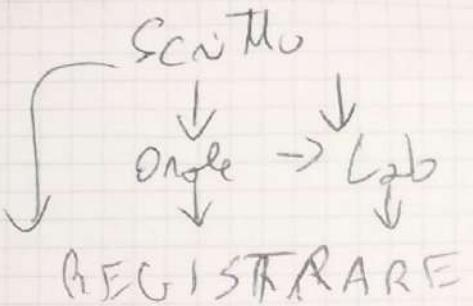


Esempio

Scritto: 15 ... 22

Onde: -5 ... 5

Lab: -5 ... 5



Sistema interconnessione - Closed Computing - IOT

Se si piccola la vita alle persone

Sistema di connessione fatto da:

- Nessun filo o trasmettore il regole — pubblica  
della costituzionalità
- Strutture logiche (software) — finché nel  
mondo

Cavo, onde elettromagnetiche ecc... — Alcuno poi

Meno — Trasmettente — Ricevitore

Il regole non è informazione, le strutture logiche  
si occupa di "trasmettere" un'informazione

Si deve individuare l'informazione al giusto modo.

Le strutture logiche si occupa di tutto

Problema: Il regole quando riguarda il suo pubblico,  
ma c'è lo stesso software che serve a  
risolvere il problema non ha questa interpretazione  
del regole (non regole)

Il regole pensamento troppo spazio e dogmatico, a  
volte il meccanismo non regolare del regole

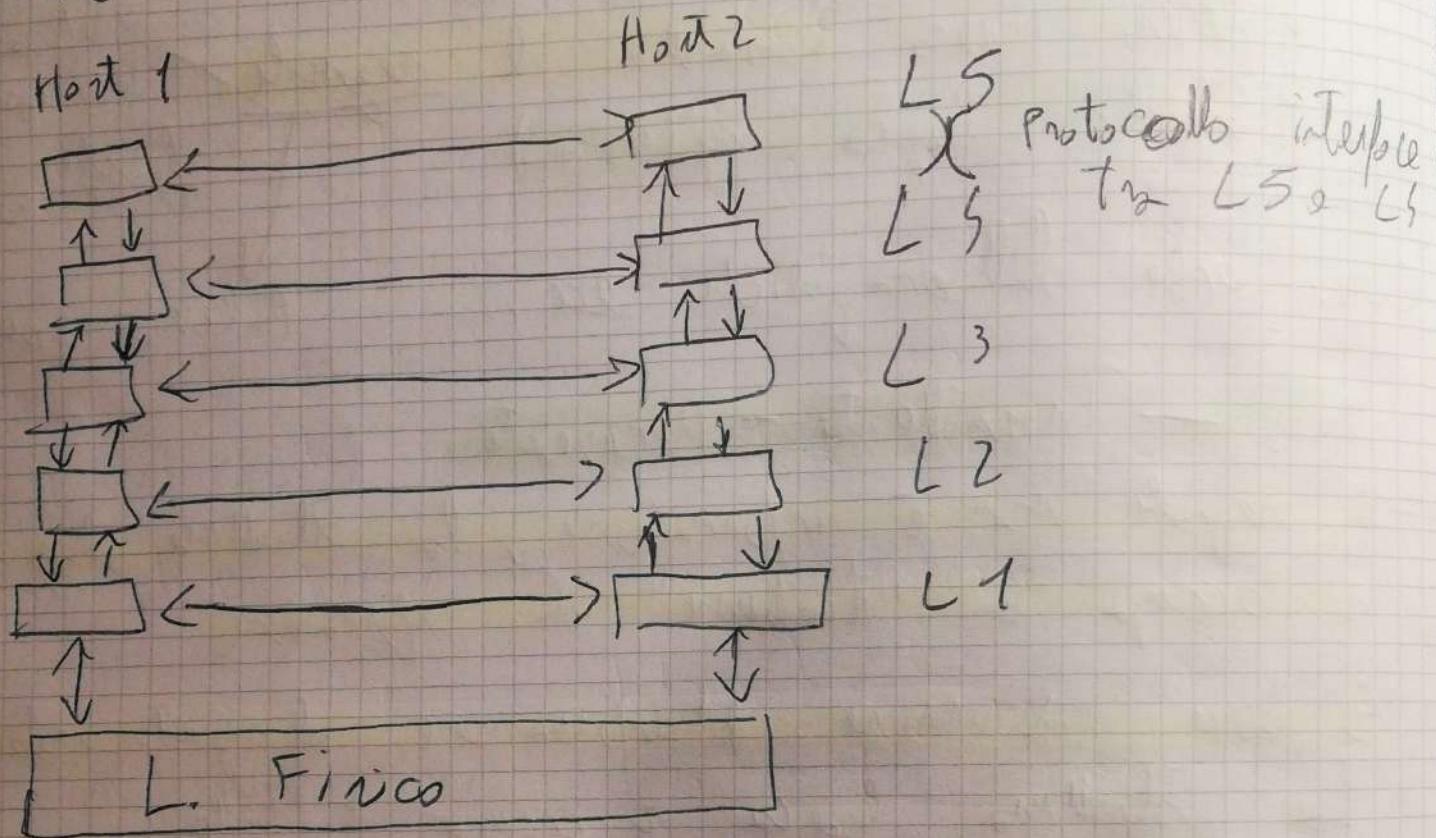
## Protocolli

definisce come e' attivato un messaggio

Fra di Host finchè ogni layer c'è un protocollo  
Fra un layer S e S+1 c'è l'interface

Host 1: Browser richiede informazione a

Host 2: WWW. E' ricevuta la richiesta con un  
protocollo di un certo tipo, ma poi subisce  
la modifica (Host 2 invia un segnale di risposta)



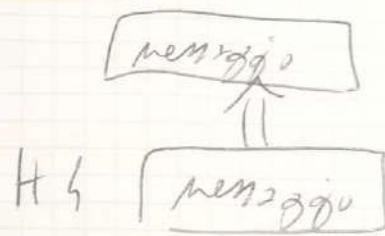
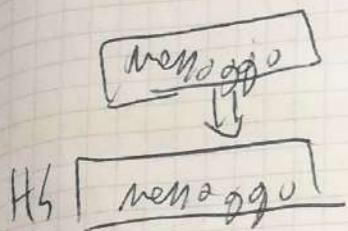
Fra vari L c'è uno dei protocolli.

