Network API

Application Programming Interface → interface a liv. applicativo

- Interface exported by the network
- Since most network protocols are implemented in software and nearly all computer systems implement their network protocols as part of the operating system, when we refer to the interface "exported by the network", we are generally referring to the interface that the OS provides to its networking subsystem

fino a liv. 4 implementato in S.O., sopra a liv. utente

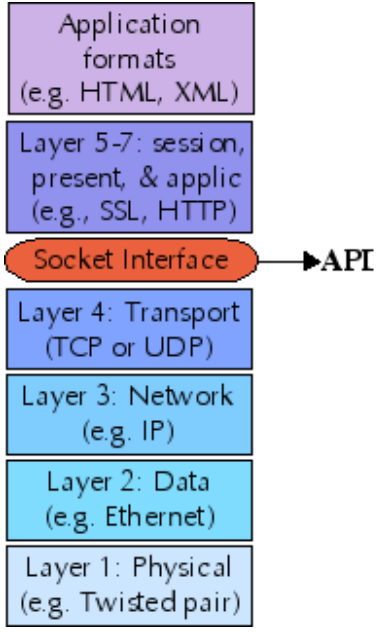
- The interface is called the **network Application Programming Interface (API)** => SOCKET

Application Programming Interface (Sockets)

- **Socket Interface** was originally provided by the Berkeley distribution of Unix
  - Now supported in virtually all operating systems, and found in any programming language
  - Some languages implement higher-level interfaces on top of sockets
- Each protocol provides a certain set of services, and the API provides a syntax by which those services can be invoked in this particular OS

Socket

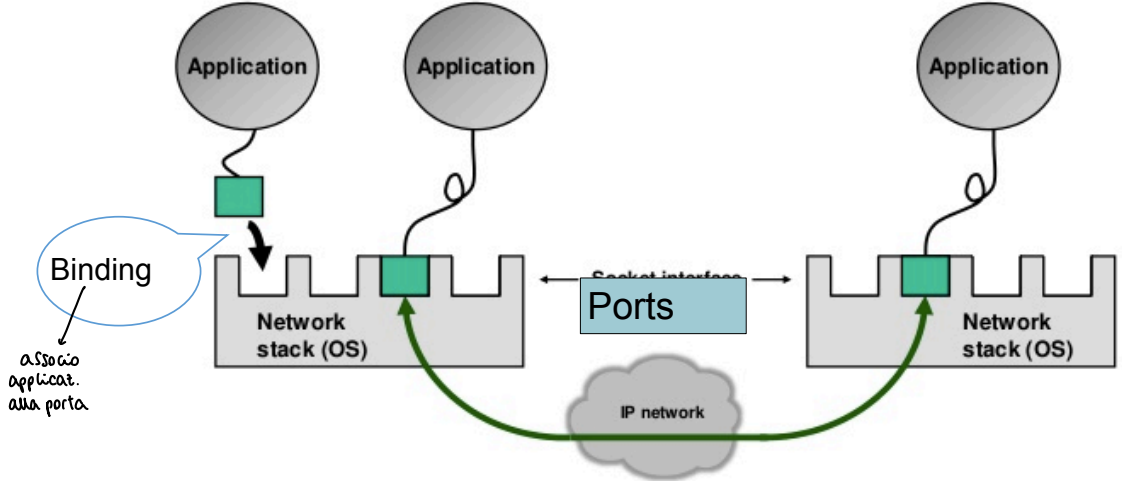
- What is a socket?
  - The point where a local application process attaches to the network
  - An interface between an application and the network → tubo tra le applicazioni
- The interface defines operations for
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket



non sono implementati a liv. s.o, sono a livello di programmaz. es. librerie

Stream oriented Sockets

- After connection, the channel is bidirectional.



