Livello applicazione: programmazione socket, P2P

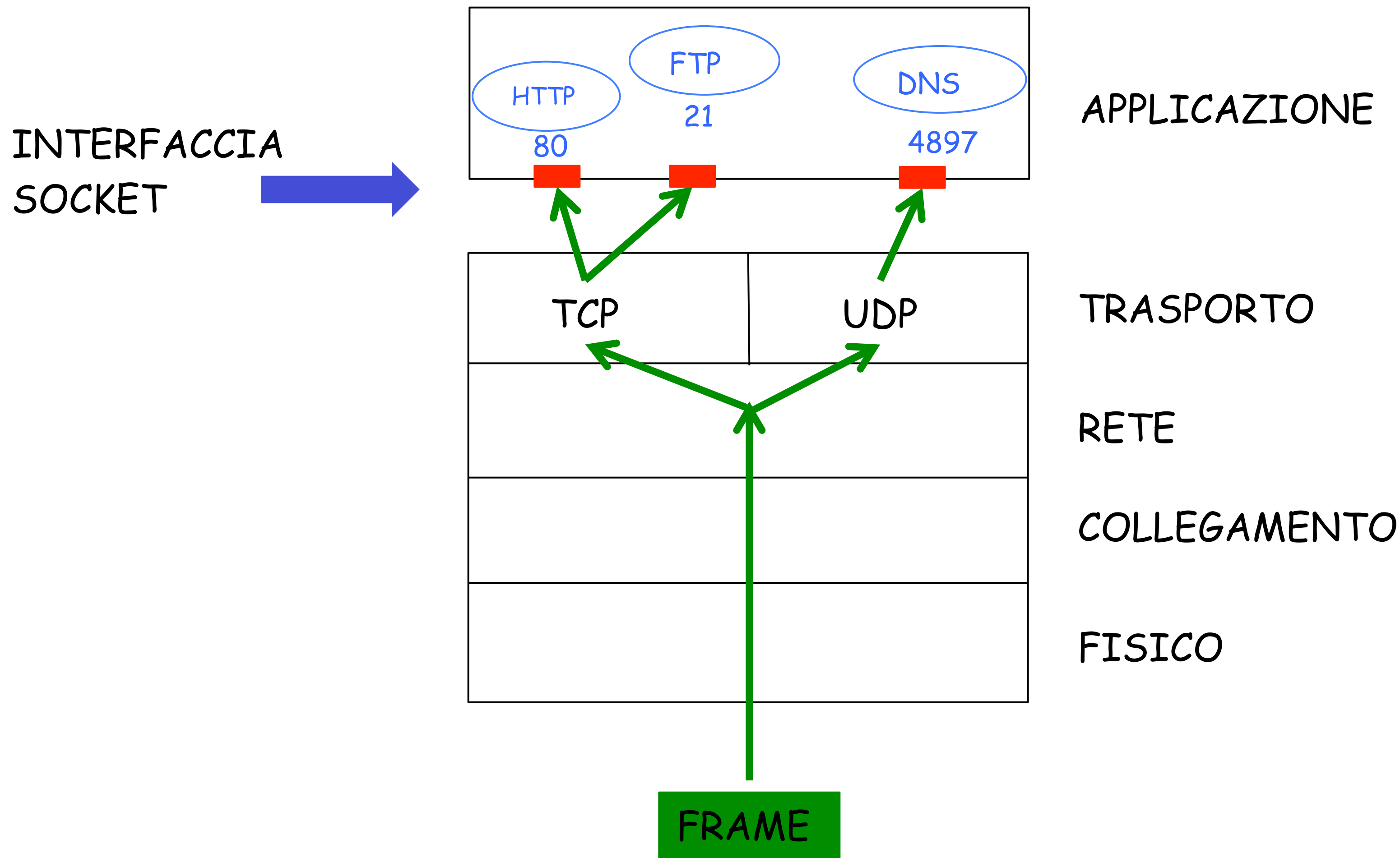
Gaia Maselli

Queste slide sono un adattamento delle slide fornite dal libro di testo e pertanto protette da copyright.
All material copyright 1996-2007 J.F Kurose and K.W. Ross, All Rights Reserved

Processi applicativi

- Un host può avere più processi applicativi attivi contemporaneamente (FTP, Web client, sessioni Telnet)
- Quando il livello di trasporto riceve dati dal livello di rete, deve sapere a chi passare i dati (a quale processo applicativo)
- I dati vengono passati ai processi applicativi mediante l'interfaccia dei socket (IP, porta)
- Ogni processo avrà un socket per inviare e ricevere dati
- Un processo può avere più socket aperti (es. un web server serve più richieste di diversi client e ha una connessione con ognuno di essi)
- Ogni socket ha un identificatore univoco

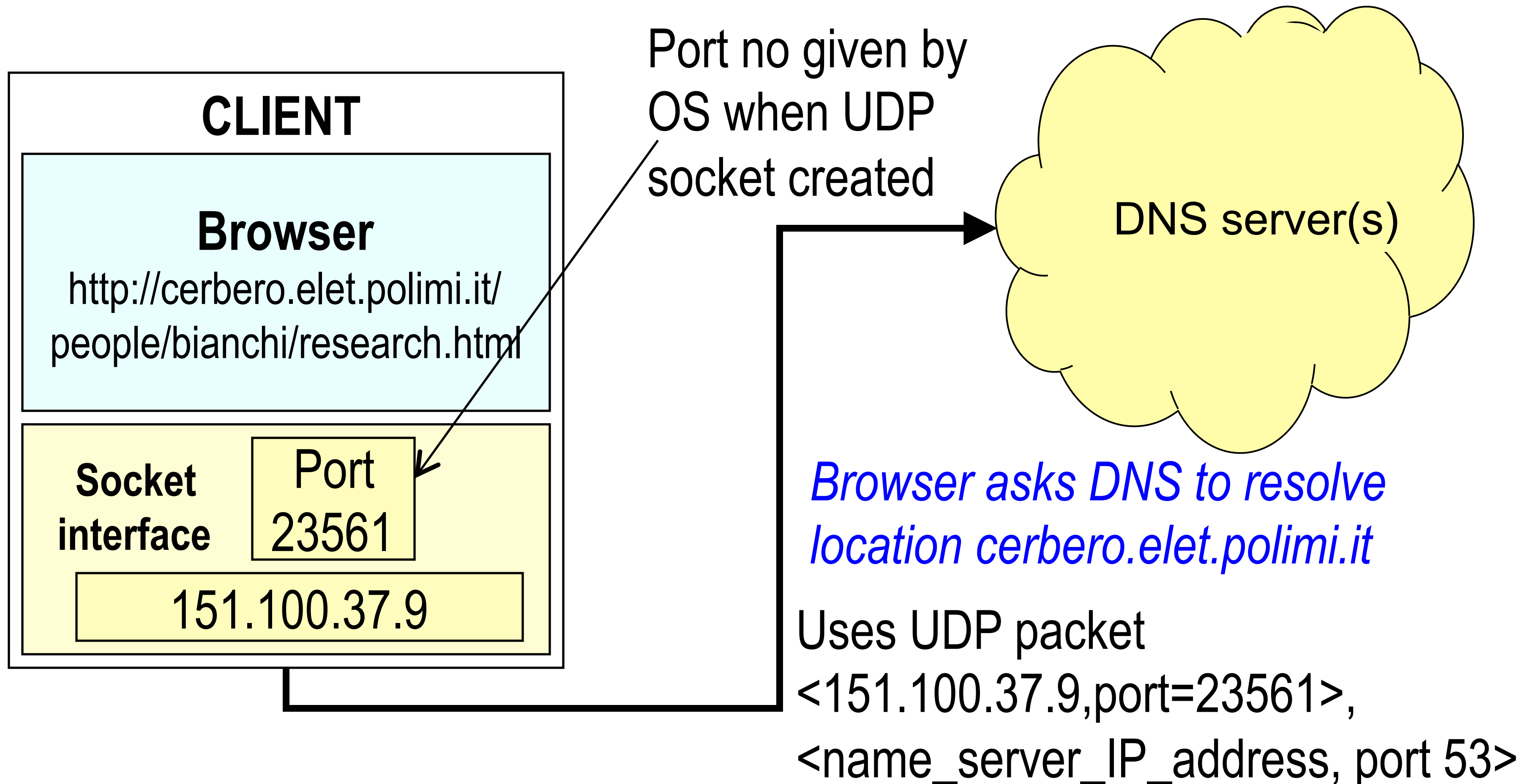
Come viene recapitato un pacchetto all'applicazione