Fra Host c'è un protocollo per comunicare

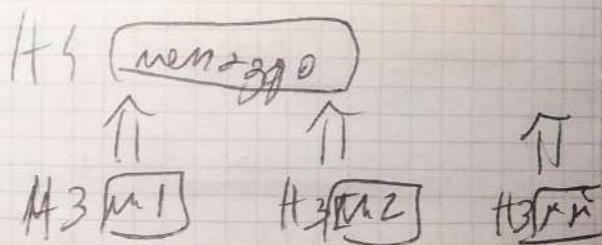
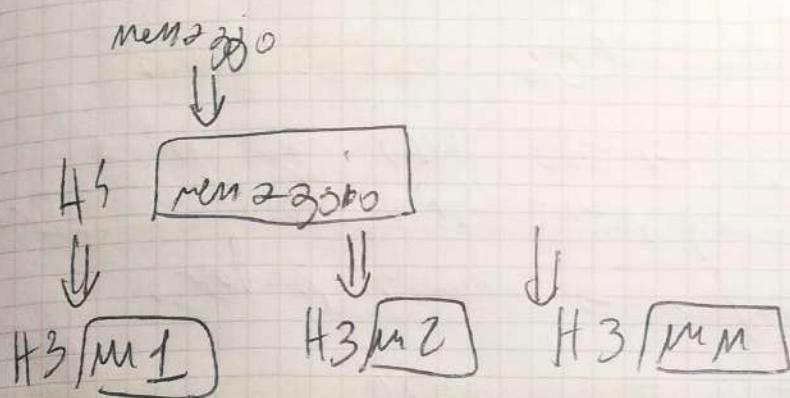
3 protocolli: verso il basso, alto, box / fax

Protocollo def: definisce come e' attivato un messaggio  
rispetti alle entita' in comunicazione e le  
anche le attivita' in fase di trasmissione e/o ricezione  
messaggio o altro evento

## Schemi di comunicazione



E' più facile mantenere il messaggio verso il basso (alto)



Interdizione: viene letto come interdizione il messaggio  
 Informazione di controllo: fa capire cosa fare al messaggio

La dimensione totale dei

spaziamenti è più grande del messaggio, dato che i spaziamenti oltre al messaggio altre informazioni (di controllo)

## Protocol Headers:

Viengono trasmesse varie informazioni:

- source address
- dest address

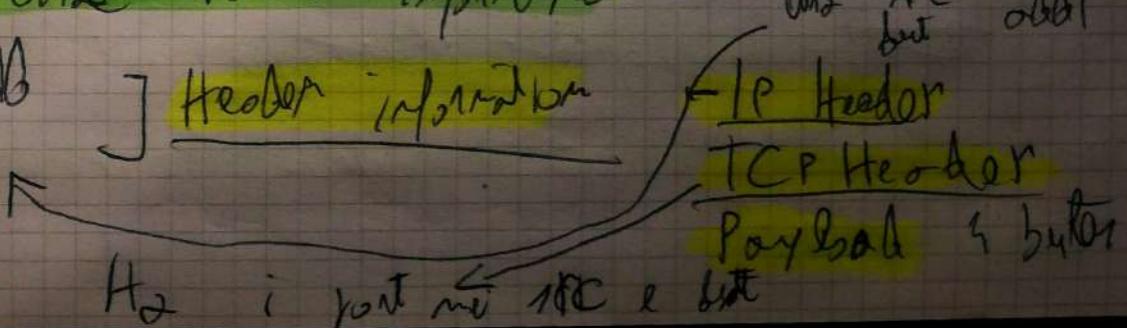
Header information

MSB 1FC but offset

IP Header

TCP Header

Payload & bytes



A volte c'è un rapporto 50:1 tra info  
di controllo e messaggio

Ei distinguono i messaggi e i invi (messaggio)  
al livello corrispondente dell'alto livello (i invi come  
la strada di controllo Siger) (invio)

Problemi: E' necessario un canale di comunicazione  
affidabile e veloce.

Il canale è implementato così: esempio

Canale binico:

Simplex	→ solo invio
Half-duplex	→ Entrambe le direzioni a turno
Full-duplex	→ Entrambe le direzioni

• Messaggi di livello basso: non possono avere lunghezza  
(ci sono dei regoli)

• Trasmettore veloce non può trasmettere a velocità  
lenta.

• E' dove trovare il percorso migliore - rotta - destinazione  
(la ricerca è arbitraria) (ci sono molte variabili)

• Ordine arrivo messaggi deve essere uguale a  
quello di ricezione

N.B.: I regoli di controllo servono per dirigere bene  
le info i regoli-messaggio

Per evitare tutti questi problemi servono dei protocolli  
ben precisi

## Comunicazione

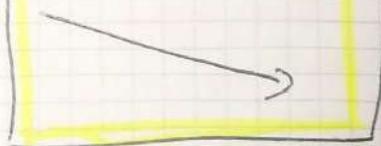
Connection bus = Affidabile

Connection socket = non affidabile

Teléfono è connection oriented

## Connection less

Morbore utile

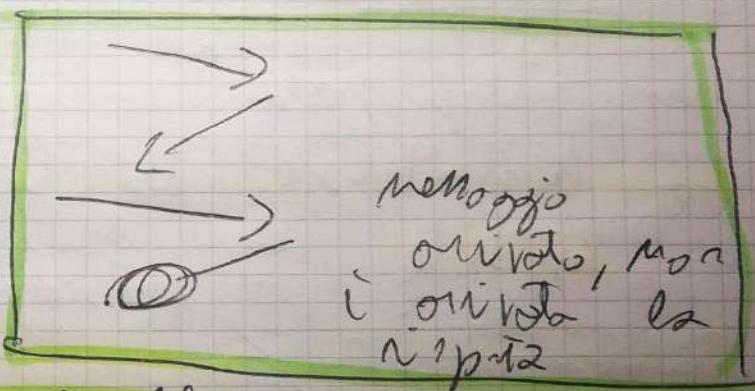
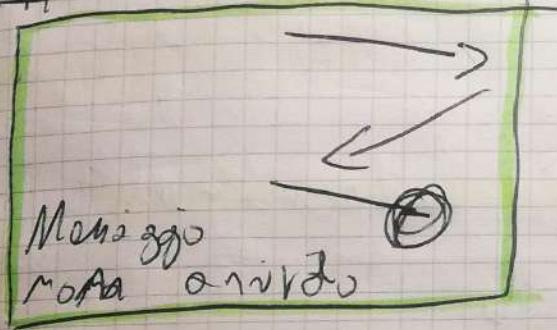


Azione — Comunicazione — Chiamata

Entro protocollo ben preciso

Non c'è un protocollo ben preciso

Affidabilità: Si perde il regole, non è affidabile



Dunque manda il pacchetto, se ne perde uno tracce

Soluzione: Arer: in caso di perdita ripete n. invia il pacchetto.

Quindi non vale come soluzione.

Serve un protocollo per ogni possibile situazione problematica relativa ai messaggi che non arrivano o non ricevono risposte  
I timer funzionano in base al protocollo

Problema: determinare a quale impostare il timer

Problema: il protocollo deve essere unico, non  
può esserci un protocollo per il servizio A  
Bob è un servizio per Elisa

Problema: bisogna distinguere bene i pacchetti <sup>informazione</sup> e i pacchetti incaricati.

Il messaggio e l'ACK devono essere numerati.

Nell'ACK c'è solo informazioni di controllo  
(numero come risposta per bire "OK messaggio incaricato")

"Quanto tempo passa?"

