



*Corso di Laurea Triennale in Informatica
Università degli Studi della Basilicata*

Reti di Calcolatori

Docente: Ugo Erra

ugo.erra+reti@unibas.it

6° Lezione – Programmazione delle socket

Sommario



- **Socket API**
- Esempio di applicazione TCP
- Esempio di applicazione UDP
- Packet Sniffer

Socket API

- Le API per la programmazione delle socket sono state introdotte in BSD4.1 UNIX nel 1981
 - ▣ Per questo motivo solitamente le funzioni di programmazione dei socket vengono chiamate *Berkeley socket API*
- Utilizzando il paradigma client/server
- Offrono due servizi di trasporto
 - ▣ Datagramma inaffidabile
 - ▣ Affidabile, orientata ai byte

Definizione di socket



Una **socket** è un'interfaccia di un *host locale*, *creata dalle applicazioni, controllata dal S.O.* (una “porta”) in cui il processo di un'applicazione può inviare e ricevere messaggi al/dal processo di un'altra applicazione

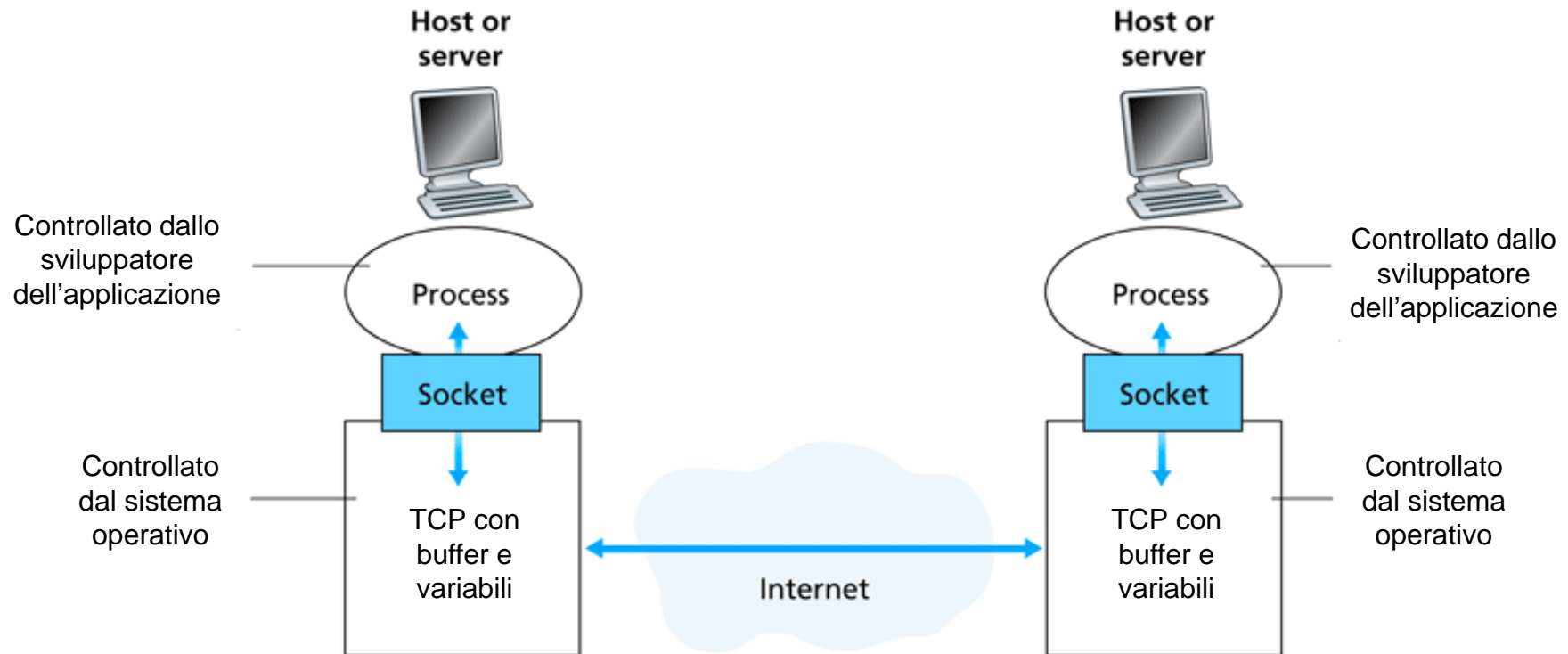
Sommario



- Socket API
- **Esempio di applicazione TCP**
- Esempio di applicazione UDP
- Packet Sniffer