Socket creation

At creation the application specifies:

- Socket Family
  - PF_INET denotes the Internet family
  - PF_UNIX denotes the Unix pipe facility
  - PF_PACKET denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack)
- Socket Type
  - SOCK_STREAM is used to denote a byte stream: reliable, ordered, connection-oriented
  - SOCK_DGRAM denotes a message oriented service, usually unreliable, such as that provided by UDP

Creating a Socket

```
int sockfd = socket(address_family, type, protocol);
```

↳ syscall

- The socket number returned is the socket descriptor for the newly created socket

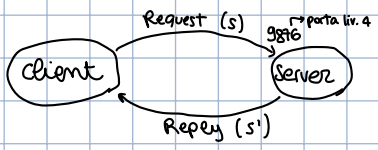
```
int sockfd = socket (PF_INET, SOCK_STREAM, 0);
int sockfd = socket (PF_INET, SOCK_DGRAM, 0);
```

SOCK_STREAM → TCP
SOCK_DGRAM → UDP

- The combination of PF_INET and SOCK_STREAM implies TCP

MAIUSCOLATORE s → s'
↑
maiuscolo

UDP:

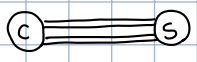


ogni volta che si fa una request o reply si effettua un nuovo collegamento → CONNECTIONLESS

SERVER
receive è bloccante
send non è bloccante

noi prenotiamo la porta 9876 sul server e sul client. Ogni SRV reale ha le sue porte, es. HTTP porta 80 → lo troviamo nel file /etc/services

TCP:



crea il canale di connessione, FULL DUPLEX, e poi ci si scambia i messaggi
due canali unidirezionali

	Servizio	Esempio
Orientato alla connessione	Flusso affidabile di messaggi	Sequenza di pagine
	Flusso affidabile di byte	Login remoto
	Connessione inaffidabile	Voce digitalizzata
Senza connessione	Datagram inaffidabile	Posta elettronica spazzatura
	Datagram con conferma	Posta raccomandata
	Request-reply	Interrogazione di un database

confini dei messaggi vengono preservati e mai uniti
↑
→ MESSAGE SEQUENCE
→ BYTE STREAM
↓
flusso continuo, limiti non rispettati

Client-Server Model with TCP

- Server
  - Passive open
  - Prepares to accept connection, does not actually establish a connection

tubo bidirezionale

- Server invokes

```
int bind (int socket, struct sockaddr *address, int addr_len)
int listen (int socket, int backlog)
int accept (int socket, struct sockaddr *address, int *addr_len)
```

Client-Server Model with TCP

- Bind → associa l'IP del SRV alla sua porta
  - Binds the newly created socket to the specified address i.e. the network address of the local participant (the server)
  - Address is a data structure which combines IP and port
- Listen → a quanti client al max può servire il SRV → # max di connessioni
  - Defines how many connections can be pending on the specified socket
- Accept
  - Carries out the passive open ↳ apertura passiva, ovvero il server si apre e rimane in attesa di richieste
  - Blocking operation
    - Does not return until a remote participant has established a connection
    - When it does, it returns a new socket that corresponds to the new established connection and the address argument contains the remote participant's address

Client-Server Model with TCP

- Client
  - Application performs active open
  - It says who it wants to communicate with

apertura attiva, è il client a richiedere di collegarsi al server

- Client invokes

```
int connect (int socket, struct sockaddr *address,
             int addr_len)
```

- Connect
  - Does not return until TCP has successfully established a connection at which application is free to begin sending data
  - Address contains remote machine's address

Client-Server Model with TCP

- In practice
  - The client usually specifies only remote participant's address and let's the system fill in the local information
  - Whereas a server usually listens for messages on a well-known port
  - A client does not care which port it uses for itself, the OS simply selects an unused one

nel client io fisso la porta del srv
e poi al client viene assegnata una porta random
nell' UDP ogni volta che faccio la request gli viene assegnata una porta