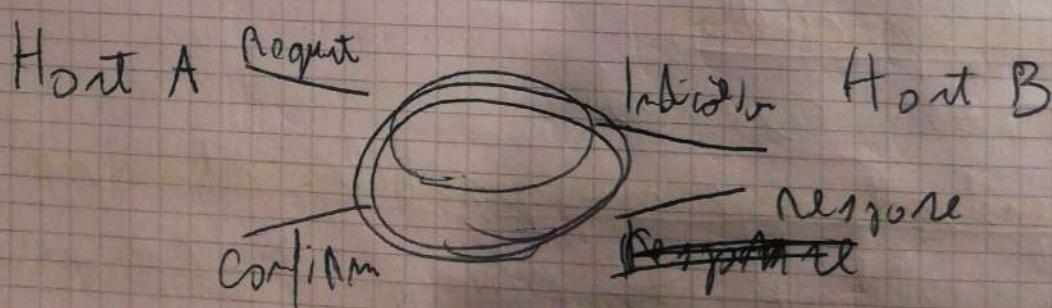
La distribuzione del tempo è estremamente variabile

Protocollo Snello

Connection-less  $\leftarrow$  c'è iraffidibilità

Nei connection-oriented c'è affidabilità

↑  
Protocollo confuso



Modello OSI, divide in 7 layer

Application

Presentation

Sessions

Transport

Network

Data Link

Physical

Ogni livello ha un protocollo per comunicare col livello connesso  
a un altro Host, e protocollo  
per comunicare su livello col  
successivo

**TCP/IP** ha:

- Application layer, Host-to-host
- Transport layer

Non c'è a livello Internet layer, Network Interface layer

Set di protocolli più usato

Modello OSI: è un modello teorico

TCP/IP: modello reale (molti diversi in TCP e IP)

Application: serve applicazioni & utente e definisce protocolli

Presentation: garantisce che applicazioni che vogliono comunicare si interpretino i dati nello stesso modo

Sessions: fornisce definizione e sincronizzazione  
delle relazioni fra dati

**Transport**: trasferire i messaggi dal livello di applicazione tra gradi periferici gestiti dalle applicazioni

Ex: TCP fornisce servizio orientato alla connessione (consente connessione garantita e controllata di flusso) TCP fornisce i messaggi lunghi in più pacchetti

**Rete (network)**: si occupa di trasferire pacchetti a livello di rete, detti datagrammi, da un host a un altro

Usa il protocollo IP

**Data LINK** (<sup>(da ms p<sup>a</sup> elaboratore dati)</sup>): ci sono offerto il servizio dato da questo layer per trasferire un pacchetto da un n<sup>o</sup> host al successivo nel percorso

Dunque in ogni n<sup>o</sup> host il livello di rete può utilizzare gli strumenti del Data LINK per trasportare il n<sup>o</sup> successivo

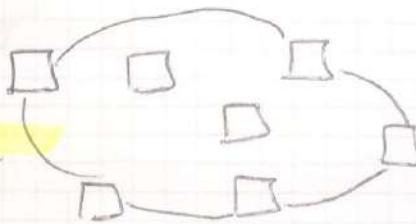
Alcuni protocolli garantiscono efficienza, altri no  
Quelli che sono ottimizzati per attraversare più nodi, lungo il percorso sono quelli che sono più efficienti

**Fisico**: trasferisce i bit dal jocattolo (a livello Data LINK) da un host al successivo

I protocolli qui riguardano gli strumenti fisici esistenti (cavi, onde...) Dunque i bit possono essere trasferiti secondo diverse modalità

## Tipologie di rete

- Collegamento semplice : PUNTO - PUNTO □ - □  
due macchine collegate in modo diretto
- Broad Cast <sup>Punto di raggiungere tutti altri contemporaneamente</sup>  
Per gestire rete e regole



Sempre un protocollo per decidere chi parla e quando.  
Bisogna evitare le interferenze, collisione del segnale

Condivisione / Sincronizzazione dei segnali, i segnali stanno in incidenza non si interaggono e ricorda, non sovrappongono, si sommano.

La tipologia di somma è retinolare:

Condizione: chi ascolta non capisce già nulla

E se un segnale è molto più grande dell'altro, questo segnale si sovrappone all'altro, la somma è trascurabile

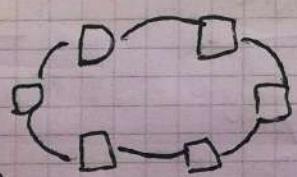
## Tipi di broad cast

Bus condiviso



Tutti i interi tra di loro ignorano il bus

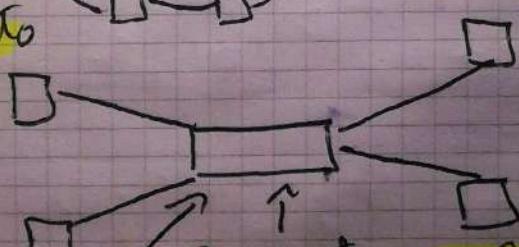
Anello



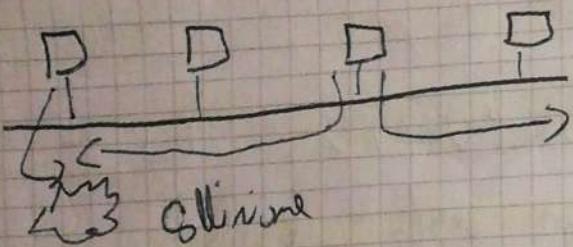
Il segnale non si perde all'infinito

è un modo per fare broad cast con cori Punto - Punto. Non deve entrare collisioni