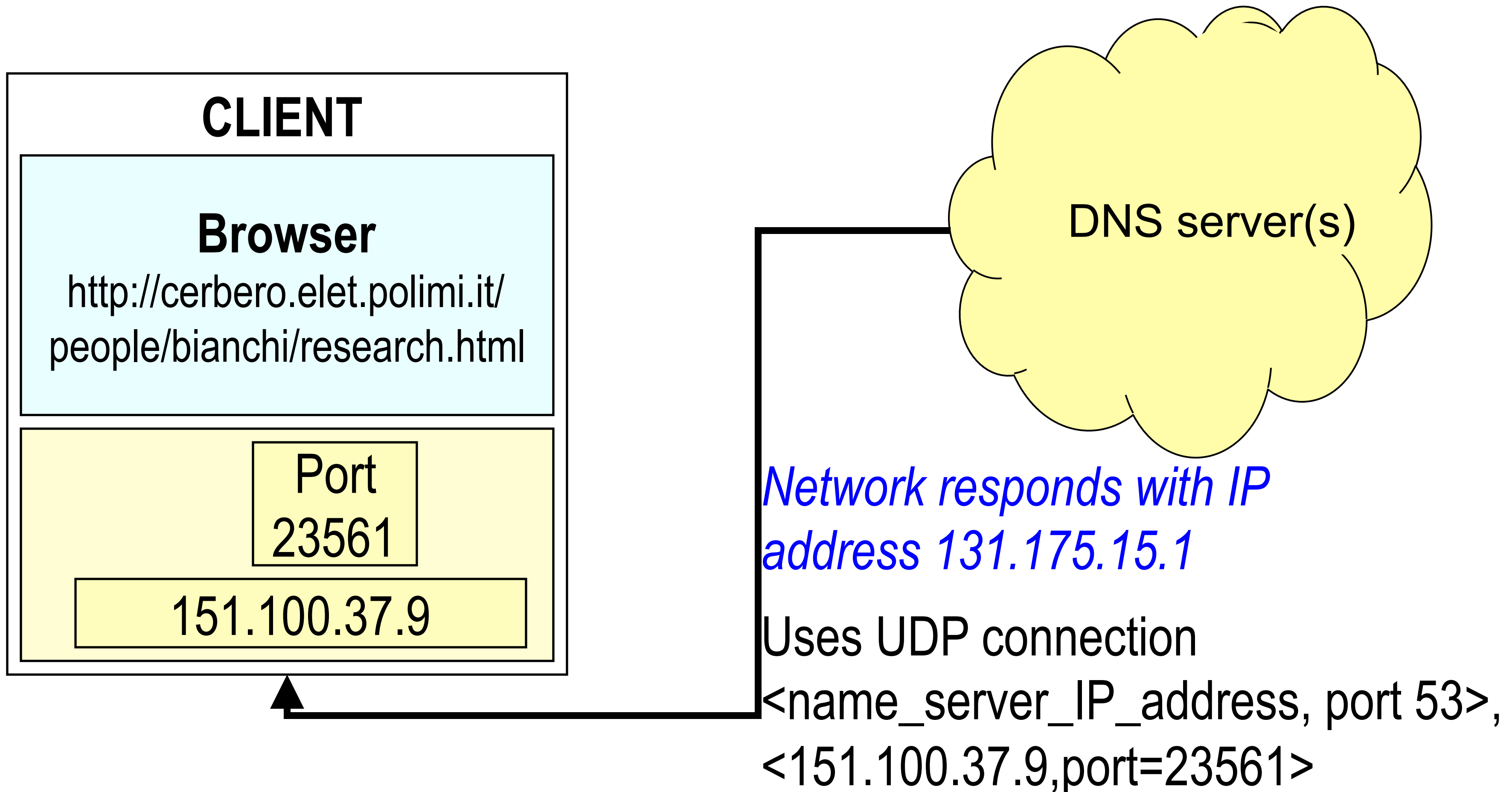
Port numbers

- 16 bit address (0-65535)
- well known port numbers for common servers
  - FTP 20, TELNET 23, SMTP 25, HTTP 80, POP3 110, … (full list: RFC 1700)
- number assignment (by IANA)
  - 0 not used
  - 1-255 reserved for well known processes
  - 256-1023 reserved for other processes
  - 1024-65535 dedicated to user apps
- See IANA for port numbers
  - TCP
  - UDP

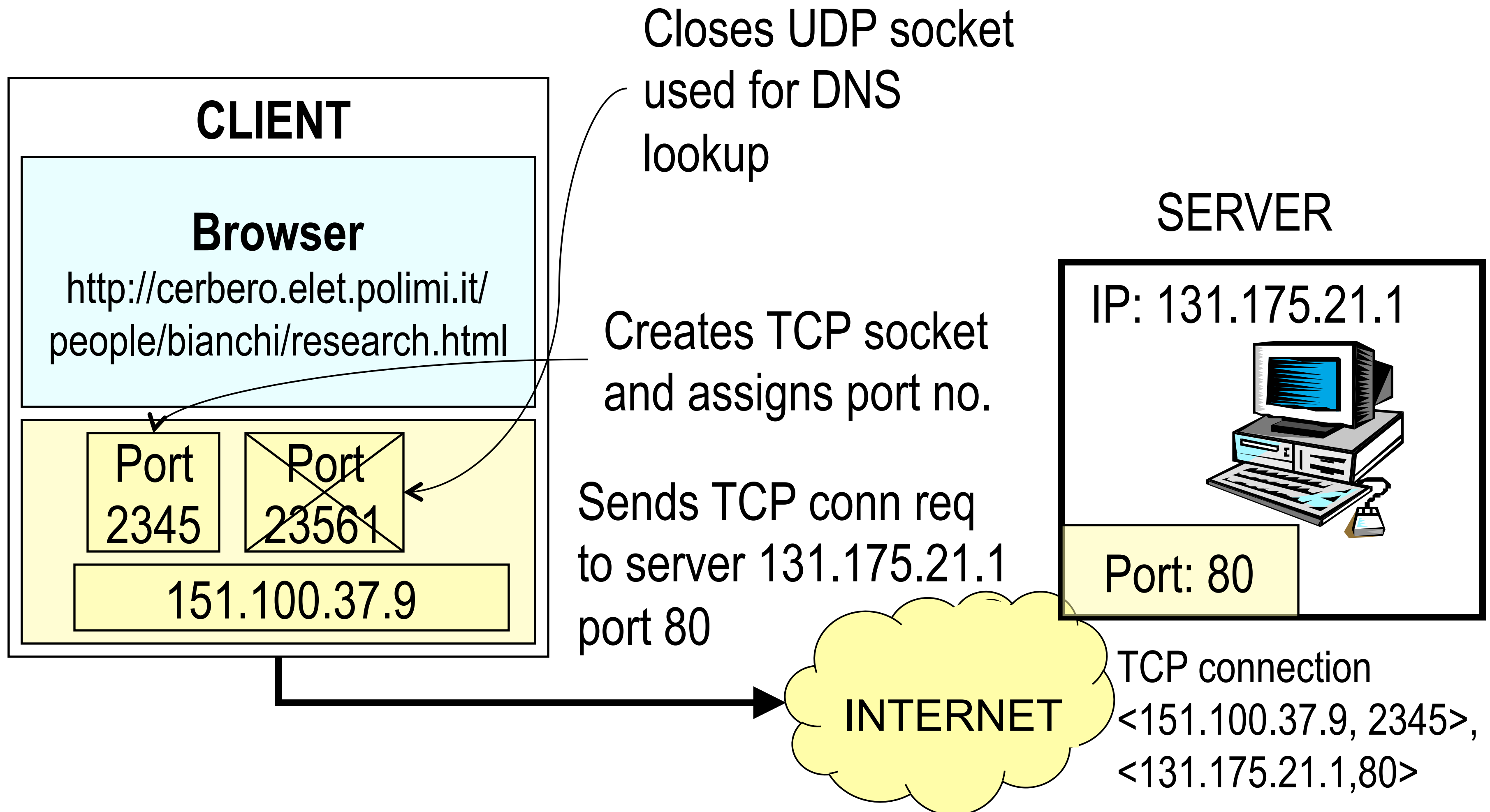
Un esempio: uso di DNS da parte di un client web



opening transport session: client side, step b



opening transport session: client side, step c



opening transport session: server side

- httpd (http daemon) process listens for arrival of connection requests from port 80.
- Upon connection request arrival, server decides whether to accept it, and send back a TCP connection accept
- This opens a TCP connection, uniquely identified by client address+port and server address+port 80