Programmazione delle socket TCP

- ❑ Socket: una porta tra il processo di un'applicazione e il protocollo di trasporto end-end (UCP o TCP)
- ❑ Servizio TCP: trasferimento affidabile di **byte** da un processo all'altro



Programmazione delle socket TCP

- Lato server
 - ▣ Il processo server deve essere in esecuzione
 - ▣ Il server ha creato precedentemente una socket che dà il benvenuto al contatto con il client
- Lato client
 - ▣ Il client crea una socket TCP
 - ▣ Specifica indirizzo IP e il numero di porta del processo server
 - ▣ Quando il client crea la socket: il client TCP stabilisce una connessione con il server TCP
- Al termine della connessione del client, il server TCP crea una nuova 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

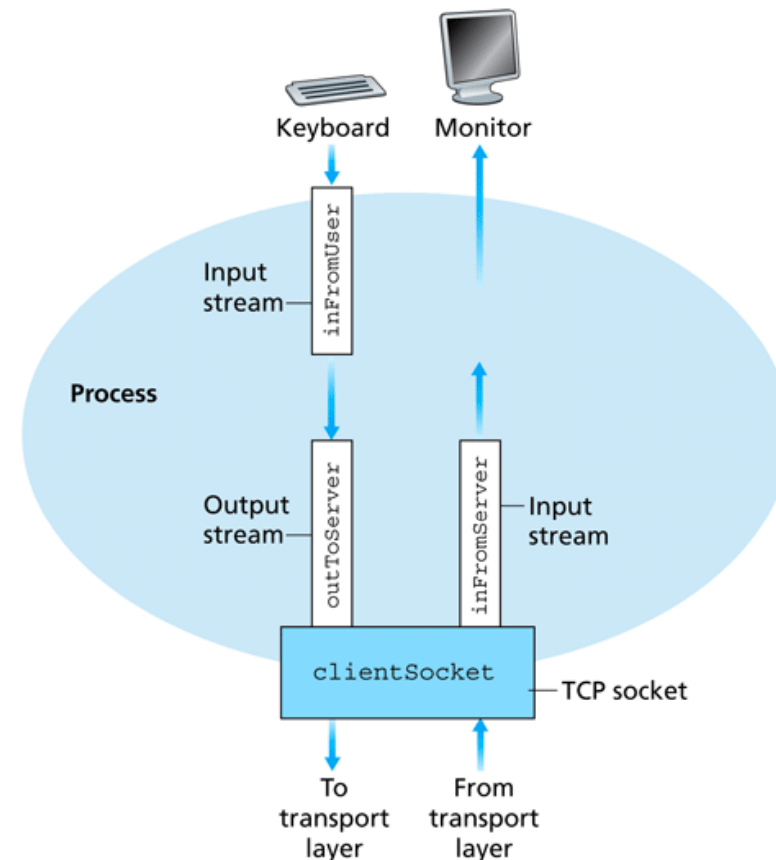
Alcuni termini



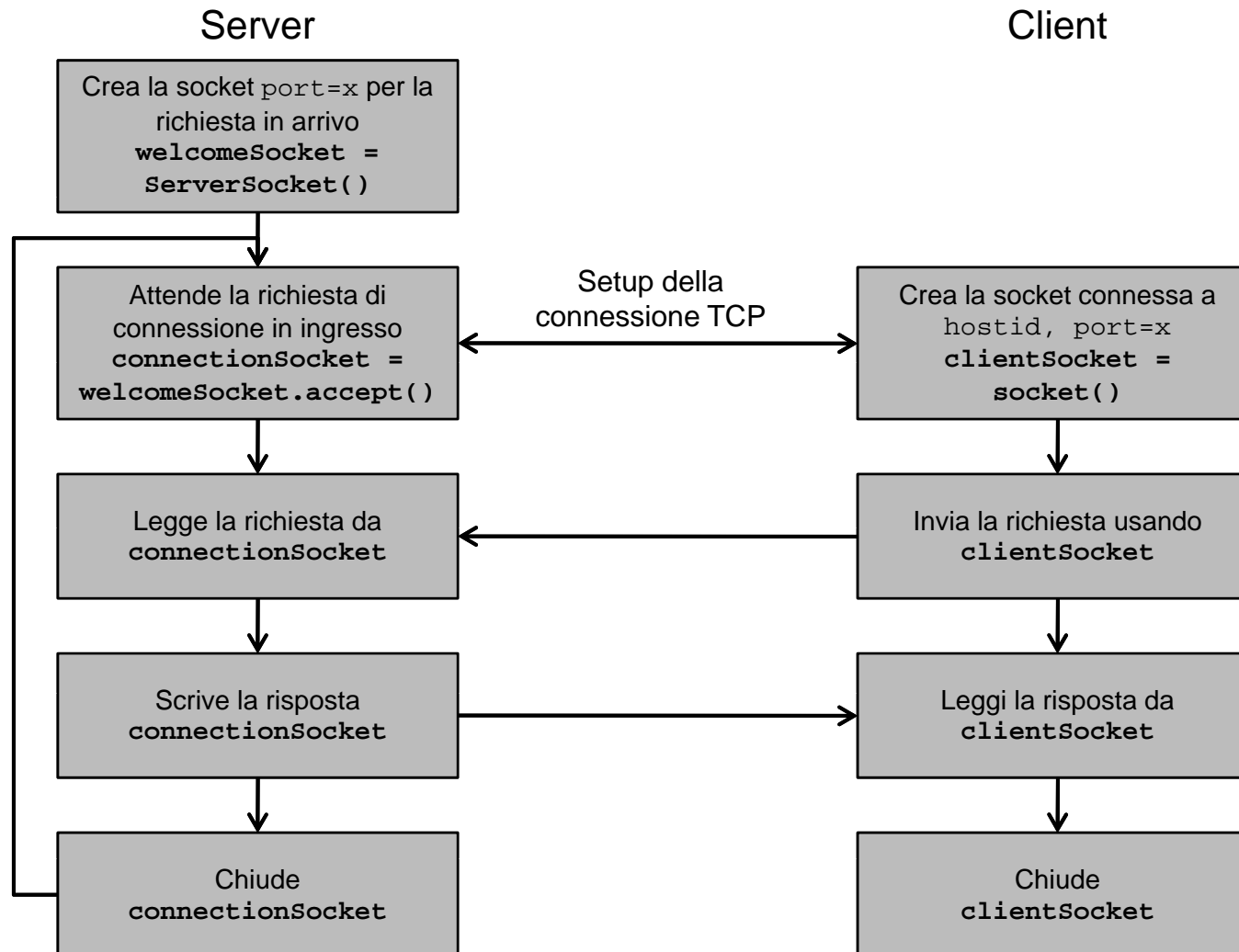
- Un **flusso** (*stream*) è una sequenza di caratteri che fluisce verso/da un processo
- Un **flusso d'ingresso** (*input stream*) è un'origine di dati in input per il processo, ad esempio la tastiera o la socket
- Un **flusso di uscita** (*output stream*) è un'uscita di dati per il processo, ad esempio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

Esempio di applicazione TCP client/server

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



Interazione delle socket TCP client/server



Esempio client Java in TCP - 1

```
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 di ingresso

} Crea una socket client
connessa al server

} Crea un flusso di uscita
collegato al server

Esempio client Java in TCP - 2

```
BufferedReader inFromServer =  
new BufferedReader(new  
InputStreamReader(clientSocket.getInputStream()));  
  
sentence = inFromUser.readLine();  
  
outToServer.writeBytes(sentence + '\n');  
  
modifiedSentence = inFromServer.readLine();  
  
System.out.println("FROM SERVER: " + modifiedSentence);  
  
clientSocket.close();  
}  
}
```

Crea un flusso di ingresso
collegato al socket

Invia una riga al server

Riceve una riga dal server

Esempio server Java in TCP - 1

```
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 una socket di benvenuto sulla porta 6789

Attende sulla socket di benvenuto un contatto con il client

Crea un flusso di ingresso collegato alla socket

Esempio server Java in TCP - 2

```
DataOutputStream outToClient =
```

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

} Crea un flusso di uscita collegato al socket

```
clientSentence = inFromClient.readLine();
```