Stella



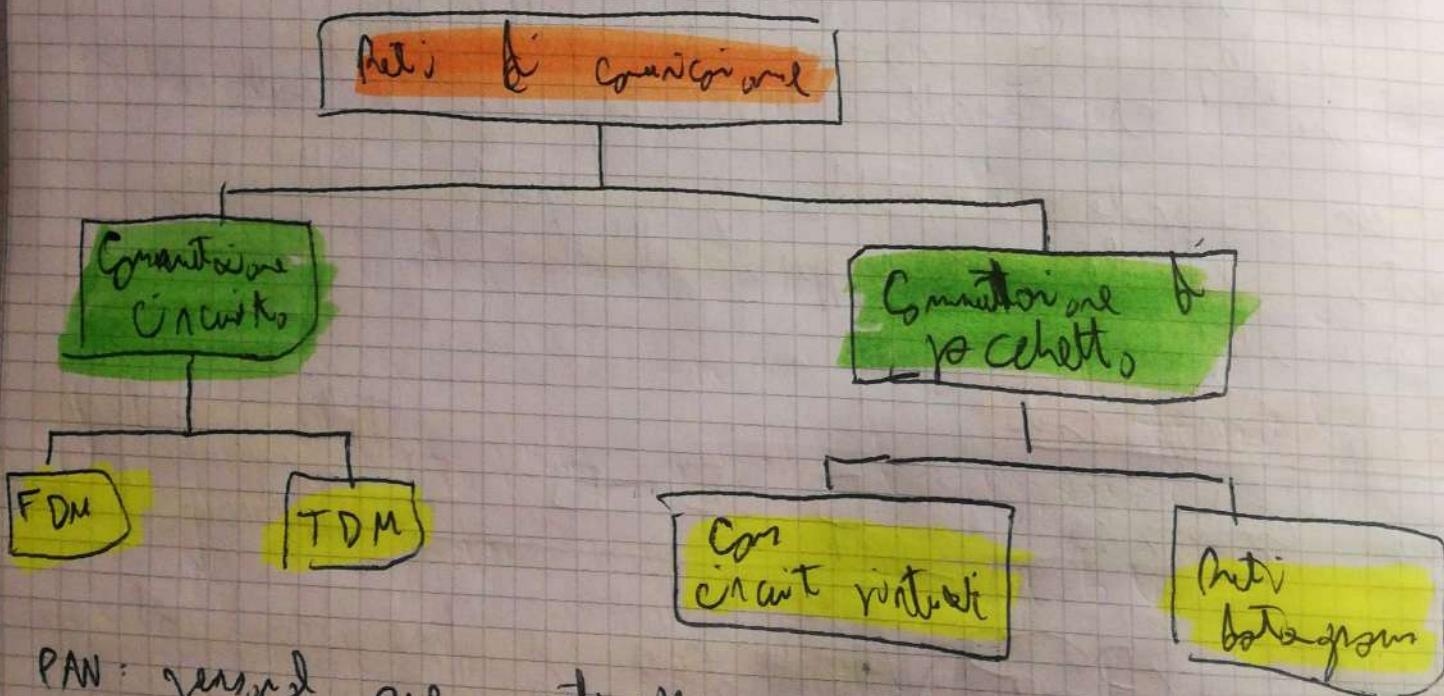
Concentrazione Collegato Punto - Punto con tutte le macchine disponibili; ascolta tutti, non ha il segnale a tutti



~~Ritardo dovuto a processi  
basingo a resezione regole~~

Il broad Cast abusivo permette di ignorare il regole, in questo modo si evita la sovrabbondanza eccessiva del regole. ~~ritardo dovuto a ritardo dovuto a processi~~ e rigenerazione regole

Nel broad cast ~~abusivo~~ e stellare il regole non è operato, qui siamo basati solo a messa in Hub per creare un colpo di regole da inviare i request a tutti;



PAN: personal area network locali

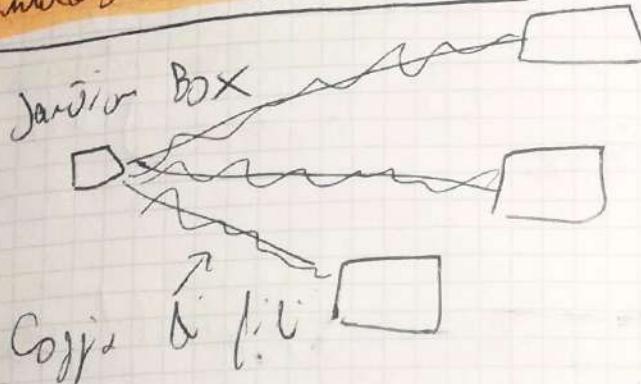
LAN 1 Km

MAN 10 Km

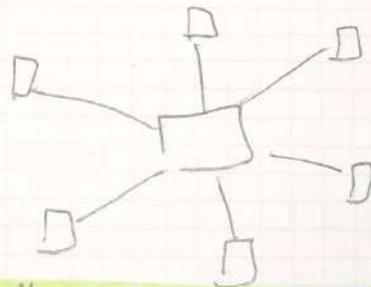
WAN 100-1000 Km

Internet 10K + Km

## Connessione a circuito



F1: Sintesi filosofica



## Fili collegati fiancante

## Connessione a pacchetto

Faccia un giro

## Traess pacchetto rigore

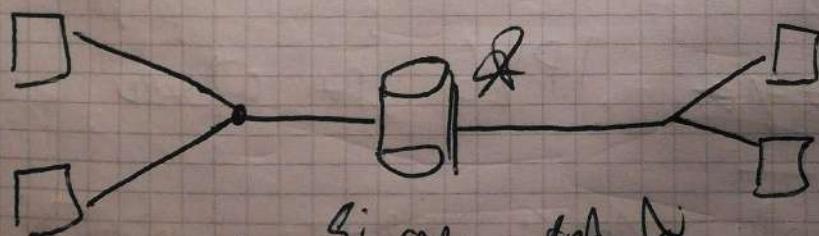
Progetto Airport: →

~~Faccia~~ Ormai non è più già connessione a circuito, ma a pacchetto.

La connessione consiste in uno switch, il solo switch il pacchetto nel pacchetto rigore.

La linea trae la parola mare per inviare gli informi / segnali sulle linee Ogni.

In realtà creando una colonna pacchetti che danno in attesa per entrare inviati al solo successivo



E' ora colonna pacchetti

I generatori di pacchetti hanno il giro colonna

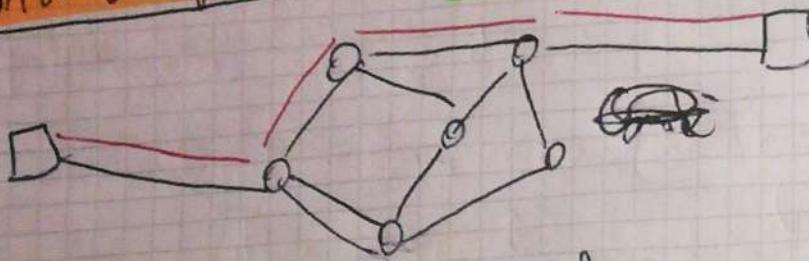
Nel caso di passaggio a sing. intelligenti per gestire traffico pacchetti

E si vede lo spazio del buffer. Per la colonna, non fa open buffering

Crediti ritardo

Jitter minimo

Ogni pacchetto ~~scende~~ negli  
ha meno ritardo  
In tempi Jitter basso



Si realizza nel grado il raggiungimento minimo (finito)

Vantaggi: pacchetti arrivano in ordine, non c'è nessun  
processamento, la decisione di ritardo.  
Il routing è fatto una volta solo  
determinando delle informazioni del 1° pacchetto, tutte  
oggetto

Svantaggi: se un router rotta; si deve riavviare  
ogni pacchetto che deve uscire quel router

Datagram: Per ogni pacchetto si decide una tratta,  
il pacchetto scatta e rotta

Vantaggi: se un router rotta, ci sono  
altro

Jitter: L'utente vuole stampo la propagazione minima, nel  
datagram non è più possibile.

Inoltre  
variazione di ms o di caratteristiche di rete

IPV4 ha solo Datagram, IPV6 ha entrambi

→ N.B. non dice se le altre tecnologie mantengono  
i intervalli di 50 ms, vogliono dire sinistramente  
con intervallo di 50 ms.