Applicazioni di rete

- Applicazioni di rete: coppia di programmi client-server, che risiedono su sistemi diversi
- In esecuzione: processo client e processo server comunicano tramite socket
- Applicazione di rete può implementare
  - Protocollo *standard* definito (es. in una RFC)
    - Conformità all'RFC (garantita l'interazione)
    - Utilizzo di porte assegnate
    - Sviluppatori differenti e anche solo da un lato
  - Applicazione *proprietaria*
    - Sviluppo sia del client che del server
    - Utilizzo di porte non assegnate
    - No sviluppatori indipendenti
- Decisione preliminare: TCP o UDP?
  - In base al servizio di trasporto richiesto cambia il tipo di socket da usare

Programmazione dei socket

Obiettivo: imparare a costruire un'applicazione client/server che comunica utilizzando le socket

Socket API

- introdotta in BSD4.1 UNIX, nel 1981
- esplicitamente creata, usata, distribuita dalle applicazioni
- paradigma client/server
- due tipi di servizio di trasporto tramite una socket API:
  - datagramma inaffidabile
  - affidabile, orientata ai byte

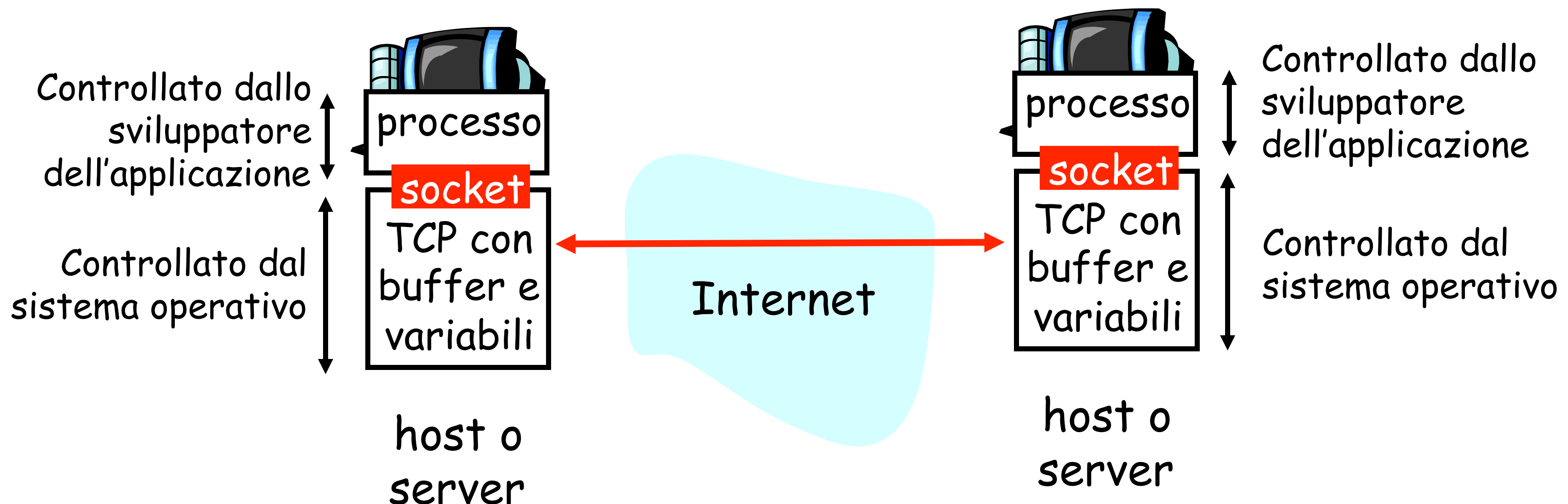
socket

Interfaccia di un *host locale*, *creata dalle applicazioni*, *controllata dal SO* in cui il processo di un'applicazione può inviare e ricevere messaggi al/dal processo di un'altra applicazione

Programmazione dei socket con TCP

Socket: un'ingresso tra il processo di un'applicazione e il protocollo di trasporto end-end (UDP o TCP)

Servizio TCP: trasferimento affidabile di **byte** da un processo all'altro



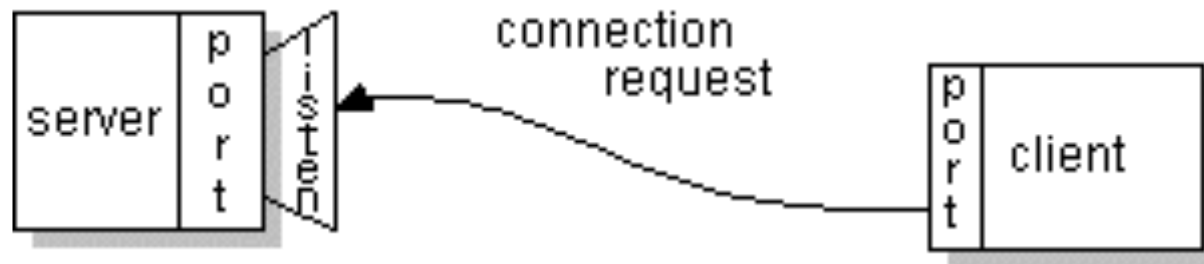
Programmazione dei socket con TCP

Il client deve contattare il server

- Il processo server deve essere in corso di esecuzione (sempre attivo)
- Il server deve aver creato un socket che dà il benvenuto al contatto con il client

Il client contatta il server:

- Creando un socket TCP
- Specificando l'indirizzo IP, il numero di porta del processo server
- Quando il client crea il socket: il client TCP stabilisce una connessione con il server TCP

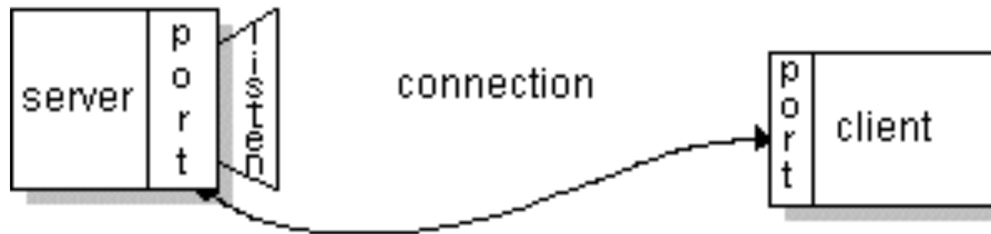


Programmazione dei socket con TCP (2)

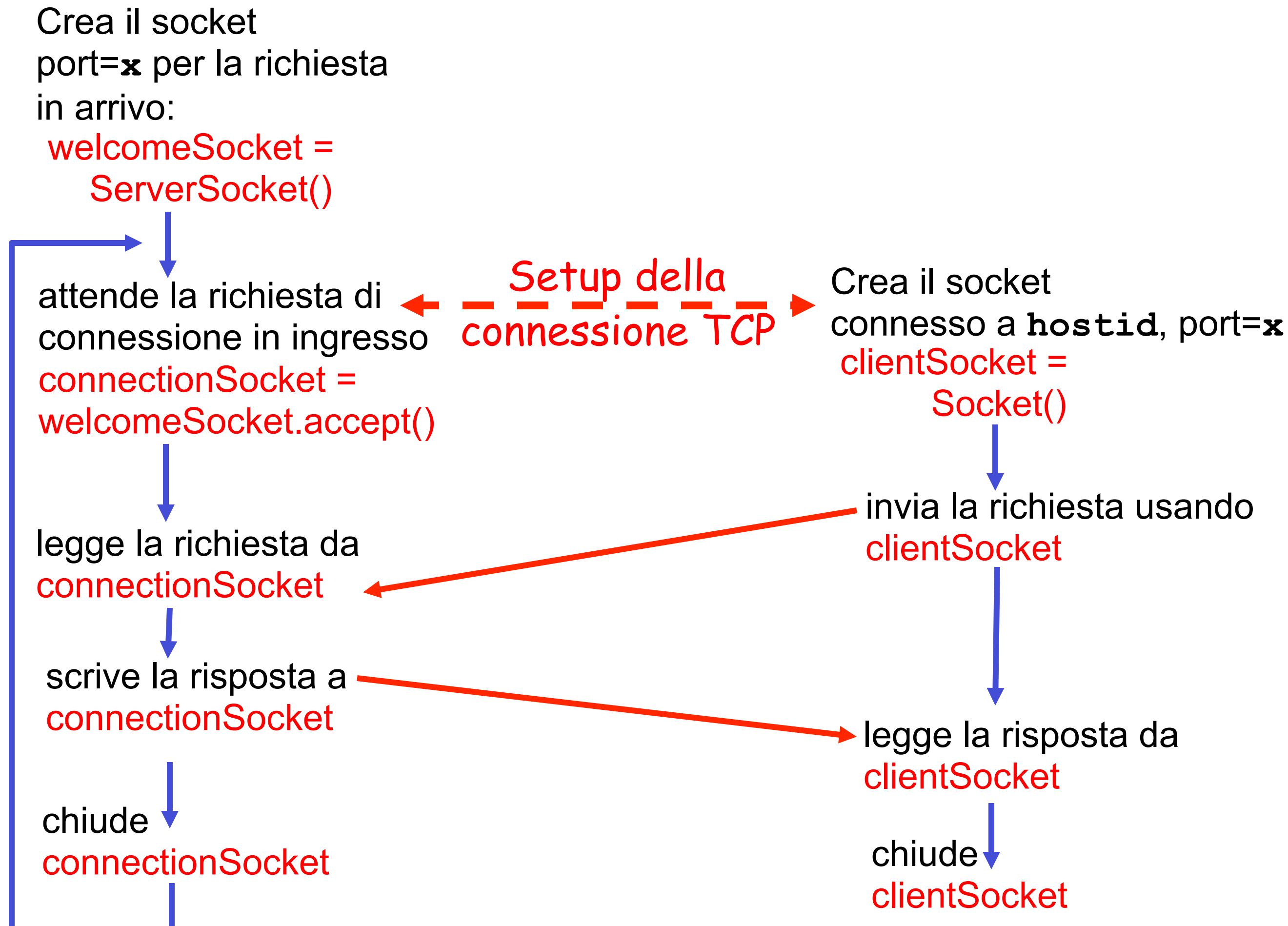
- Quando viene contattato dal client, il server TCP crea un nuovo socket per il processo server per comunicare con il client
  - consente al server di comunicare con più client
  - numeri di porta origine usati per distinguere i client (maggiori informazioni
    nelle lezioni su TCP)

