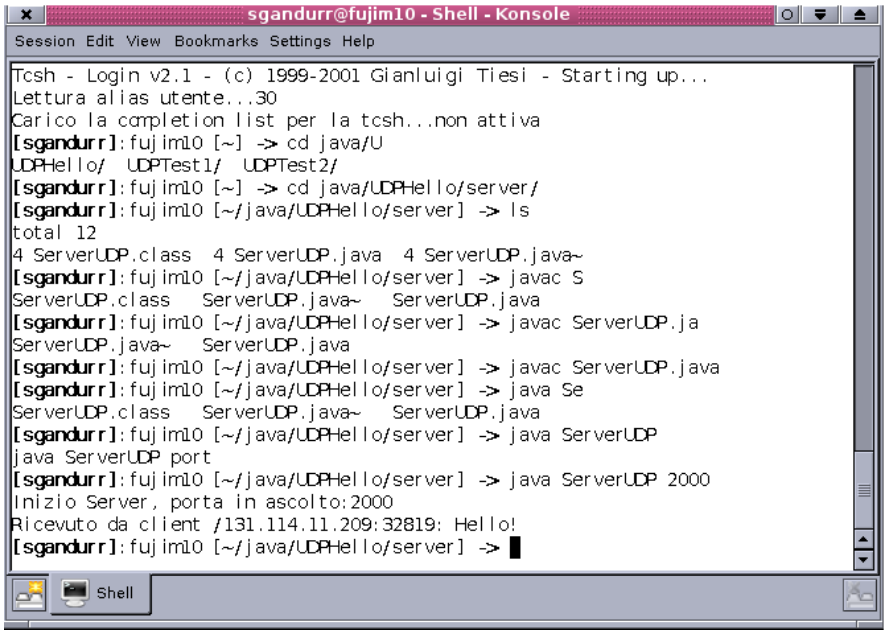
Il Server Riceve la Stringa

- Il server riceve la stringa `Hello!` dal client fujim9.
- Il server continua a rimanere in ascolto.

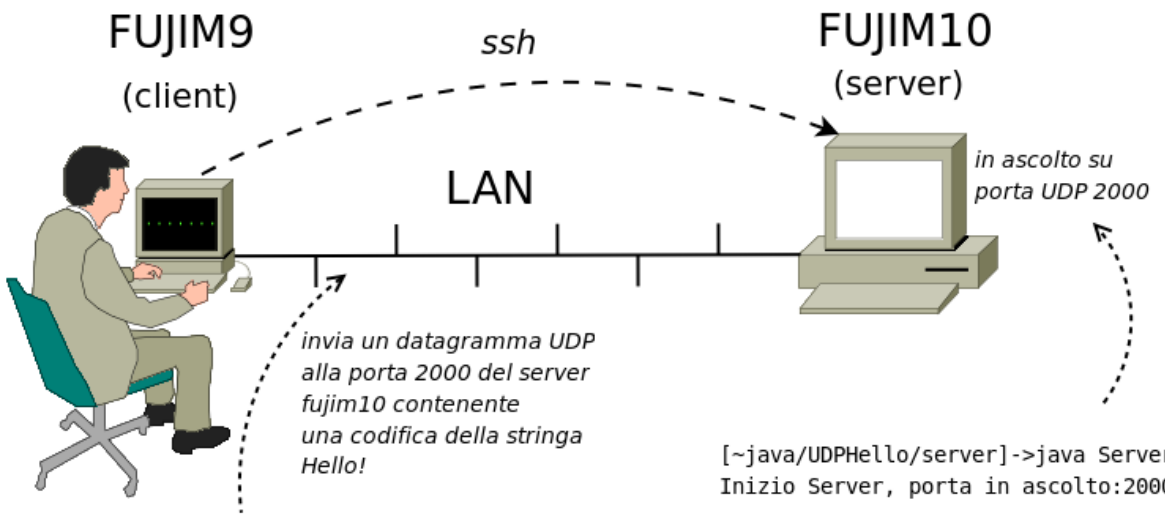


Terminare il Server

- Per terminare il server, digitare `ctrl + c`.
- Questa carattere e' utilizzato per inviare il segnale `SIGINT` (SIGnal INTerrupt) ad un processo per interromperlo.