Più è grande la diffidenza al guadagno e più è fine più è alto il Jitter.

## Livello Applicativo

Le applicazioni richiedono alla rete un insieme di librerie applicative per realizzare messaggi.

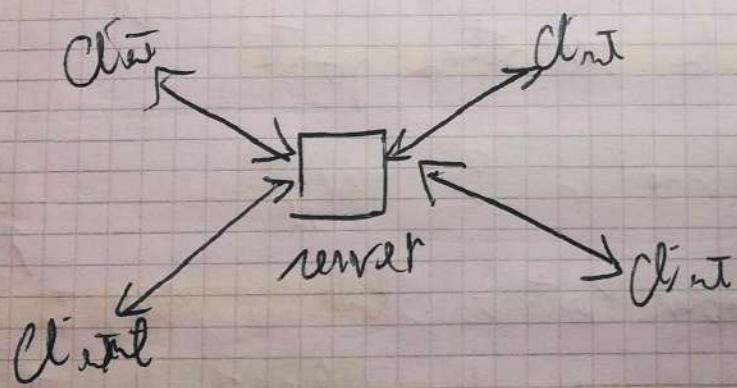
## Comunicazione tra processi

Nel sistema distribuito due processi remoti possono procedere in parallelo se i sincronizzano.

La sincronizzazione è garantita con lo scambio di messaggi.

Il processo invia messaggi alla rete e riceve messaggi dalla rete attraverso un'interfaccia software chiamata socket (che reflette come una "porta")

## Modello Client - server



Client: Node, legge  
Server: scrive, scrive

Server: scrive storia  
Client: guarda storie

Il processo server è attivato ~~per~~ per le domande storie attivate da altri clienti.

Il client-processo è attivo quando scrive, quando vede la barra blu, quando scrive.

## Registi applicazione

### Tolleranza ai punti

- Throughput - lunghezza di banda : Entra in cache
- Fine codice - visual temporal : "Voglio che io veda lo my"
- Sicurezza

Ognuno ha delle caratteristiche diverse dal servizio stesso, nel trasformare di un file non voglio questi, in un file tollestante i tollerabili. ecc ecc

Sono previsti due servizi UDP e TCP, due protocolli di trasporto.

Processi - interi ? tranne le porte \* Somewhere in mobi stabab  
le connessione / collegare

Il processo è legato a un socket dove c'è una porta. Questo socket è collegato per il corrispondente socket della stessa porta

Porte di rete dirette in 3 gruppi

Winter, P28

0-1023 Well Known Port: tutti i servizi già impostati

1024-49151 che mi specifico Registered ports

Dynamic and / or Private ports 49152-65535 ]

\* A oggi processi sono legati una porta per portare vicine in internet. Ogni porta è memorizzata

Ogni porta è memorizzata e impostata con UDP o TCP

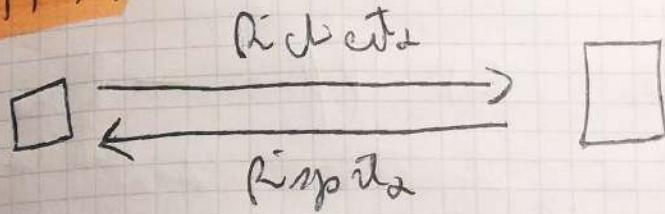
Sono già state

(TCP) Telnet : il servizio su internet, genera accesso a una shell remota, basta e password  
Funzione : utilizza socket, invia output sul socket

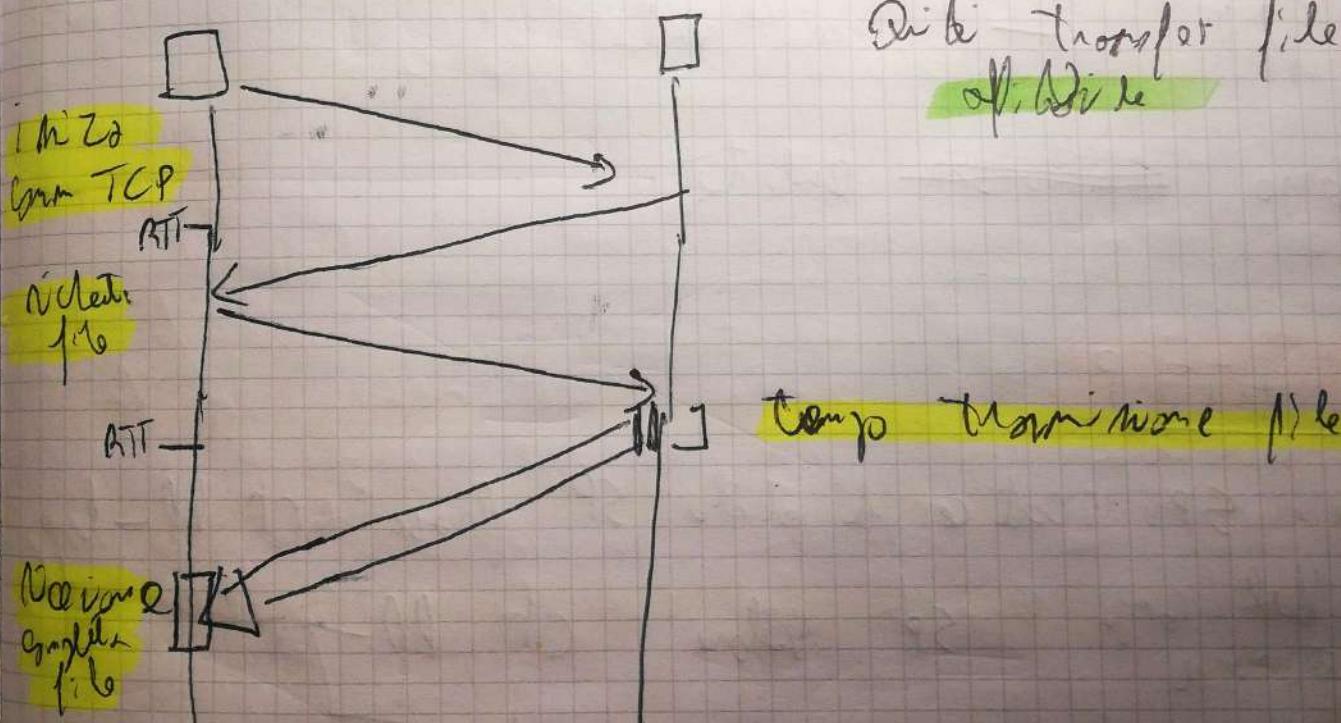
Client, server tutto → invia → Arrivo a server

Telnet ≠ Secure, la password inviata in chiaro può essere spiazzata e la password è in chiaro o trattata.  
Non serve a proteggere i dati. (client-server)

HTTP



Ecco TCP è un canale comune nelle transazioni



HTTP 1.0 chiede connessione TCP per trasferire singoli oggetti

HTTP 1.1 Max TCP permettimento

HTTP 2 usi compression intestazioni Primo Push Adi  
offro re il dato deve fare due domande