Punto di vista dell'applicazione

TCP fornisce un trasferimento di byte affidabile e ordinato ("pipe") tra client e server

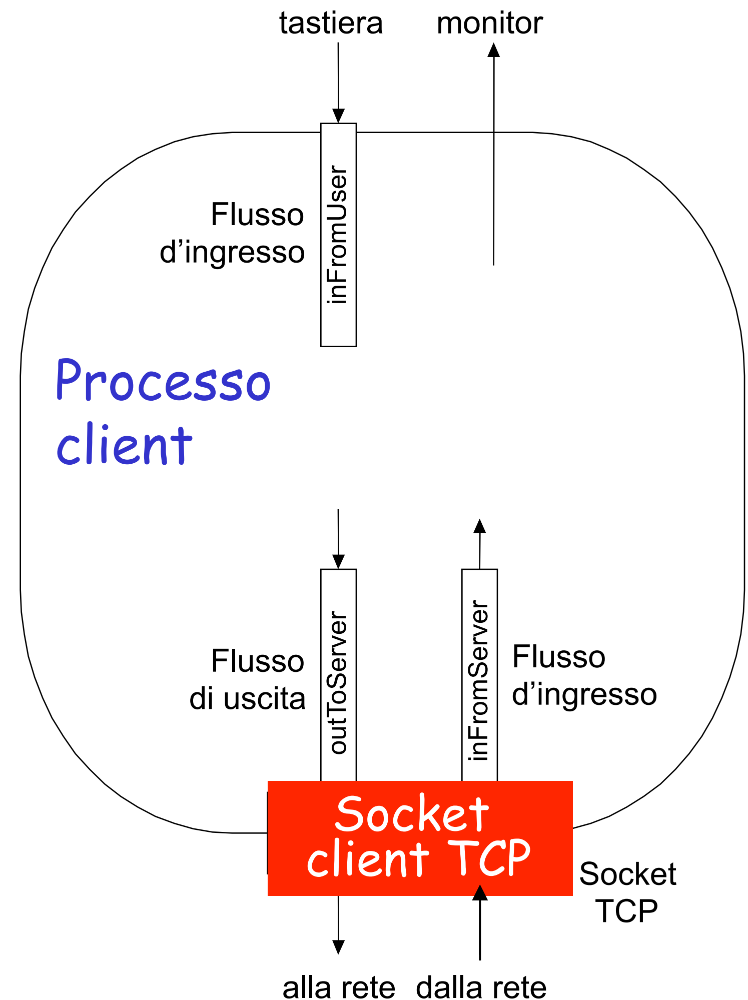


Interazione dei socket client/server: TCP



Termini

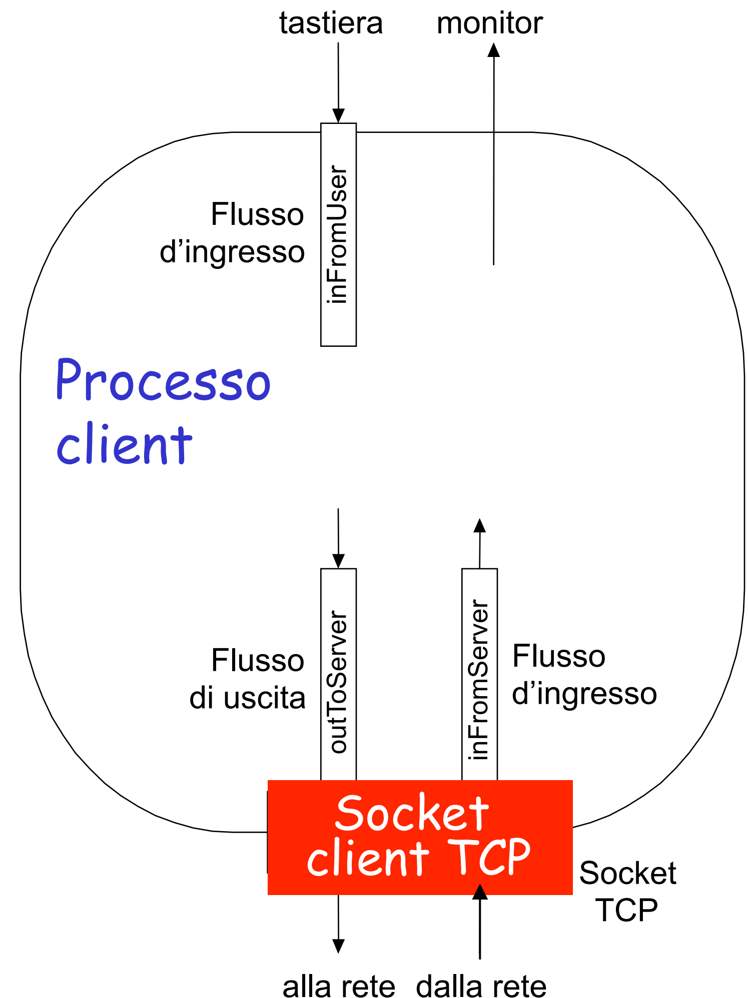
- Un flusso (*stream*) è una sequenza di caratteri che fluisce verso/da un processo.
- Un flusso d'ingresso (*input stream*) è collegato a un'origine di input per il processo, ad esempio la tastiera o la socket.
- Un flusso di uscita (*output stream*) è collegato a un'uscita per il processo, ad esempio il monitor o la socket.



Programmazione delle socket con TCP

Esempio di applicazione client-server:

1) Il client legge una riga dall'input standard (flusso **`inFromUser`**) e la invia al server tramite la socket (flusso **`outToServer`**)
2) Il server legge la riga dalla socket
3) Il server converte la riga in lettere maiuscole e la invia al client
4) Il client legge nella sua socket la riga modificata e la visualizza (flusso **`inFromServer`**)



Esempio: client Java (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Crea un flusso d'ingresso → `BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));`

Crea un socket client, connesso al server → `Socket clientSocket = new Socket("hostname", 6789);`

Crea un flusso di uscita collegato al socket → `DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());`

Esempio: client Java (TCP), continua

Crea un flusso d'ingresso collegato alla socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
```

Invia la frase inserita dall'utente al server

```
outToServer.writeBytes(sentence + '\n');
```

Legge la risposta dal server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Chiude socket e connessione con server

Esempio: server Java (TCP)

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

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

        Socket connectionSocket = welcomeSocket.accept();

       BufferedReader inFromClient =
         new BufferedReader(new
         InputStreamReader(connectionSocket.getInputStream()));
```

Crea un socket di benvenuto sulla porta 6789 → `ServerSocket welcomeSocket = new ServerSocket(6789);`

Attende, sul socket di benvenuto, un contatto dal client → `Socket connectionSocket = welcomeSocket.accept();`