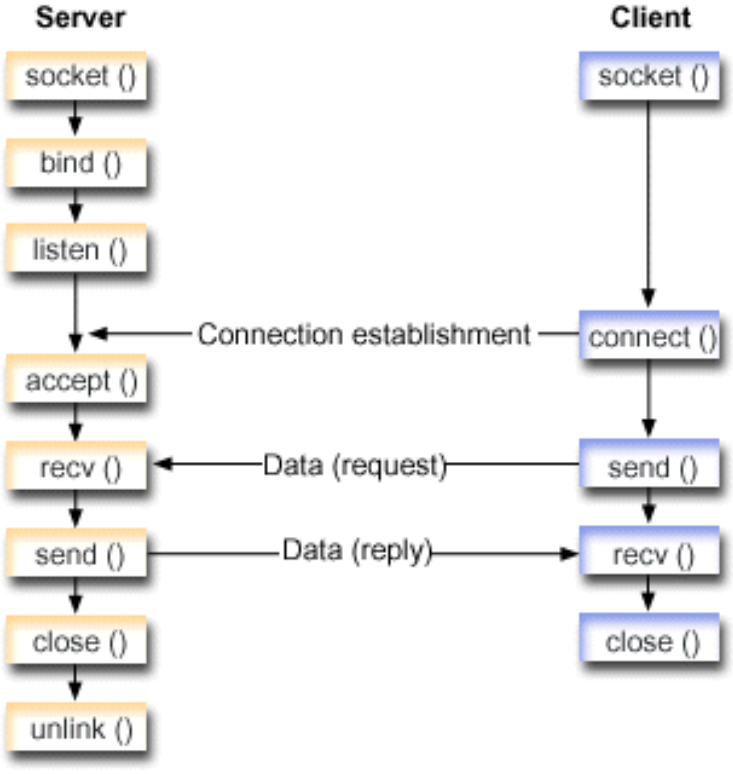
Client-Server Model with TCP

- Once a connection is established, the application process invokes two operation

```
int send (int socket, char *msg, int msg_len, int flags)
int recv (int socket, char *buff, int buff_len, int flags)
```

- (but also other operations are available)

Client-Server Model with TCP



Example Application: Client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 5432
#define MAX_LINE 256

int main(int argc, char * argv[])
{
        FILE *fp;
        struct hostent *hp;
        struct sockaddr_in sin;
        char *host;
        char buf[MAX_LINE];
        int s;
        int len;
        if (argc==2) {
                host = argv[1];
        }
        else {
                fprintf(stderr, "usage: simplex-talk host\n");
        exit(1);
        }
        /* translate host name into peer's IP address */
        hp = gethostbyname(host);
        if (!hp) {
                fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
                exit(1);
        }
        /* build address data structure */
        bzero((char *)&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
        sin.sin_port = htons(SERVER_PORT);
        /* active open */
        if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
                perror("simplex-talk: socket");
                exit(1);
        }
        if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
                perror("simplex-talk: connect");
                close(s);
                exit(1);
        }
        /* main loop: get and send lines of text */
        while (fgets(buf, sizeof(buf), stdin)) {
                buf[MAX_LINE-1] = '\0';
                len = strlen(buf) + 1;
                send(s, buf, len, 0);
        }
}
```

Example Application: Server

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE 256

int main()
{
        struct sockaddr_in sin;
        char buf[MAX_LINE];
        int len;
        int s, new_s;
        /* build address data structure */
        bzero((char *)&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = htons(SERVER_PORT);

        /* setup passive open */
        if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
                perror("simplex-talk: socket");
                exit(1);
        }
        if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
                perror("simplex-talk: bind");
                exit(1);
        }
        listen(s, MAX_PENDING);
        /* wait for connection, then receive and print text */
        while(1) {
                if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
                perror("simplex-talk: accept");
                exit(1);
        }
        while (len = recv(new_s, buf, sizeof(buf), 0))
                fputs(buf, stdout);
                close(new_s);
        }
}
```