Messaggio

Domanda 1 → segnale risposta → Domanda 2 →

messaggio Domanda 1 → Domanda 2 ...  
completa download parallel

### Risposte HTTP

metodo SP URL SP versione

content controls  
CR LF

- linea  
newline

none del

Corpo interazione

SP valore

CR LF

- linea  
interazione

; Oggi

CR LF - linea vuota

[Corpo ~~intestazione~~ entità]

### Risposte HTTP

versione SP codice di stato SP frase CR LF - linea  
di testo

none del Corpo  
interazione

SP valore CR LF ]

di testo

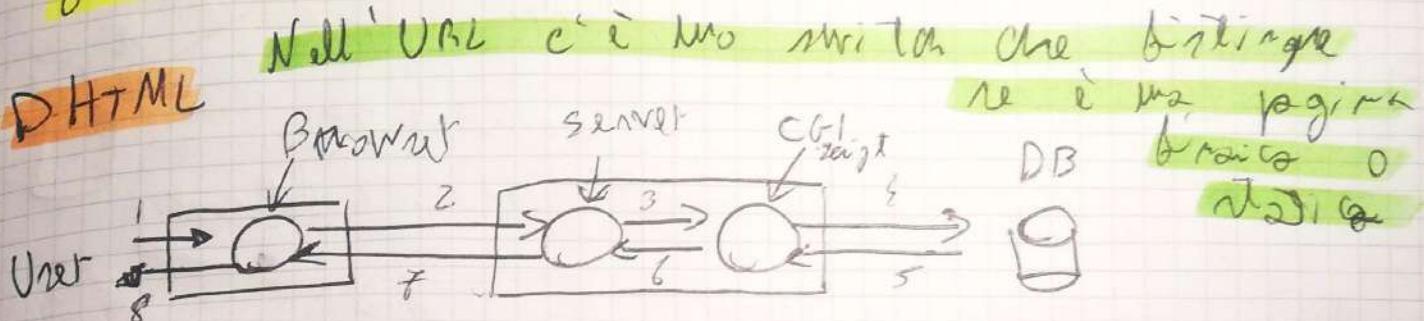
; Oggi

— linea di interazione

CR LF — linea vuota

[Corpo entità]

Nella risposta HTTP compare un codice di stato,  
che non riguarda che i subordinati sono OK,  
o ci sono errori o altro.



- 1) User fills in form
- 2) Form sent back
- 3) Handled to CGI
- 4) query
- 5) record found
- 6) browser pag
- 7) page returned
- 8) page displayed

E' un sistema dinamico che permette migliore interazione client-server, l'output è l'output di un script nello language CGI script - (l'output è comminato in HTML)

### Cookie

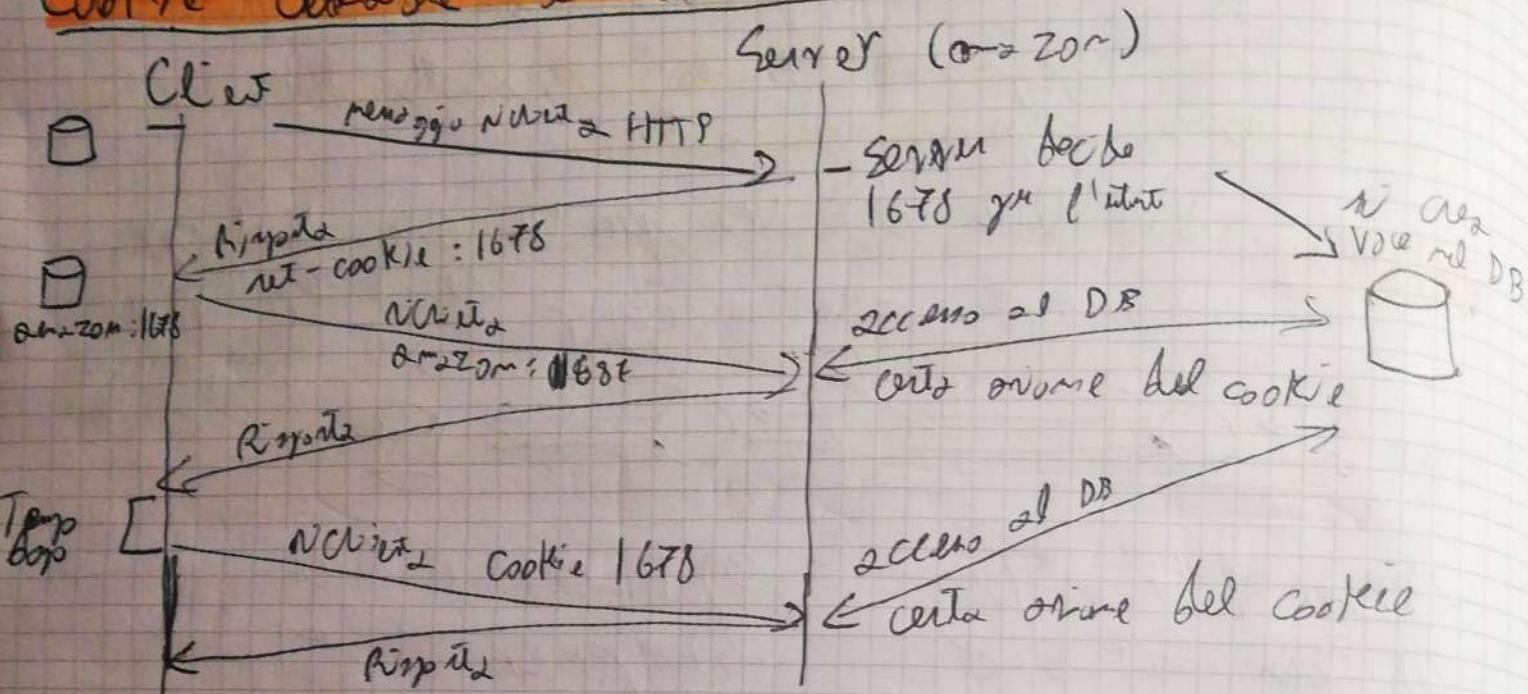
E' una stringa alfanumerica per l'authenticazione utente.

HTTP offre i cookie che permettono ai server di tener traccia degli utenti.