Crea un flusso d'ingresso collegato al socket → `BufferedReader inFromClient = ...`

Esempio: server Java (TCP), continua

Crea un flusso di uscita collegato al socket →

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Legge la riga dal socket →

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Scrive la riga sul socket →

```
outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

Fine del ciclo while, ricomincia il ciclo e attende un'altra connessione con il client

from The Java^{TM} Tutorial

- **Definition:** *A socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.
- An endpoint is a combination of an **IP address** and a **port number**. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

Esempio con socket TCP

- Web client semplificato: programma che richiede una pagina web a un server, senza interfaccia grafica
- Codice

Programmazione delle socket *con UDP*

UDP: non c'è "connessione" tra client e server

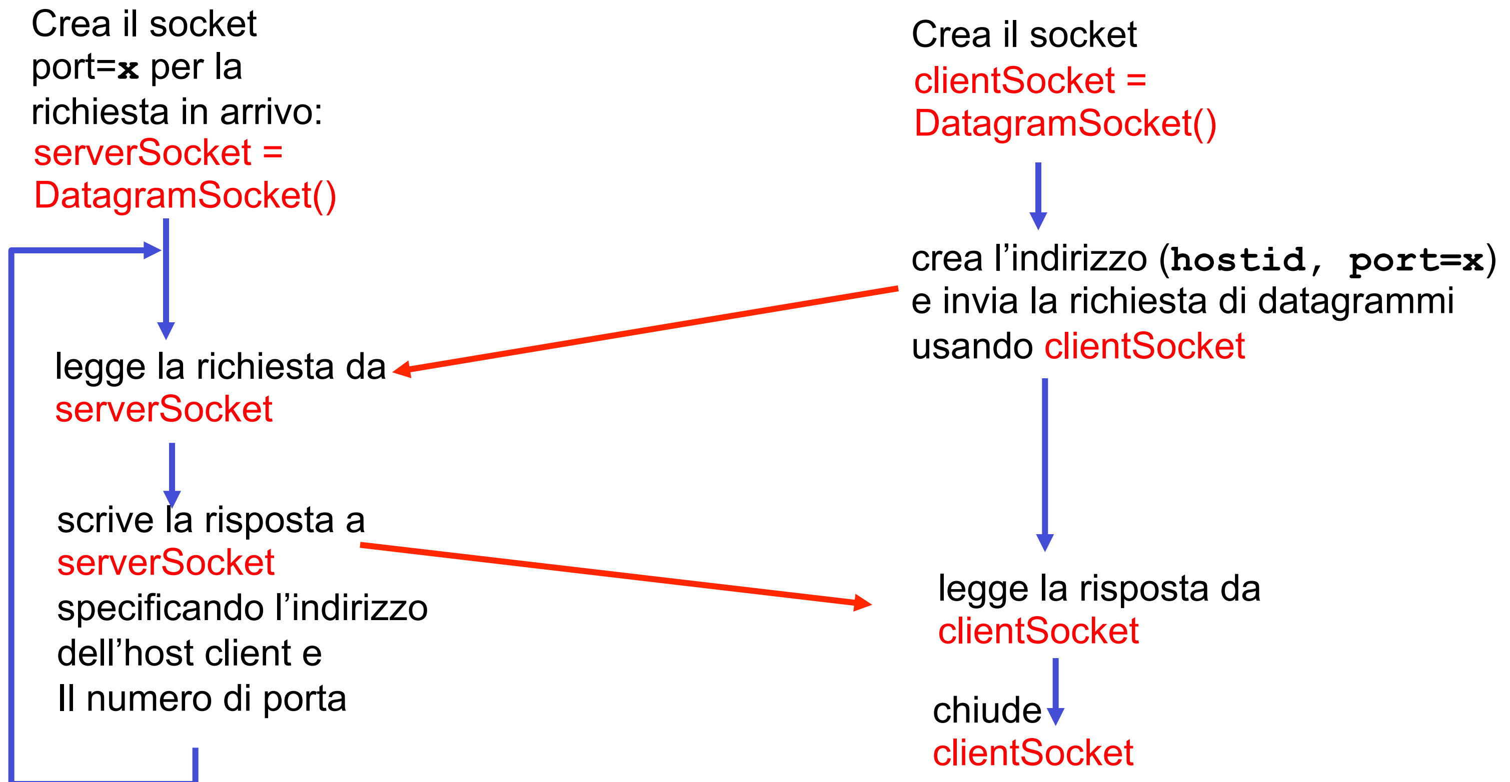
- Non c'è handshaking
- Il mittente allega esplicitamente a ogni pacchetto l'indirizzo IP e la porta di destinazione
- Il server deve estrarre l'indirizzo IP e la porta del mittente dal pacchetto ricevuto

UDP: i dati trasmessi possono perdersi o arrivare a destinazione in un ordine diverso da quello d'invio

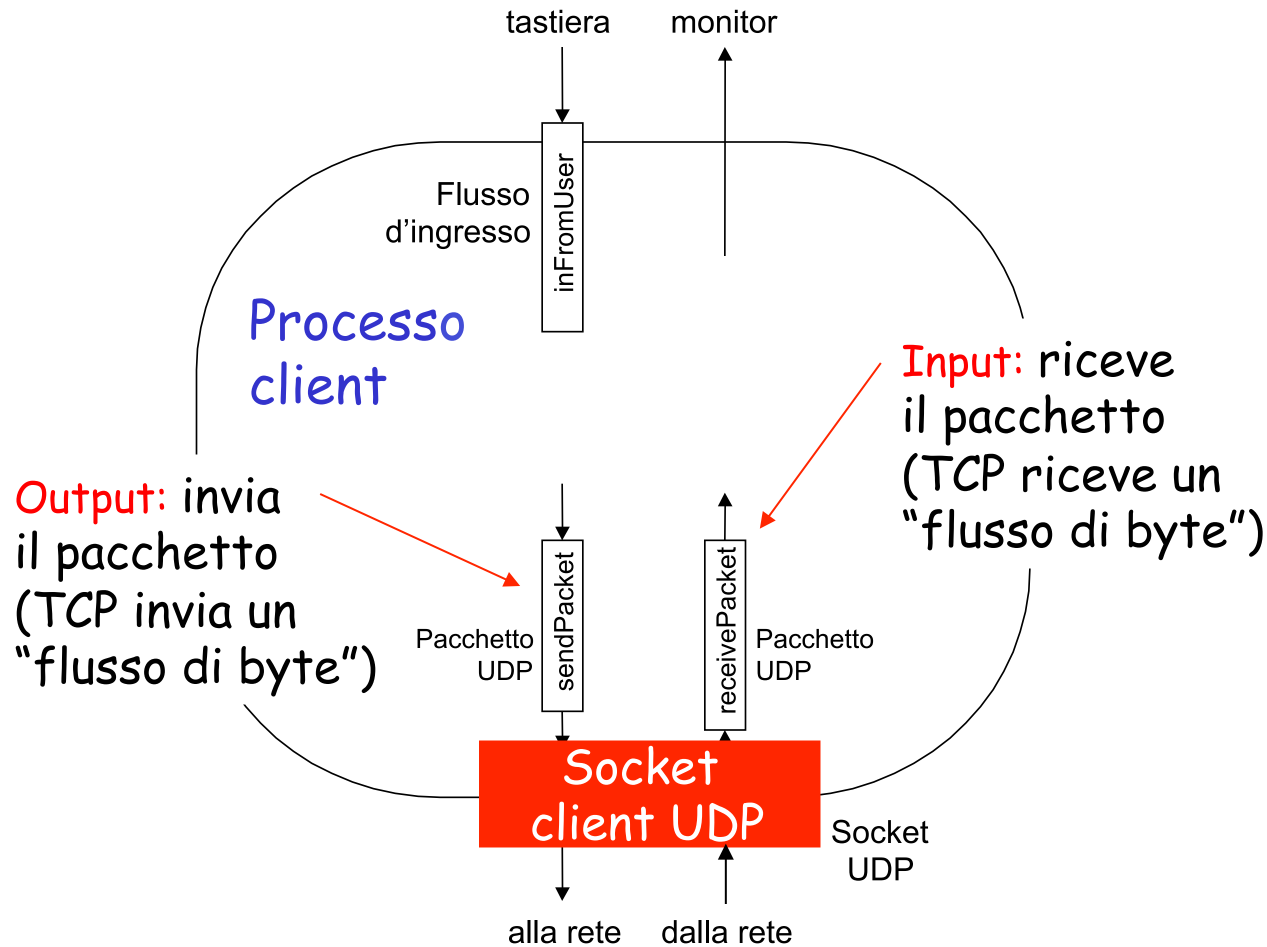
Punto di vista dell'applicazione

UDP fornisce un trasferimento inaffidabile di gruppi di byte ("datagrammi") tra client e server

Interazione delle socket client/server: UDP



Esempio: client Java (UDP)



Esempio: client Java (UDP)

Crea un flusso d'ingresso

Crea un socket client

Traduce il nome dell'host nell'indirizzo IP usando DNS

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

class UDPClient {
    public static void main(String args[]) throws Exception
    {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

Esempio: client Java (UDP), continua

Crea il datagramma con i dati da trasmettere, lunghezza, indirizzo IP, porta

Invia il datagramma al server

Legge il datagramma dal server

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
  new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
   new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
}
```