} Legge la riga dal socket

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

} Scrive la riga sul socket

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

```
}  
}  
}
```

Terma il ciclo while, ricomincia il ciclo in attesa di un nuovo client

Sommario

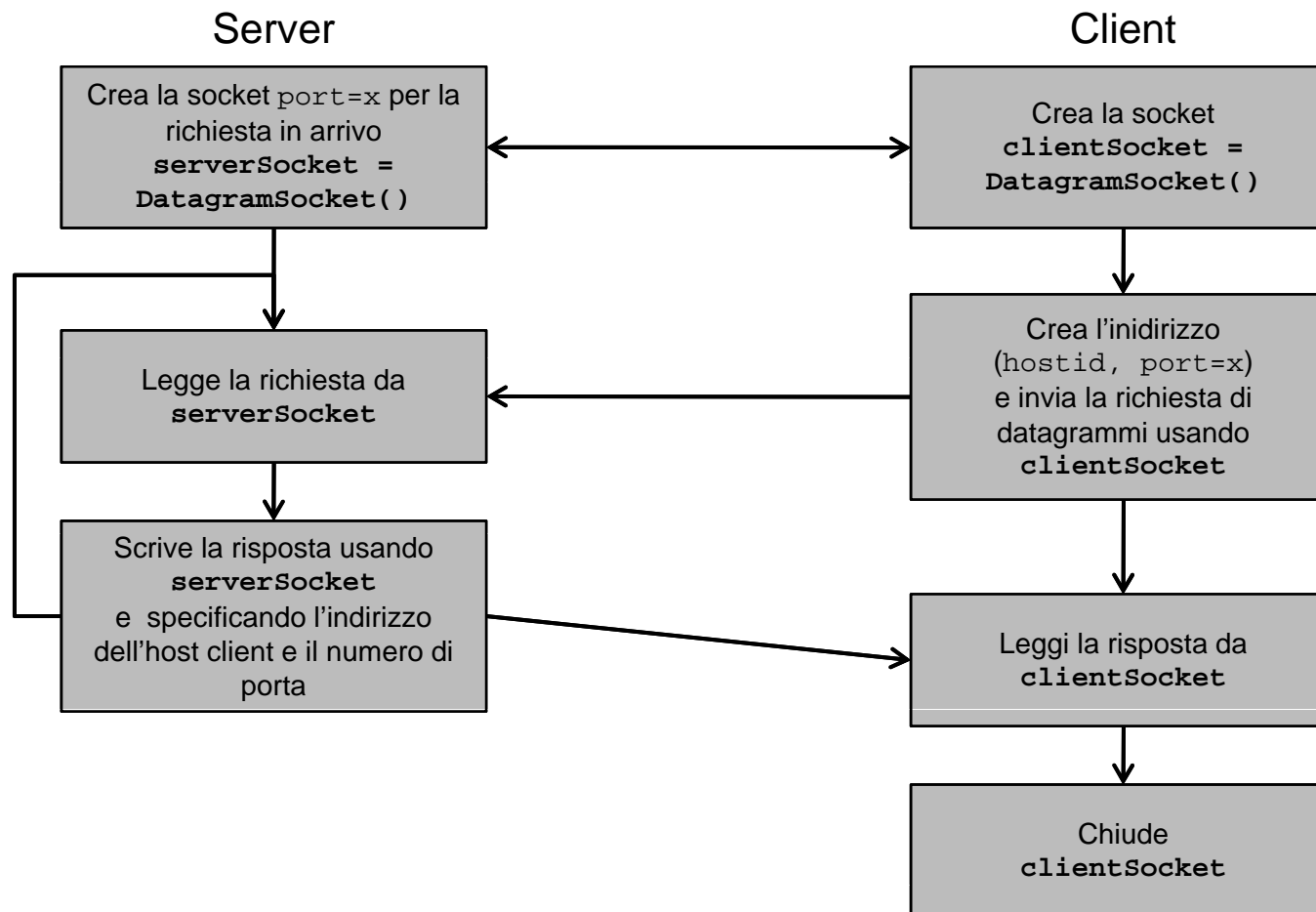


- Socket API
- Esempio di applicazione TCP
- **Esempio di applicazione UDP**
- Packet Sniffer