Performance

Performance

più pompiamo su una rete più c'è ritardo

- Network performance is measured in two fundamental ways:
  - Bandwidth, but more properly **throughput**: Number of bits per second (bits/s or b/s) that can be transmitted over a communication link
    (↑ capacità del canale)
  - **Delay**, or **latency**: the time between sending a message, and its reception
- 1 Kbps: 1 x 10^3 bits/second = ~1x2^{10} bits/sec
- 1 Mbps: 1 x 10^6 bits/second = ~1x2^{20} bits/sec
- 1 Gbps: 1 Kbps · 1 Mbps = ~1x2^{30} bits/sec
- 1 x 10^{-6} seconds to transmit each bit or imagine that a timeline, now each bit occupies 1 micro second space.
- On a 2 Mbps link the width is 0.5 micro second. ($\frac{2Mb}{1s}$) $\Rightarrow 2\frac{Mb}{s} \leftrightarrow$ 1 bit ogni 0,5s $\Rightarrow \frac{1}{2\cdot10^6}\left[\frac{1}{bit/s}\right] = 0{,}5\cdot10^{-6}\left[\frac{s}{bit}\right]$ (↳ lasso di tempo in cui trasmetto un bit)
- Smaller the width more will be transmission per unit time

Performance

- Notice that the term **bandwidth** is quite confusing
- When you talk to informatics: throughput (bit per seconds)
- But when you talk to engineers and physicists: width of the frequency band (in Hertz: Hz = 1/s) used in the communication
  - This is the correct definition.
  - So we will use *throughput* for the number of bits transmitted in one second, and *bandwidth* for the real bandwidth.

↳ intervallo delle frequenza → AMPIEZZA dello SPETTRO

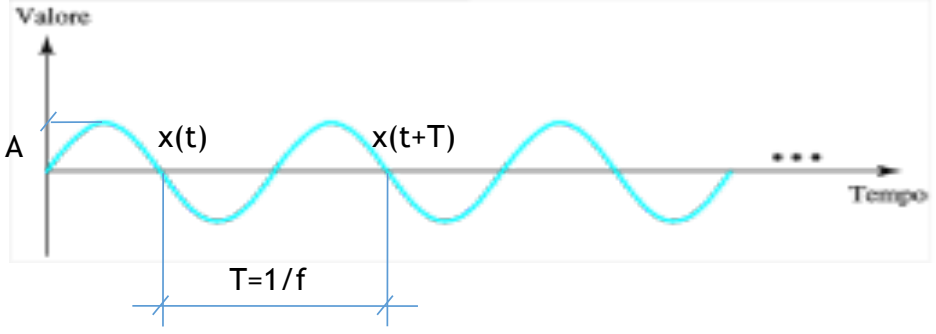
gli Hz → maggiore è il numero più veloce è il processore

periodo [Hz] ↘
frequenza [s] ↗ sono INVERSAMENTE PROP.

What is the frequency band?

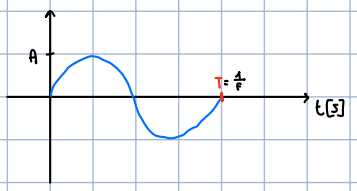
→ la maggior parte dei dispositivi usa onde

- Basic, stationary (i.e., periodics) signals are **sinusoids** → onde formate da SINUSOIDI
- A sinusoid is defined by three values (A,f,φ):
  - Amplitude A [some value unit, e.g. volt]
  - Frequency f [Hertz = 1/s]
  - Phase φ [rad] → quanto è spostata avanti e indietro



Sinusoid $x(t) = A \sin(2\pi ft+\varphi)$

PASSAGGIO DA ONDA a SPETTRO



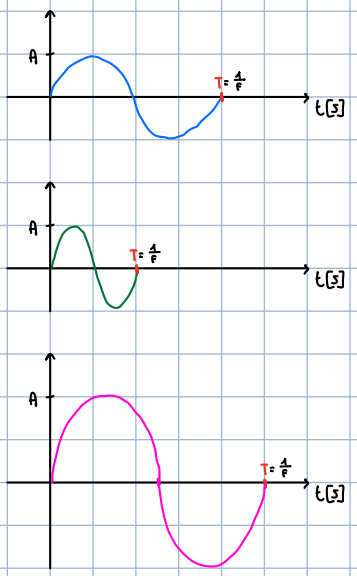
=> calcolo $f = \frac{1}{T}$

↳ $\frac{1}{s}$ → qualcosa al secondo

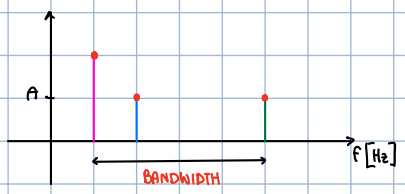
=> ovviamente nel tempo le onde non sono sempre uguali perciò avrò frequenze diverse