Il client rimane inattivo fino a quando riceve un pacchetto

Esempio: server Java (UDP)

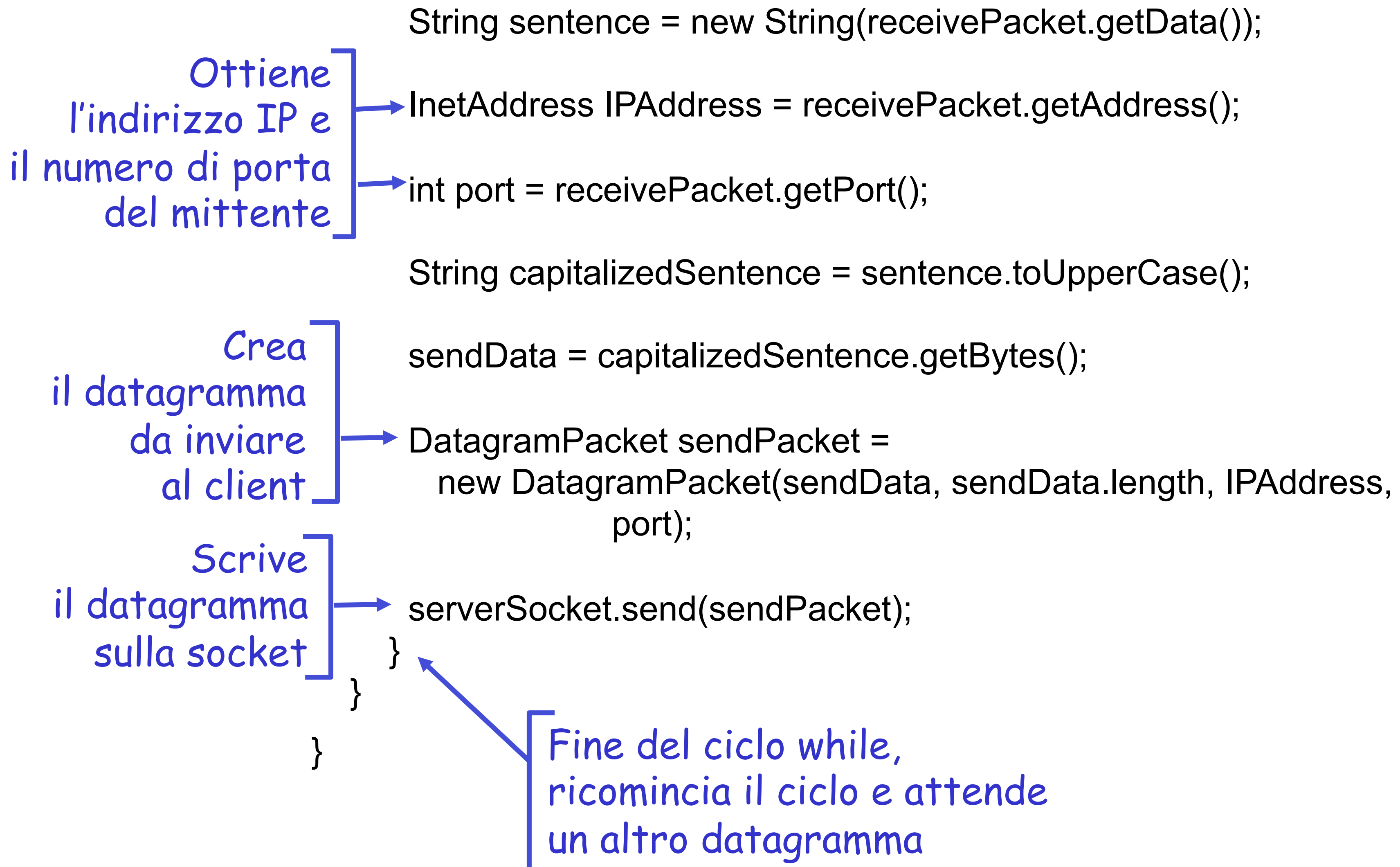
Crea un socket per datagrammi sulla porta 9876

Crea lo spazio per i datagrammi

Riceve i datagrammi

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

class UDPServer {
  public static void main(String args[]) throws Exception
    {