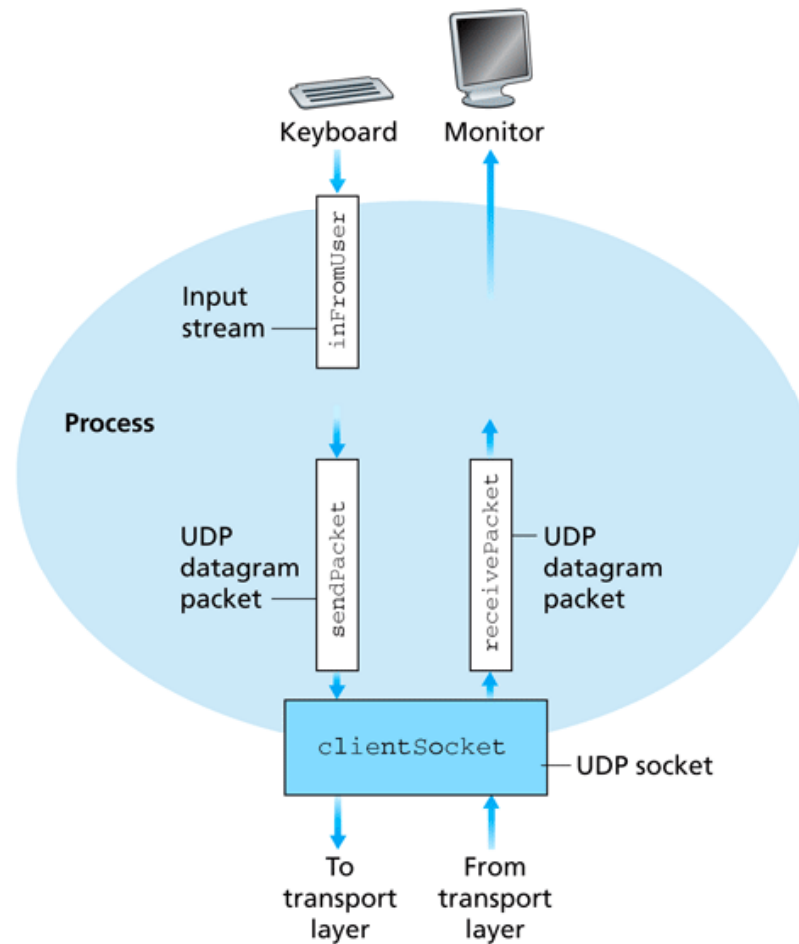
Programmazione delle socket UDP

- In 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

Interazione delle socket UDP client/server



Esempio di applicazione UDP client/server



Esempio client Java in UDP - 1

```
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();  
    }  
}
```

} Crea un flusso di ingresso

} Crea un socket client

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

Esempio client Java in UDP - 2

```
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();  
}  
}
```

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

Invia il datagramma al server

Legge il datagramma dal server

Esempio server Java in UDP - 1

```
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);
            String sentence =
                new String(receivePacket.getData());
        }
    }
}
```

Crea una socket per datagrammi sulla porta 9876

Crea lo spazio per i datagrammi

Riceve i datagrammi

Esempio server Java in UDP - 2

```
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();  
  
String capitalizedSentence = sentence.toUpperCase();  
  
sendData = capitalizedSentence.getBytes();  
  
DatagramPacket sendPacket =  
new DatagramPacket(sendData, sendData.length,  
                    IPAddress, port);  
  
serverSocket.send(sendPacket);  
}  
}  
}
```

Ottiene l'indirizzo IP e la porta del mittente

Crea il datagramma da inviare al client

Scrive il datagramma sulla socket

Terma il ciclo while, ricomincia il ciclo in attesa di un nuovo client

Costruire un semplice server web



- Un server web deve svolgere i seguenti compiti
 - ▣ Gestire una richiesta HTTP
 - ▣ Accettarla e analizzarla
 - ▣ Prendere il file richiesto dal file system del server
 - ▣ Creare un messaggio di risposta HTTP costituito dal file richiesto preceduto da righe di intestazione
 - ▣ Inviare la risposta direttamente al client

Sommario



- Socket API
- Esempio di applicazione TCP
- Esempio di applicazione UDP
- **Packet Sniffer**

Studiare i protocolli di rete

- Due modalità
 - ▣ Simulazione (tool di simulazione, applet, ..)
 - ▣ Osservazione in un ambiente di rete reale