l'insieme delle frequenze si chiama spettro

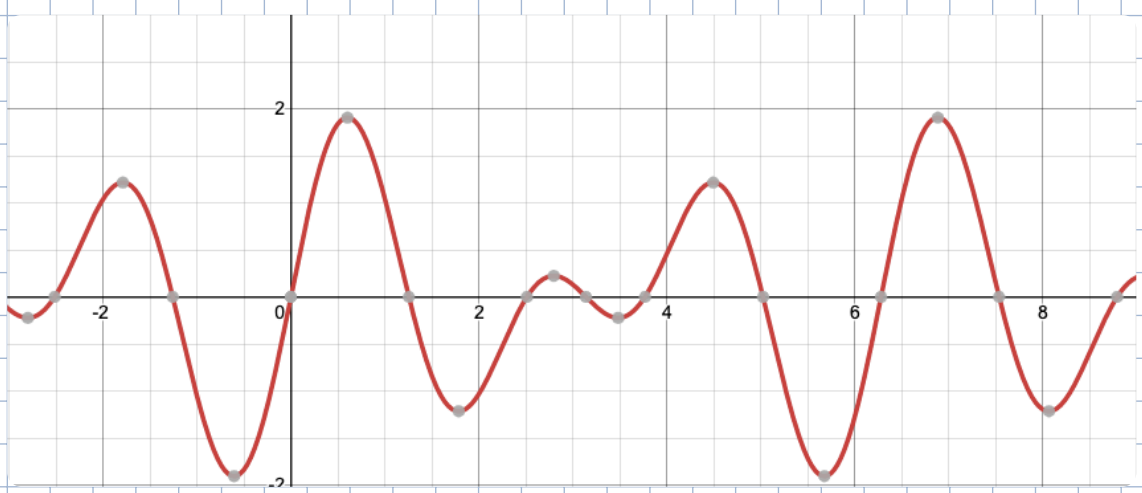
es.



=>



es. $\sin(2x) + \sin(3x)$

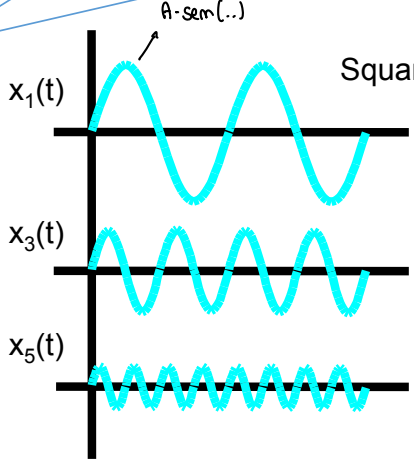


What is the fr

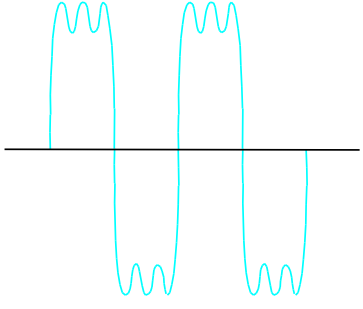
Every "stationary" (=periodic) signal x(t) can be seen as a superposition of (phased) sinusoids:

$$x(t) = \Sigma_i\, x_i(t) = \Sigma_i\, A_i \sin(2\pi_i\, ft + \varphi_i)$$

Jean Baptiste Joseph Fourier



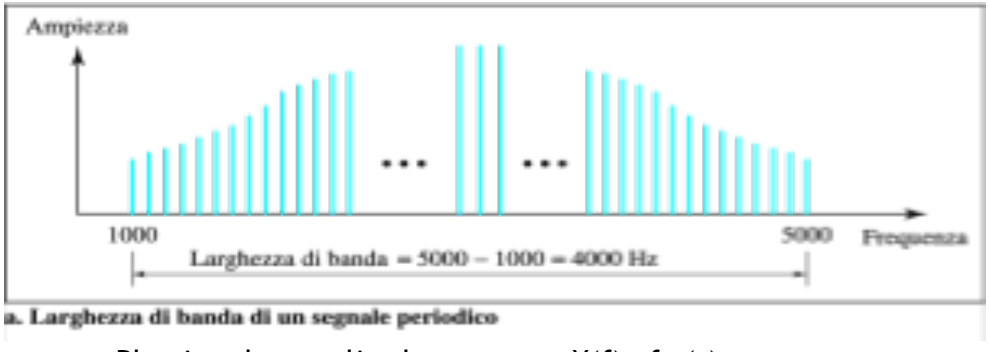
Square wave approximation $x(t) = x_1(t) + x_3(t) + x_5(t)$



Bandwidth as signal spectrum

al posto che rappresentare il segnale rispetto al TEMPO lo faccio rispetto alla FREQUENZA

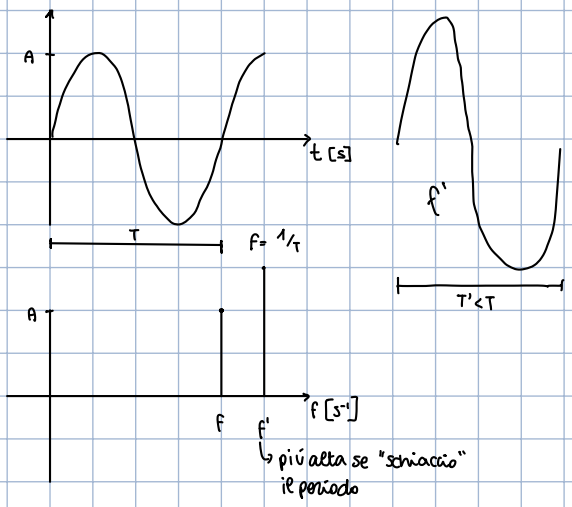
- The **spectrum** X(f) is the representation of x(t) in the space of frequencies
  - For each frequency f, X(f) represents the amplitude (and the phase) of the component of frequency f



Plotting the amplitude spectrum X(f) of x(t)

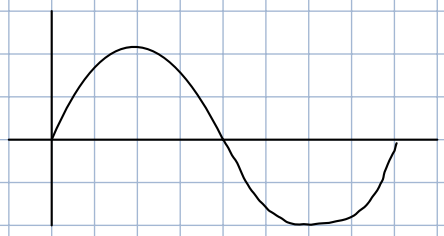
- The difference between the Maximum and the minimum frequencies with non-zero amplitude is the **bandwidth**

TRASFORMATA di FOURIER: passare da tempo a frequenza



=> ovviamente nel tempo le onde non sono sempre uguali perciò avrò frequenze diverse

l'insieme delle frequenze si chiama spettro

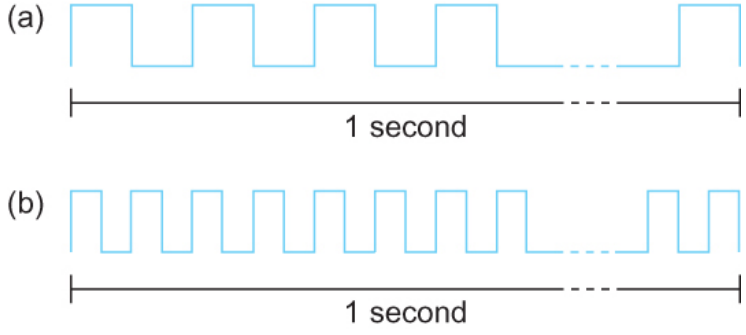


Bandwidth as transmission speed

- Bits transmitted at a particular bandwidth can be regarded as having some width (i.e. temporal extension):

(a) bits transmitted at 1Mbps (each bit 1 μs wide);

(b) bits transmitted at 2Mbps (each bit 0.5 μs wide).