Esempio

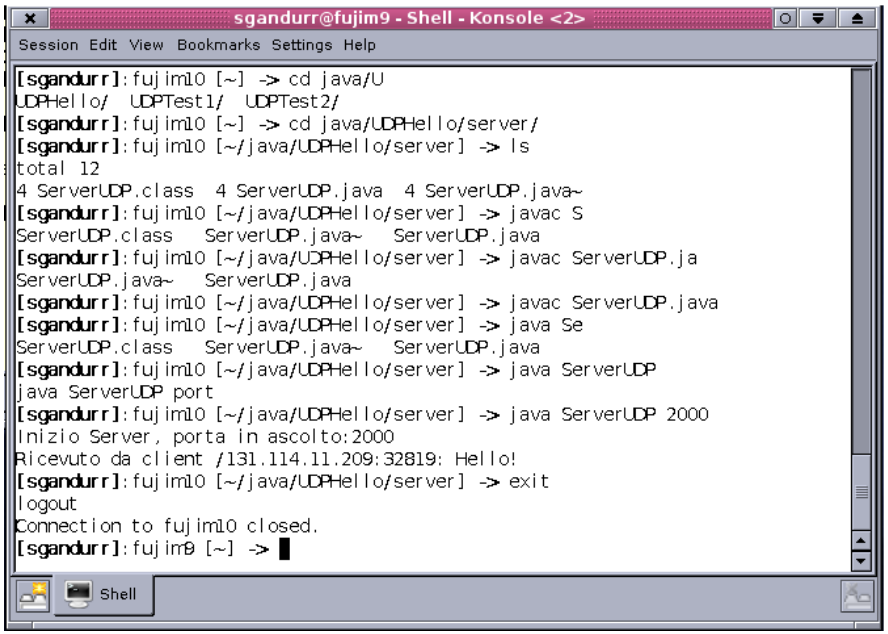



```
[~java/UDPHello/client]->java ClientUDP Hello! fujim10 2000
Inizio client
Inviato:Hello! al server fujim10/131.114.11.210:2000
Fine client
[~java/UDPHello/client]->
```

```
[~java/UDPHello/server]->java ServerUDP 2000
Inizio Server, porta in ascolto:2000

(..wait..)

Ricevuto da client /131.114.11.209: Hello!

(ctrl+c-->interrompe il processo)

[~java/UDPHello/server]->
```