      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {

          DatagramPacket receivePacket =
             new DatagramPacket(receiveData, receiveData.length);
           serverSocket.receive(receivePacket);
```

Esempio: server Java (UDP), continua

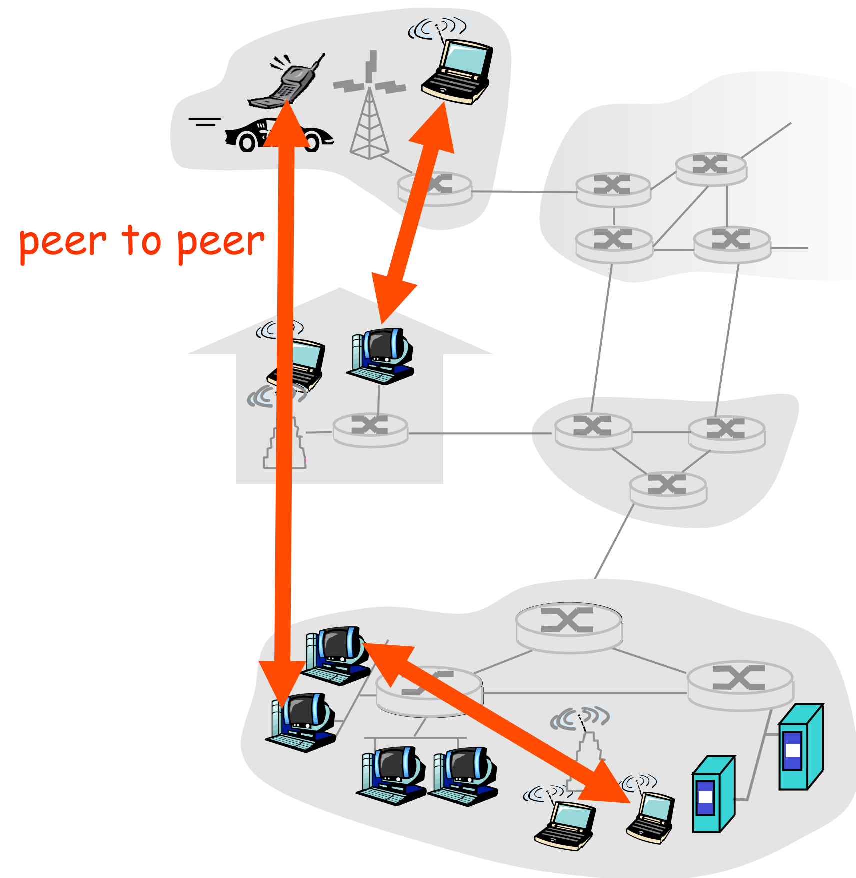


Cenni alle applicazioni peer-to-peer

- Paradigma P2P
- File sharing

Paradigma P2P

- Nato alla fine degli anni 80 è diventato popolare nel periodo 1999-2001 con Napster per condivisione dei file musicali
- Napster chiude a causa di una condanna per violazione diritti di copyright
- Nuovi meccanismi sono emersi
  - Gnutella, Kazaa, BitTorrent



Reti P2P

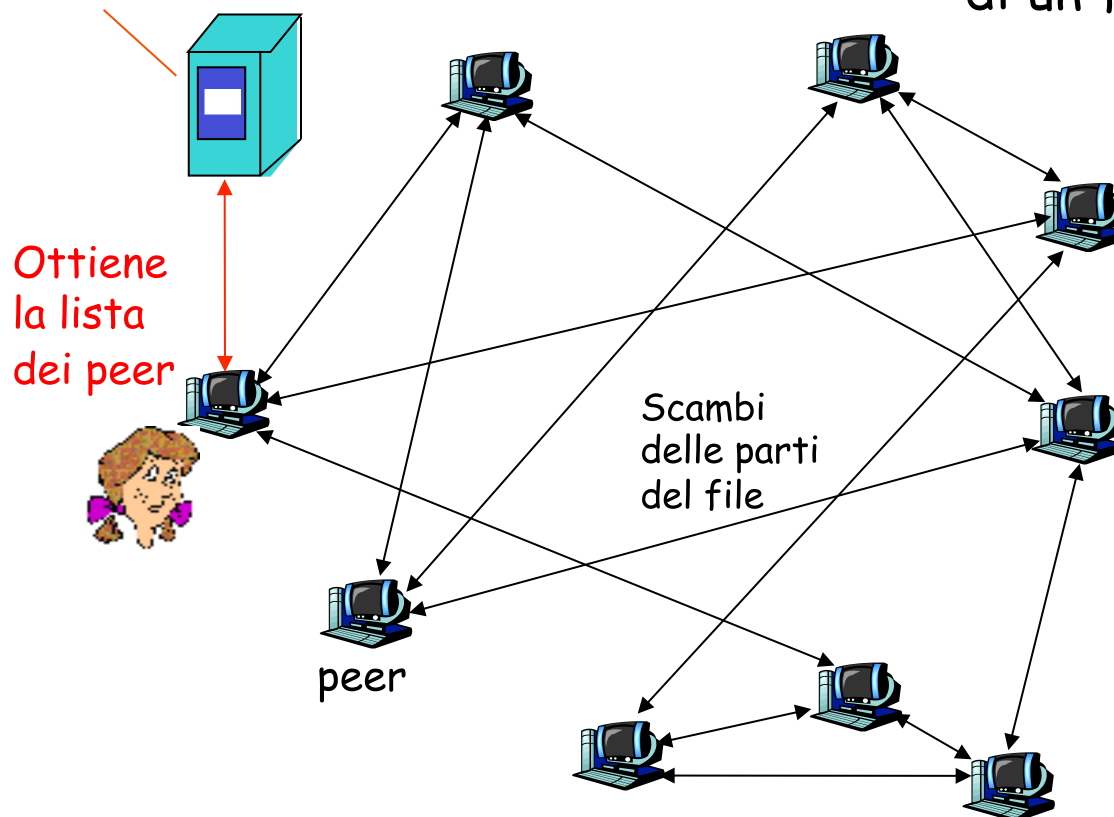
- Gli utenti Internet che intendono condividere le proprie risorse divengono “peer” (pari) e formano una rete.
- Quando uno dei peer nella rete ha un file (per esempio audio o video) da condividere, lo rende disponibile agli altri.
- Chi è interessato può connettersi al computer dove il file è memorizzato e prelevarlo; una volta prelevato, lo può a sua volta rendere disponibile ad altri peer.
- Con l’ingresso di altri peer, il gruppo ha a disposizione sempre più copie.
- Poiché il gruppo di peer può crescere e ridursi dinamicamente, il problema è come poter tenere traccia della disponibilità dei vari file tra i peer.
  - soluzioni centralizzate e decentralizzate (si affidano o meno a un server che mantiene la lista dei peer connessi)

File sharing: BitTorrent

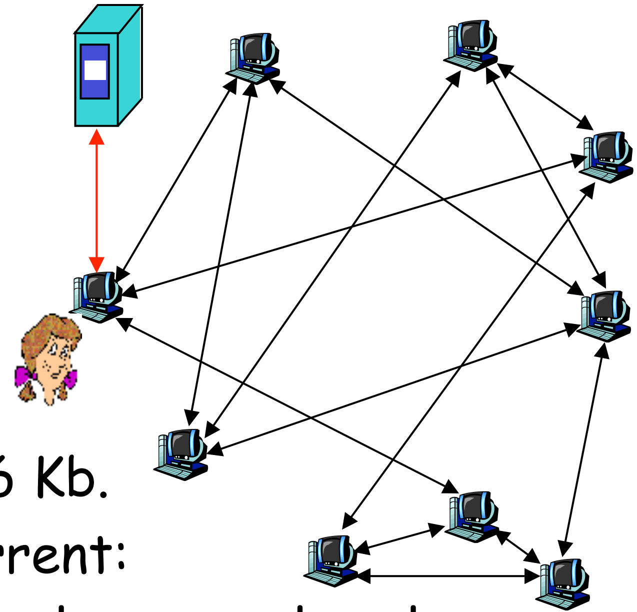
- BitTorrent è un protocollo P2P progettato da Bram Cohen per la condivisione di file particolarmente voluminosi.
- Il termine *condivisione* (sharing) è usato intendendo
  - Non si ha semplicemente un peer che consente a un altro peer di prelevare un intero file
  - Un gruppo di peer collabora per fornire ad altri peer del gruppo una copia del file.
  - La condivisone dei file avviene tramite un processo collaborativo chiamato *torrent*.
  - Ogni peer che partecipa a un torrent preleva parti del file, chiamate **chunk**, da un altro peer e contemporaneamente trasmette i propri chunk ad altri peer che non li possiedono ancora.

Distribuzione di file: BitTorrent

- Distribuzione di file P2P



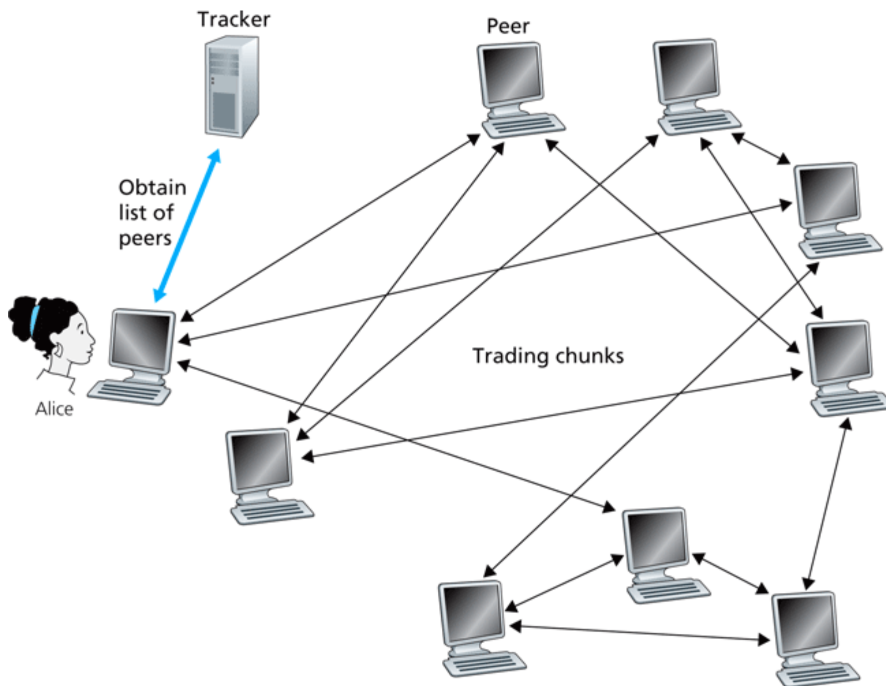
BitTorrent (1)



- Il file viene diviso in parti (*chunk*) da 256 Kb.
- Quando un peer entra a far parte del torrent:
  - non possiede nessuna parte del file, ma le accumula col passare del tempo
  - si registra presso il **tracker** (e periodicamente lo aggiorna) per avere la lista dei peer, e si collega ad un sottoinsieme di peer vicini ("neighbors"), indicati dal tracker
- Mentre effettua il download, il peer carica le sue parti su altri peer.
- I peer possono entrare e uscire a piacimento dal torrent
- Una volta ottenuto l'intero file, il peer può lasciare il torrent (egoisticamente) o (altruisticamente) rimanere collegato.

BitTorrent (2)

- In un dato istante, peer diversi hanno differenti sottoinsiemi del file
- Peer in grado di inviare file a frequenza compatibili tendono a trovarsi…



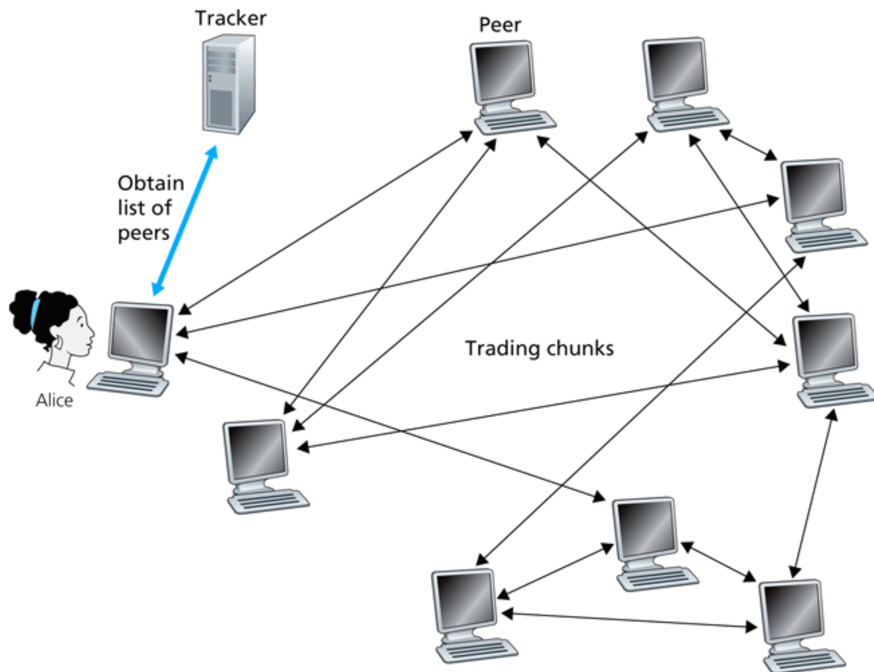
Un nuovo utente (Alice) entra nel Torrent

- Si registra al tracker
- Il tracker seleziona un sottoinsieme di peer (50) e invia la lista dei loro IP a Alice
- Alice prova a stabilire connessioni TCP con i peer nella lista