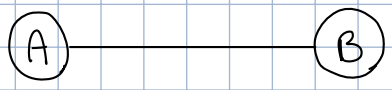
Performance

- Latency = Propagation + Transmit + queue
  where
  - Propagation = distance/speed of light (in the specific mean)
    - E.g. 3.0×10^8 m/s in a vacuum, 2.3×10^8 m/s in a cable, and 2.0×10^8 m/s in a fiber
  - Transmit = size/bandwidth
  - Queue = system-dependent

- One bit transmission => propagation is important
- Large bytes transmission => bandwidth is important (see Figure 1.17)

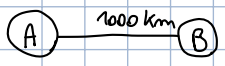
per convertire da km/h in m/s dividi per 3.6 $\Rightarrow \frac{1000}{3600}$

t → tempo di TRASMISSIONE



una volta che faccio l'impulso ci vuole del tempo prima che arrivi dall'altra parte → TEMPO di PROPAGAZIONE

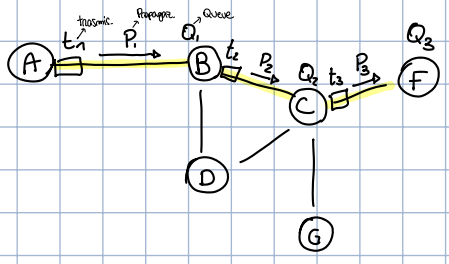
ho la fibra lunga 1000 km → 10^6 m e velocità = $2 \cdot 10^8$ m/s $\quad v = \frac{s}{t} \quad t = \frac{s}{v}$



$t_{PROP} = \frac{\text{Spazio}}{\text{Vel}} = \frac{10^6 m}{2 \cdot 10^8 m/s} = 5 ms$

Quando i pacchetti arrivano si fermano in una coda

spedisco da A a F



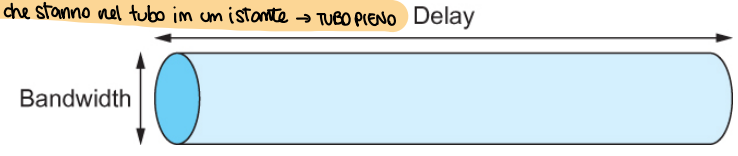
LIMITI: - tempo propagaz: FISICA
- tempo trasmissione: come trasmettiamo il segnale
- tempo queue: dipende dalla capacità dei dispositivi → rischio congestione (PACKET DROP)

FISSI

Delay x Bandwidth

- We think the channel between a pair of processes as a hollow pipe
- Delay assimilable to propagation time
- Delay: length of the pipe.
- Bandwidth: width of the pipe
- Delay of 50 ms and bandwidth of 45 Mbps:
  50×10^{-3} seconds $\times$ 45×10^{6} bits/second =
  $= 2.25 \times 10^{6}$ bits = 280 KB data
- This is the data which can be contained in the "pipe", in flight

↳ QUANTITÀ di DATI che stanno nel tubo in un istante → TUBO PIENO



Network as a pipe

Delay x Bandwidth

- Relative importance of bandwidth and delay depends on application
- For large file transfer, bandwidth is critical
  - posso caricare poco carico alla volta
- For small messages (HTTP, NFS, SMS, etc.), delay is critical
- Variance in delay (*jitter*) can also affect some applications (e.g., audio/video conferencing)
  - variazione nel ritardo

Delay x Bandwidth

- The product Delay x Bandwidth represents how many bits the sender must transmit before the first bit arrives at the receiver if the sender keeps the pipe full
- Takes another one-way delay to receive a response from the receiver
  - The time to go and come back is called **Round Trip Time** (RTT) - usually equal to 2*Delay
- If the sender does not fill the pipe (i.e., send a whole RTT × bandwidth product's worth of data before it stops to wait for a signal) it will not fully utilize the network

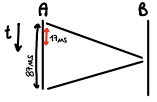
Delay x Bandwidth

- Some example Delay x Bandwidths (here the delay is RTT)