I cookie non sono mai mai sicuri

NB: la versione TCP non si chiude in HTTP  
(quello di oggi)

## Cookie: creazione e utilizzo



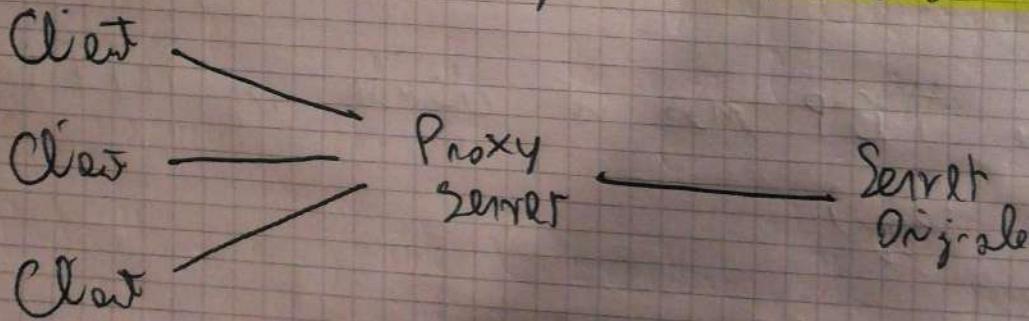
## Proxy Server

Si crea un'interazione tra Client e Server basta di mettere, sotto Web cache o proxy server.

- Entità interne che soffrono (ne guadagnano) richiesta HTTP al posto del web server effettivo.

Come funziona?

- 1) Browser invia richiesta HTTP al server proxy.
- 2) Se l'oggetto richiesto è in memoria si sottopone alla richiesta HTTP e basta, altrimenti lui fa una richiesta HTTP al server originale.
- 3) Sottopone la richiesta in memoria l'oggetto immagine e sottopone la richiesta all'client.

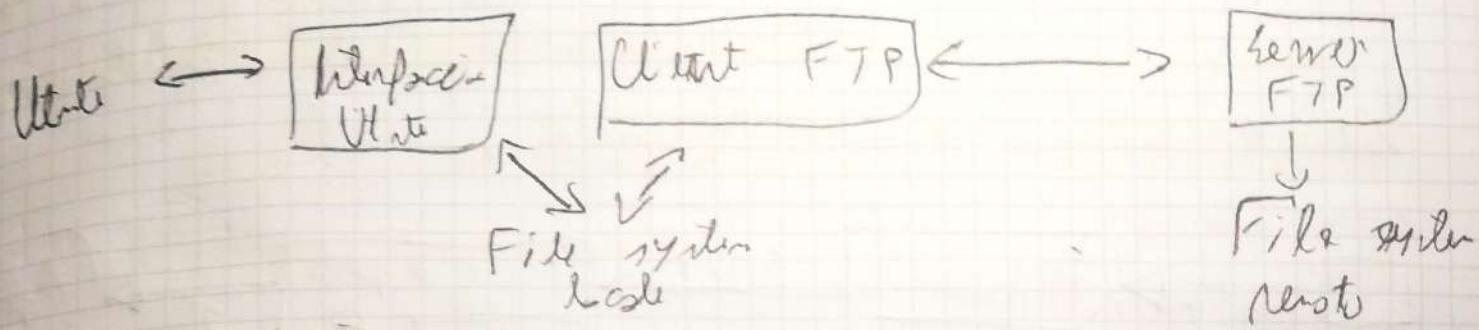


FFF P 5 m outcrop

SFTP, FTPS = secure version of FTP

FT9

Oltre due sporti segnalati; una gara; Gallo e una gara; messaggi. Verba autorizzazione scritta (in dialetto) bontà



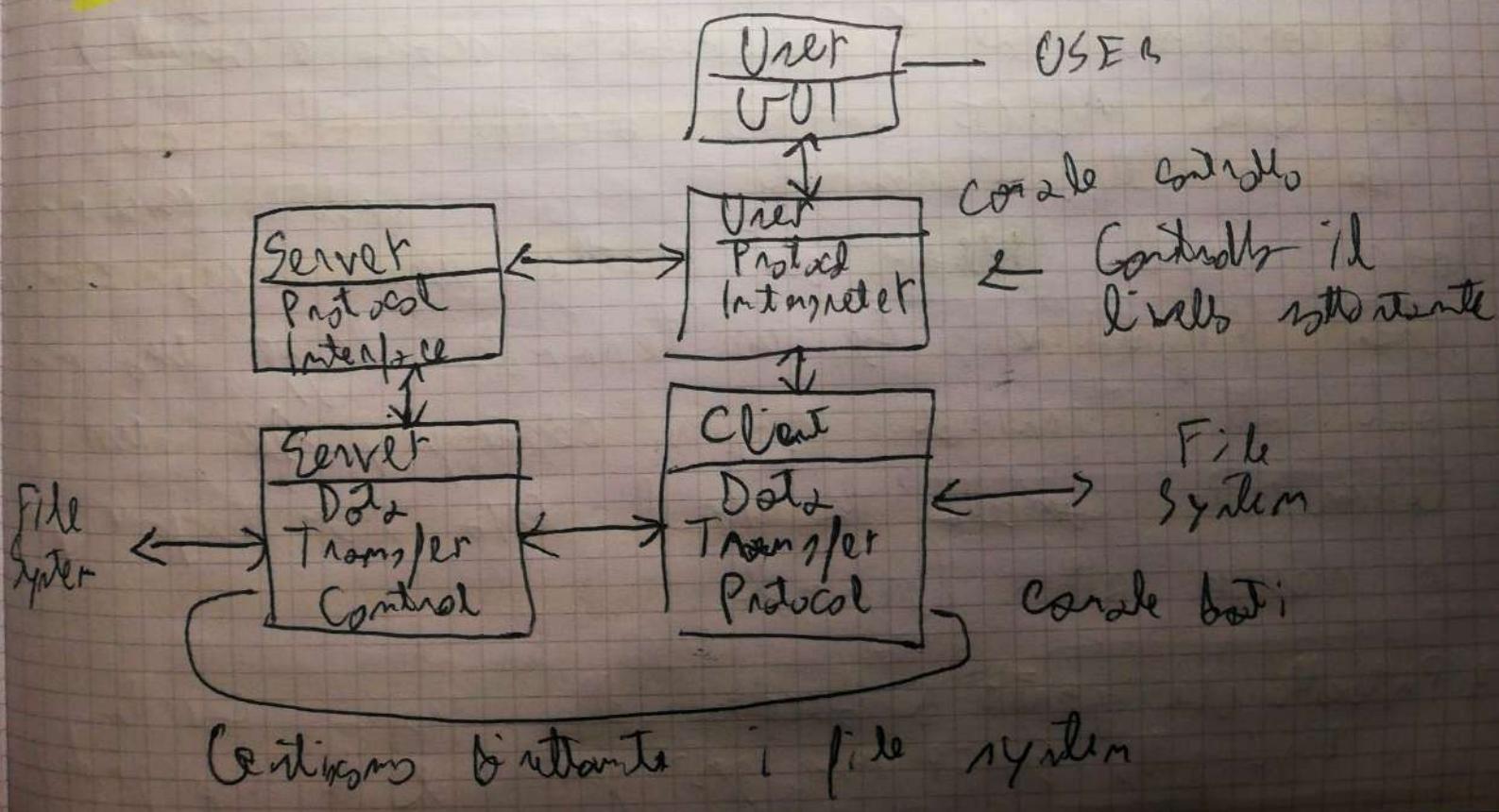
To review the lower grade I can do by HTTP

con la rete TCP, qui ti dà una o più cookie

E' un protocollo Pull e Push [ Passo graduale ifile nel server nato e circolare ]