- I peer che accettano la conessione diventano “neighboring peers” (nella figura sono 3)
- La lista di vicini cambia nel tempo, poichè alcuni possono lasciare il Torrent e altri entrare e chiedere ad Alice la connessione

<u>Invio di richieste</u>

- periodicamente, Alice chiede a ciascun vicino la lista dei chunk che possiede
- Alice seleziona I chunk di cui ha bisogno e invia le richieste ai peer che li possiedono:
  - Adotta la tecnica del *rarest first* (I chunk più rari, ovvero con meno copie, vengono richiesti per primi)

BitTorrent (3)



Invio di risposte

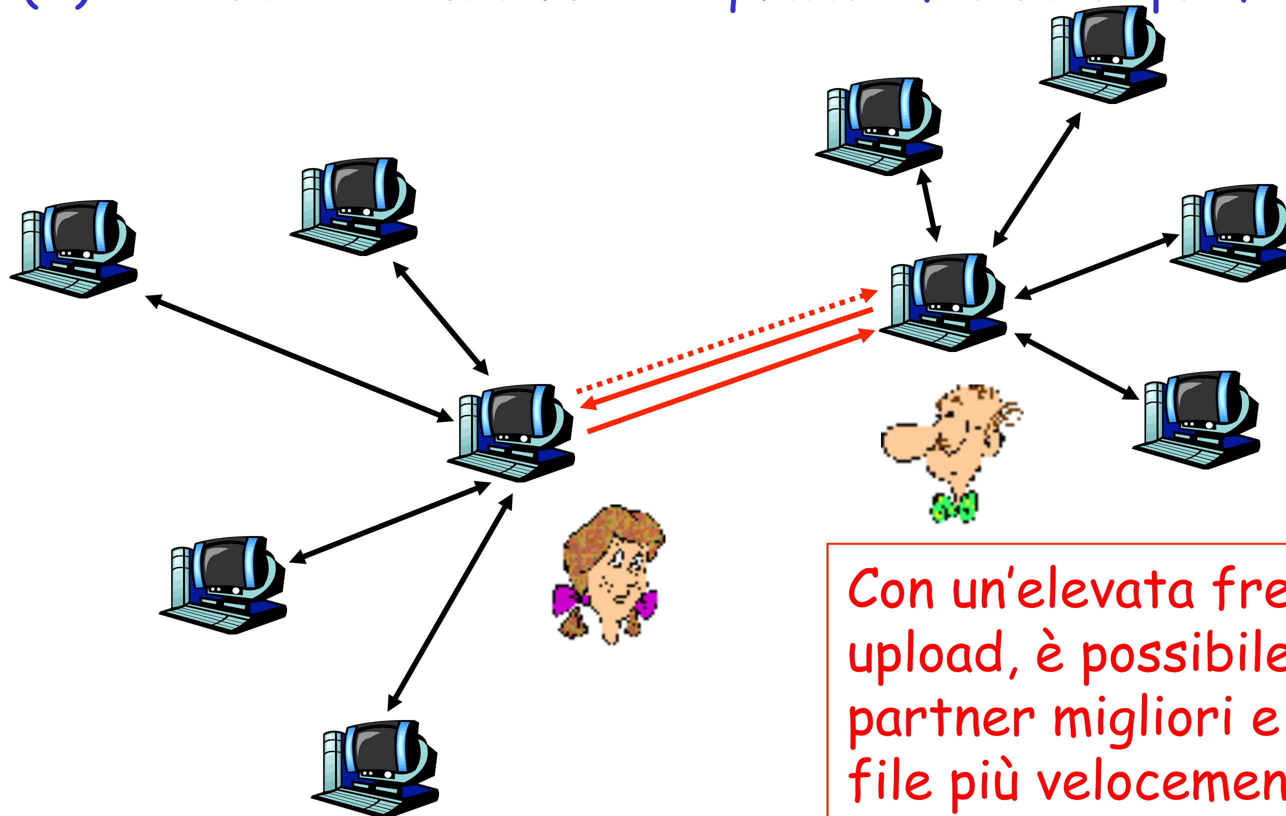
- Per decidere a quali richieste rispondere, BitTorrent usa un algoritmo di trading
- Assegnazione di priorità ai vicini che stanno inviando dati alla frequenza più alta ad Alice
- Determinazione periodica (ogni 10 secondi) dei 4 peer che inviano alla frequenza maggiore
- Alice invia le sue parti ai 4 vicini con rate reciproco
- Ogni 10 secondi viene ricalcolato il rate e modificato eventualmente l'insieme dei top 4
- Ogni 30 secondi viene selezionato casualmente un altro peer, e si inizia a inviargli chunk
    - Il peer appena scelto può entrare a far parte dei top 4
    - A parte i "top 4" e il "nuovo entrato", gli altri peer sono "soffocati", cioè non ricevono nulla

BitTorrent: occhio per occhio

(1) Alice casualmente sceglie Roberto

(2) Alice diventa uno dei quattro fornitori preferiti di Roberto; Roberto ricambia

(3) Roberto diventa uno dei quattro fornitori preferiti di Alice



L'algoritmo di trading elimina il free-riding (sfruttamento)

Con un'elevata frequenza di upload, è possibile trovare i partner migliori e ottenere il file più velocemente!

Riassunto

Lo studio delle applicazioni di rete adesso è completo!

- Architetture delle applicazioni
  - client-server
  - P2P
- Requisiti dei servizi delle applicazioni:
  - affidabilità, ampiezza di banda, ritardo
- Modello di servizio di trasporto di Internet
  - orientato alle connessioni, affidabile: TCP
  - inaffidabile, datagrammi: UDP
- Protocolli specifici:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent
- Interfaccia tra livello applicativo e di trasporto
  - socket

Riassunto

Molto importante: conoscere i *protocolli*

- Tipico scambio di messaggi di richiesta/risposta:
  - il client richiede informazioni o servizi
  - il server risponde con dati e codici di stato
- Formati dei messaggi:
  - intestazioni: campi che forniscono informazioni sui dati
  - dati: informazioni da comunicare
- Controllo o messaggi di dati
  - in banda, fuori banda
- Architettura centralizzata o decentralizzata
- Protocollo senza stato o con stato
- Trasferimento di messaggi affidabile o inaffidabile
- "Complessità nelle parti periferiche della rete"

Interazione con mail server da terminale

```
> telnet mail.uniroma1.it 25
> Trying 151.100.101.67...
  Connected to mail.uniroma1.it.
  Escape character is '^]'.
  220 mail.uniroma1.it ESMTP
> HELO di.uniroma1.it
> 250 mail.uniroma1.it
> MAIL FROM:            <vuoto>
> 250 sender <g> ok
> RCPT TO: <gaia.maselli@gmail.com>
> 250 recipient <gaia.maselli@gmail.com> ok
> DATA
> 354 go ahead
> From: gaia
  To: gaia.maselli@gmail.com
  Subject: prova
  Ciao Gaia!!!
  .
> 250 ok:  Message 158296438 accepted
> QUIT
> 221 mail.uniroma1.it
  Connection closed by foreign host.
```