Link Type	Bandwidth (Typical)	Distance (Typical)	Round-trip Delay	Delay × BW
Dial-up	56 Kbps	10 km	87 μs	5 bits
Wireless LAN	54 Mbps	50 m	0.33 μs	18 bits
Satellite	45 Mbps	35,000 km	230 ms	10 Mb
Cross-country fiber	10 Gbps	4,000 km	40 ms	400 Mb

RITARDO ANDATA-RITORNO

massimo numero di bit per riempire il canale



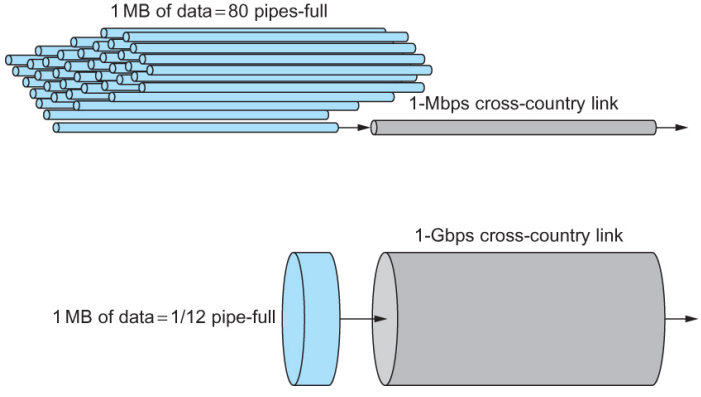
56 Kb/s in tempo? $\Rightarrow \frac{1}{56 \cdot 10^3} = 0{,}017 \cdot 10^{-3} = 1{,}7 \cdot 10^{-5} = 17\,\mu s$ ⟶ questo è il tempo per trasmettere 1 bit → 70 μs sprecati

1 bit, in quanto tempo?

$\frac{56\,Kb}{1s} = \frac{1\,bit}{x}$ $\quad 56 : 1 = 1 : x \quad \frac{1}{56}$

Delay x Bandwidth

- Infinite bandwidth: RTT dominates
- Throughput = TransferSize / TransferTime
- TransferTime = 1/Bandwidth x TransferSize + RTT
- It's all relative
  - 1-MB file to 1-Gbps link looks like a 1-KB packet to 1-Mbps link



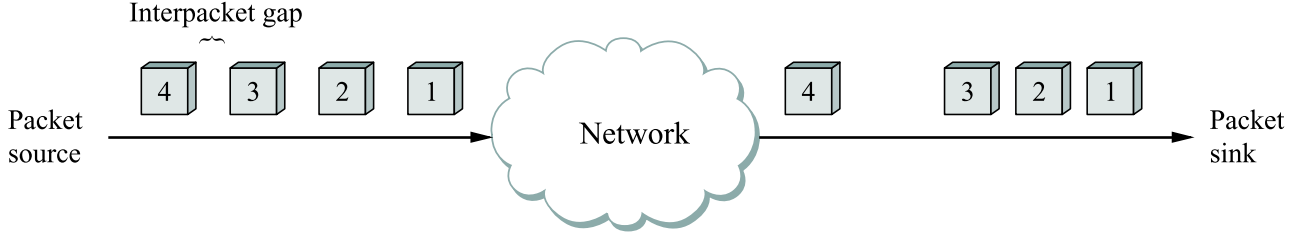
A 1-MB file would fill the 1-Mbps link 80 times, but only fill the 1-Gbps link 1/12 of RTT of 100 ms

Jitter

per i programmi è importante che il ritardo sia lineare e costante

buffering su YT ad esempio per rendere più fluida la visualizzazione

- Sometimes, delay is not a problem, but its variance is
  - E.g. multimedia, streaming audio/video
- Delay variance is called *jitter*
- Such variation is not usually introduced in links (they are passive, physical media) but rather by variable queues in switch in multi-hop network



Chapter 1 - Summary

- We have identified what we expect from a computer network
- We have defined a layered architecture for computer network that will serve as a blueprint for our design
- We have discussed the socket interface which will be used by applications for invoking the services of the network subsystem
- We have discussed two performance metrics using which we can analyze the performance of computer networks