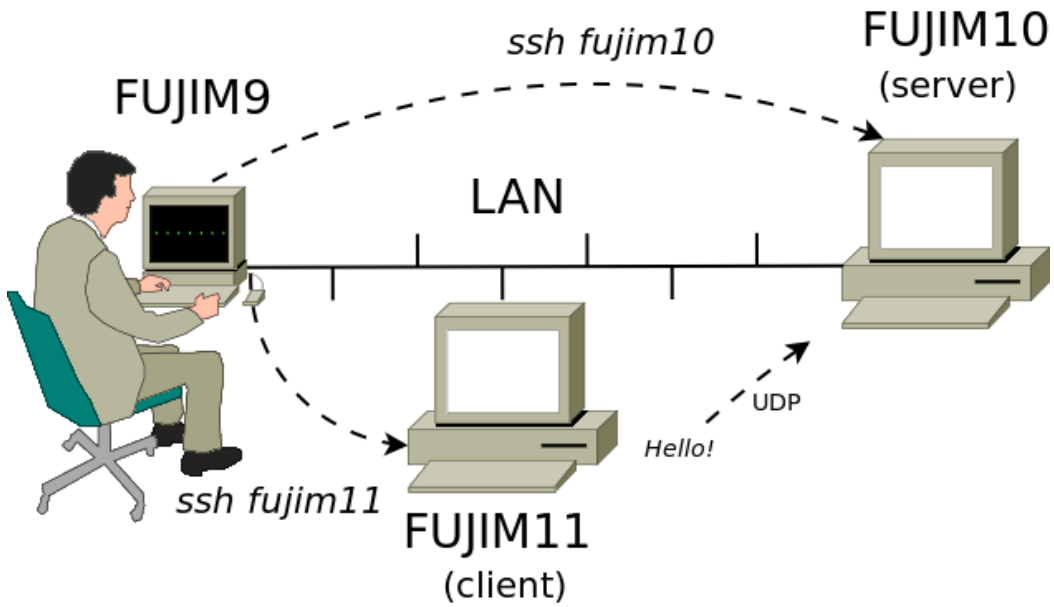
Uscire dalla Sessione Remota

- Per uscire dalla sessione remota ssh, e quindi effettuare il logout da fujim10, digitare `exit`.



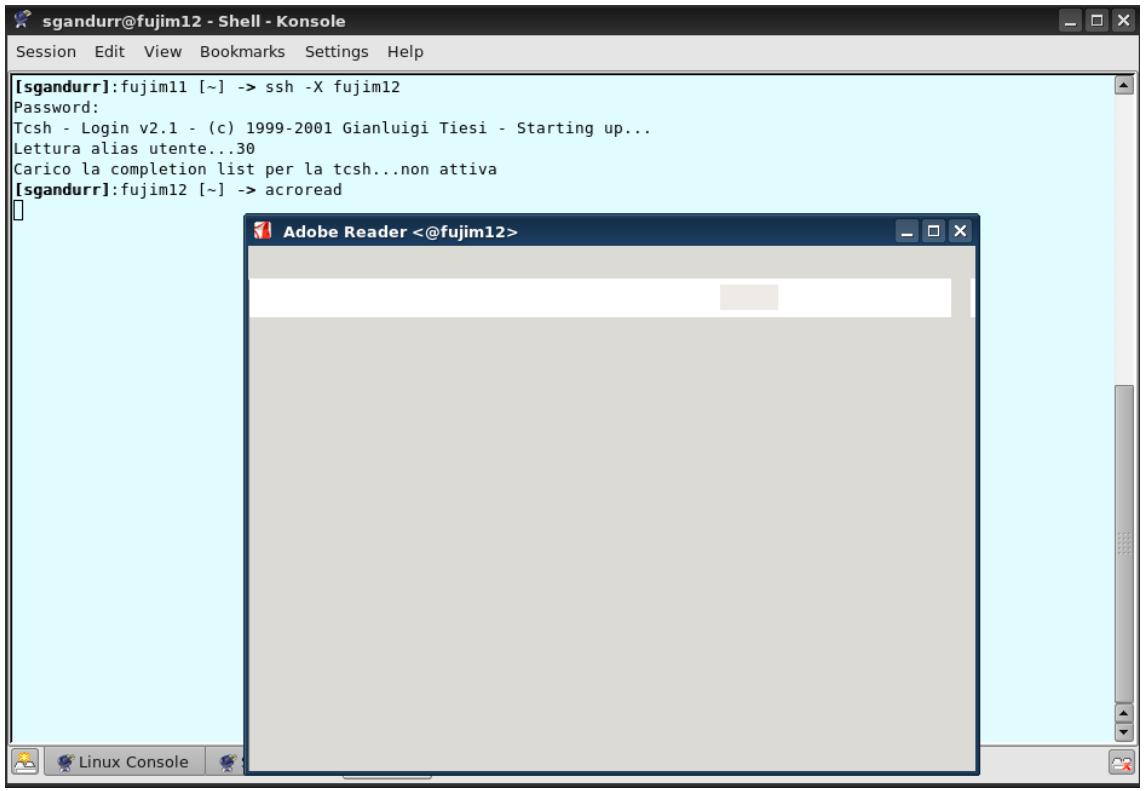
Client e Server in SSH

- Potete avviare anche il client via ssh.
- Aprite due shell remote, ad es.:
  1. `ssh fujim10` (per avviare il server).
  2. `ssh fujim11` (per avviare il client).