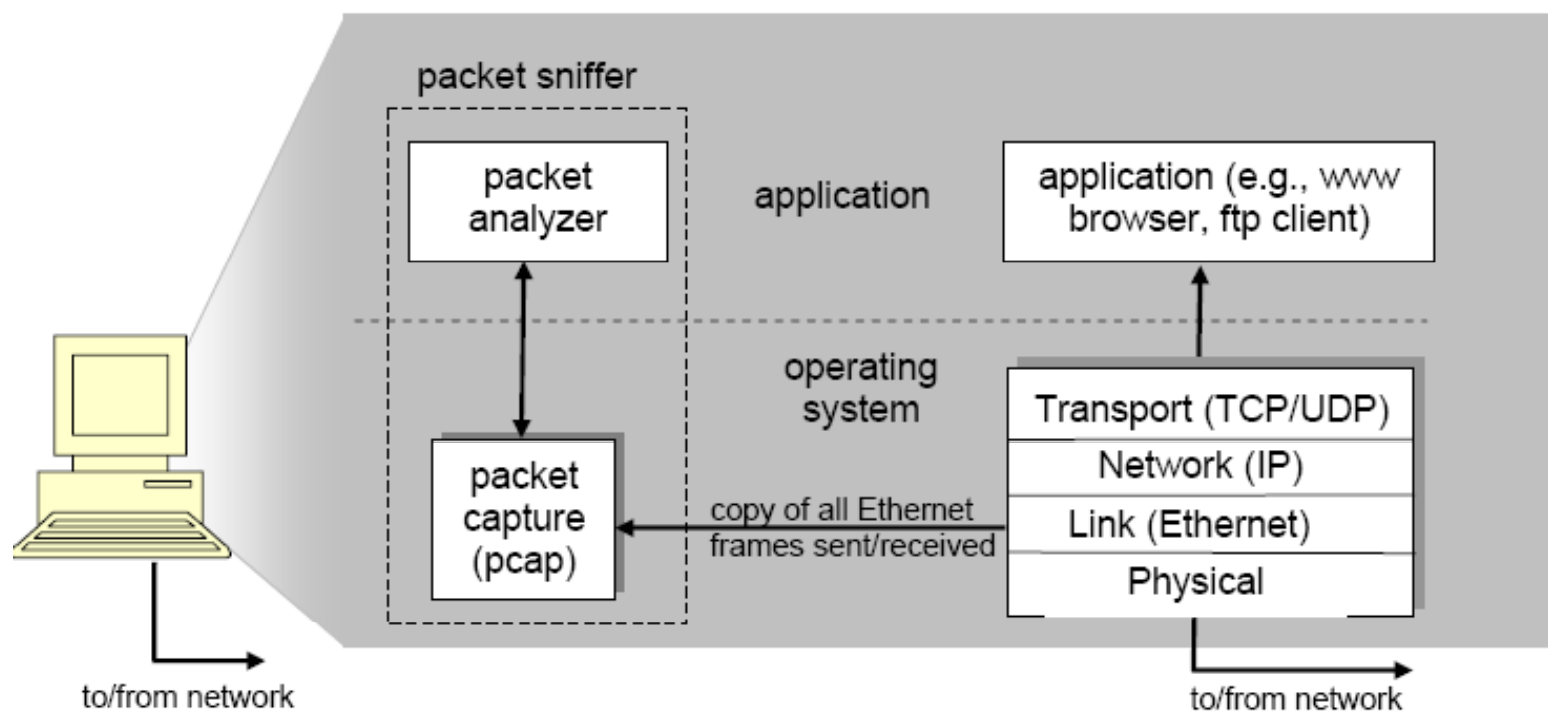
“Tell me and I forget. Show me and I remember. Involve me and I understand.” proverbio cinese

- Vogliamo comprendere come funzionano i protocolli di rete attraverso l'osservazione
 - ▣ La sequenza di messaggi scambiati
 - ▣ Le operazioni del protocollo
 - ▣ Le cause e le conseguenze delle azioni

Packet sniffer

- Sniffing
 - ▣ Attività di intercettazione passiva dei dati che transitano in una rete telematica
 - ▣ Per scopi leciti (e.g. l'individuazione di problemi di comunicazione o di tentativi di intrusione)
 - ▣ Per scopi illeciti (e.g. intercettazione fraudolenta di password o altre informazioni sensibili)
- Packet sniffer
 - ▣ Cattura i messaggi inviati/ricevuti dal proprio computer
 - ▣ Memorizza e/o visualizza i contenuti dei vari campi del protocollo nei messaggi catturati (packet analyzer)
 - ▣ Non invia mai pacchetti di per sé ovvero è passivo