SSH e X11 Forwarding

Nel caso di applicazioni client/server Java che fanno uso di AWT/SWING, per abilitare la redirezione del display remoto in locale, usare l'opzione -X (maiuscolo!):
`ssh -X user@hostname`.



Il File known_hosts

- Nella home directory esiste la directory nascosta `.ssh`.
- Qui trovate il file `known_hosts`.
- Questo file contiene l'associazione host remoto / chiave pubblica, per ogni host su cui avete fatto ssh.
- Nel caso in cui dopo aver eseguito il comando `ssh host_remoto`, vi venga segnalato:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

  Ci sono due possibilita':

  1. `host_remoto` ha cambiato la chiave pubblica.
  2. Un terzo host sta cercando di impersonificare `host_remoto`: "man-in-the-middle attack".

- Per ulteriori informazioni su ssh: `man ssh`.

File System Distribuito

- Al polo, l'home directory viene "montata" sul File System locale di ogni host da uno storage remoto, tramite protocollo NFS (Network File System).
  - I file personali sono contenuti su uno o piu' dischi remoti (es. RAID: Redundant Array of Independent Disks).
- Questo significa che se effettuate una modifica ad un file, la modifica e' globale e quindi visibile a tutti i pc:
  - es.: se modificate da fujim9 il file `ServerUDP.java`, le modifiche sono visibili anche da fujim10.
  - Questo e' comodo, in quanto vi permette di modificare e compilare i file da un unico host.

I Processi

- Altro discorso sono i processi: i processi sono LOCALI all'host su cui eseguite il comando.
  1. E' possibile eseguire la Java Virtual Machine su due host diversi con lo stesso file class.
  2. Ad es. se volete avviare diversi client in parallelo non occorre creare diverse copie del client per ogni host (`ClientUDP1`, `ClientUDP2`, ...), ma basta utilizzare lo stesso file `ClientUDP`.
  3. Non e' possibile da fujim9 "killare" il processo `java` sull'host fujim10, ad es. per terminare il server dal client.

Gli Hostname e SFTP

- L'hostname completo e' `fujim10.cli.di.unipi.it`.
- Poiche' il dominio su cui risiedono i computer e' `cli.di.unipi.it`, si puo' semplicemente scrivere `fujim10`.
- Se vi collegate da casa tramite ssh, il dominio va inserito, per cui il comando completo e':
  `ssh username@fujim10.cli.di.unipi.it`.
- Per trasferire file potete usare sftp. Per connettersi:
  `sftp username@fujim10.cli.di.unipi.it`.
  - Per fare l'upload di un file: `put nomefile`.
  - Per fare il download di un file: `get nomefile`.
  - Per cambiare directory all'interno dell'host remoto: `cd`.

Numeri di Porta

Intervalli delle porte (TCP/UDP):

- $[0, 1023]$: le Well Known Ports che, solitamente, possono essere usate solo da utenti privilegiati.
- $[1024, 49151]$: le Registered Ports.
- $[49152, 65535]$: le Dynamic Ports.
- Lato server scegliete una porta libera: il client deve conoscere questa porta.
- Per vedere le porte attualmente utilizzate:

```
[~]->netstat -antu
```

Vedere:
`http://www.iana.org/assignments/port-numbers`.

Localhost

Infine...

- Il sistema operativo puo' mettere a disposizione delle interfacce di loopback (interfacce virtuali: non sono associate ad alcun dispositivo fisico).
- Su ogni PC l'hostname `localhost` identifica l'interfaccia di loopback con l'IP `127.0.0.1`.
  - Vedi file `/etc/hosts`.
- E' caldamente sconsigliato eseguire i processi client e server sulla stessa macchina utilizzando un'interfaccia locale per le connessioni (es. l'interfaccia di loopback), a meno di essere senza scheda di rete o se la rete ha dei problemi :-)