Struttura Packet Sniffer



Packet analyzer

- Visualizza i contenuti di tutti i campi presenti in un messaggio di un protocollo
- Comprendere la struttura di tutti i messaggi scambiati tra i protocolli:
 - ▣ Comprende il formato degli Ethernet frame
 - ▣ Identifica il datagramma IP
 - ▣ Estrae il segmento TCP dal datagramma IP
 - ▣ Estrae il messaggio HTTP dal segmento TCP
 - ▣ Analizza il messaggio HTTP

Wireshark

command menus

display filter specification

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.46	128.121.50.122	TCP	1163 > http [SYN] seq=0 Len=0 MSS=1460
2	0.127987	128.121.50.122	192.168.1.46	TCP	http > 1163 [SYN, ACK] Seq=0 Ack=1 Win=57
3	0.128232	192.168.1.46	128.121.50.122	TCP	1163 > http [ACK] Seq=1 Ack=1 Win=65535
4	0.153700	192.168.1.46	128.121.50.122	HTTP	GET /news/ HTTP/1.1
5	0.329641	128.121.50.122	192.168.1.46	TCP	[TCP segment of a reassembled PDU]
6	0.330426	128.121.50.122	192.168.1.46	HTTP	[TCP Previous segment lost] Continuation
7	0.330467	192.168.1.46	128.121.50.122	TCP	1163 > http [ACK] Seq=657 Ack=1082 Win=68
8	0.342092	128.121.50.122	192.168.1.46	TCP	[TCP Retransmission] [TCP segment of a re

listing of captured packets

details of selected packet header

```

* Frame 4 (710 bytes on wire, 710 bytes captured)
  * Ethernet II, Src: Netgear_61:BE:6D (00:09:5B:61:BE:6D), Dst: westellT_9F:92:89 (00:0F:DB:9F:92:89)
  * Internet Protocol, Src: 192.168.1.46 (192.168.1.46), Dst: 128.121.50.122 (128.121.50.122)
  * Transmission Control Protocol, Src Port: 1163 (1163), Dst Port: http (80), Seq: 1, Ack: 1, Len: 656
  * Hypertext Transfer Protocol
    GET /news/ HTTP/1.1\r\n
      Host: www.wireshark.org\r\n
      User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-us; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4\r\n
      Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5\r\n
      Accept-Language: en-us,en;q=0.5\r\n
      Accept-Encoding: gzip,deflate\r\n
      Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
      keep-alive: 300\r\n
      connection: keep-alive\r\n
      Referer: http://www.wireshark.org/faq.html\r\n
      cookie: __utma=87653150.62471437.1181007382.1181007382.1181168042.2; __utmz=87653150.1181007382.1.1.1.utm
    \r\n
  
```

packet content hexadecimal and ASCII

```

0000 00 0F DB 9F 92 89 00 09 5B 61 BE 6D 08 00 45 00  .... [a.m...E.
0010 02 B8 0F 25 40 00 80 06 74 31 C0 A8 01 2E 80 79  ...88...TQ....y
0020 32 7A 04 8B 00 10 ED BC 8E 1B 4E C0 F1 18 30 18  22...P...N...P.
0030 FF FF 77 74 00 00 47 43 54 20 2F 6A 65 77 73 2F  ..wt..GCT/news/
0040 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F 73 74 3A  HTTP/1.1..Host:
0050 20 77 77 77 2E 77 69 72 65 73 68 61 72 6B 2E 6F  www.wireshark.o
0060 72 67 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20  rg..User-Agent:
0070 4D 6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 57 69 6E  Mozilla/5.0 (win
0080 64 6F 77 73 3B 20 55 2B 20 57 69 6E 64 6F 77 73  dows; U; windows
0090 20 4E 54 20 35 2E 31 3B 20 65 6E 2D 55 53 3B 20  NT 5.1; en-us;
00A0 72 76 3A 31 2E 38 2E 31 2E 34 29 20 47 65 63 6B  rv:1.8.1.4) geck
00B0 6F 2F 32 30 30 37 30 35 31 35 20 46 69 72 65 66  o/20070515 Firef
  
```