Non è questo questo protocollo per motivo di sicurezza

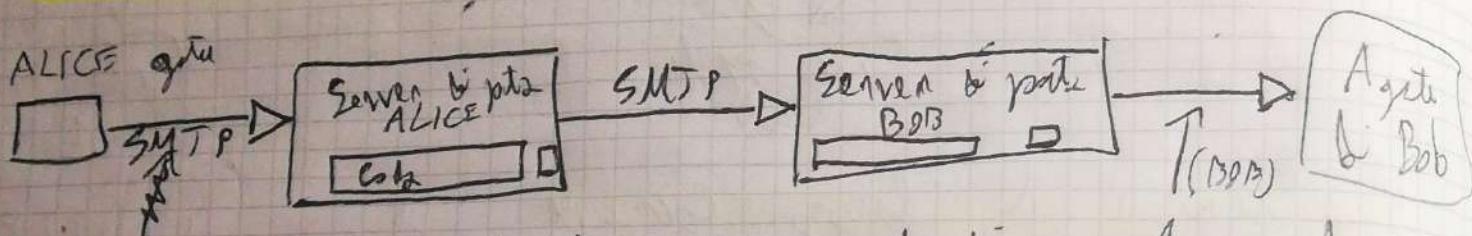
Entomo voranti di FTP che generano & accende  
una sostituzione



**SNTP**, è protocollo **Push**  
**protocollo per le email**

Nel senso che i posti sono  
generati da colo di rete  
e le corrispondenti.  
Server di posti possono fare il  
client del rete.

Utente ha account nello macchinari, riceve messaggi  
che viene messo ~~dai~~ tra spazio nello contenitore  
di un altro server di posti.



Qui siamo su un protocollo che permette di accedere al  
server di posti di ~~ALICE~~ BOB

**SNTP** gestisce comunicazioni tra server di posti

Un protocollo tipo la segnale è **Server push**

Come ad esempio **POP**.

E' occupata di far ricevere la mail nell'agente e  
di cancellare dal server (mobilità servizio e cancella)

Ma come ALICE non può accedere alla mail  
da un altro terminal.

(mobilità servizio e cancella)

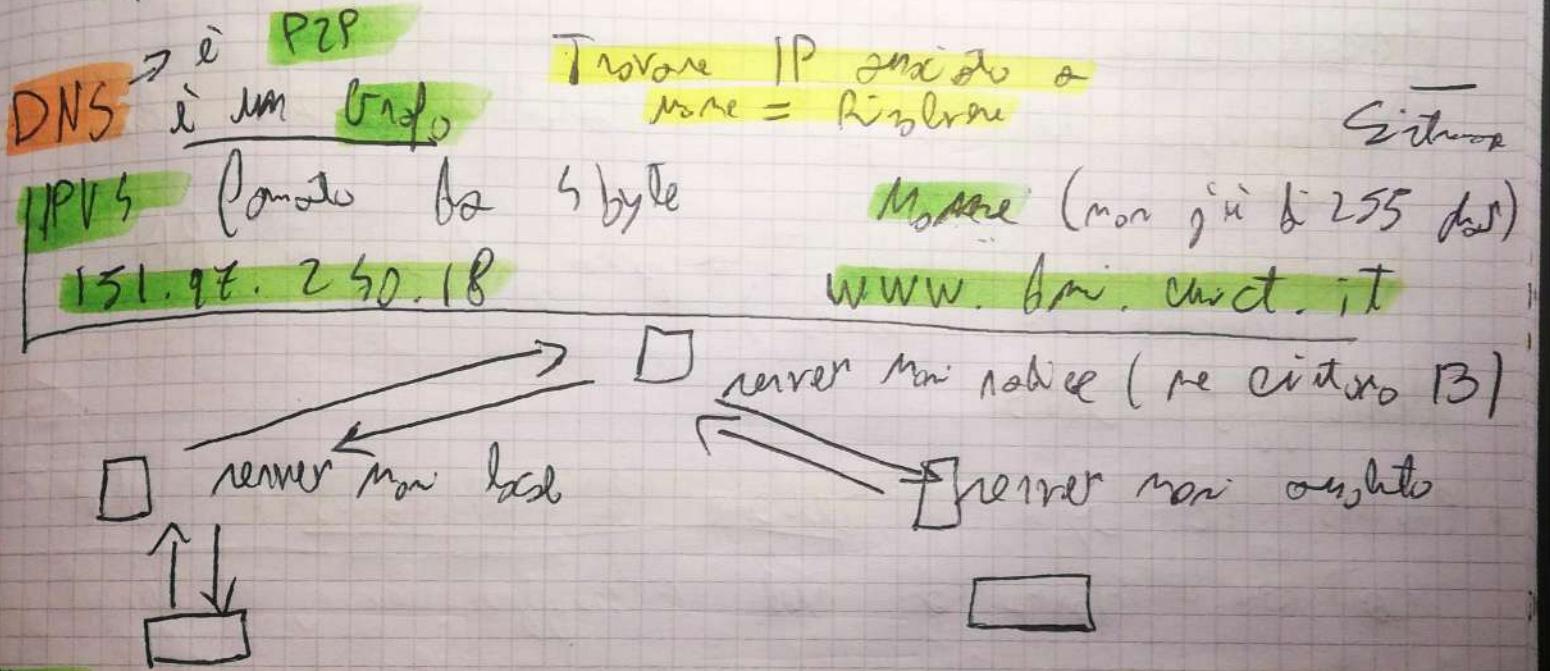
Nel caso quindi **IMAP** che permette una sincronizzazione  
tra due punti. Se un terminale legge la mail  
dal altro terminal comando comunque, se un  
terminale cancella la mail allo stesso tempo già  
in memoria sull'altro terminale.

\* Protocollo che permette di accedere alla mail quella  
(è tipo pull, SNTP non può gestire il tip push)

I server di posta conservano le loro col SMTP  
invio dei messaggi contenenti codice per segnalare l'oggetto  
e i segnali nel protocollo un solo per finire il messaggio.  
Ese: end with <CRLF>. <CRLF>

Mime: standard che permette di estendere la definizione  
di formato dei messaggi di posta elettronica

SMTP supporta anche strumenti per ~~risolvere~~ rilevare le  
spame e bloccarla.



Si accede al Server Modulo intermedio che serve moduli  
locali e server di moduli andati per collegarsi  
in moduli già finiti.

DNS non è client-server ma P2P cioè ogni  
macchina è sia server che client

L'idea del DNS è "ne non si niente un collegamento Web e chi sta sopra"

I dati sono composti come un albero per collegarsi ma in realtà è un grafo.  
Il tipo DNS è molto gerarchico, si suddivide in DNS primi e secondari.

(I record ~~sono~~ servono a individuare il traffico)

Si utilizza sempre un iteratore di coding

### Record DNS:

Definiscono le diverse esistenze dei dati

Ogni risposta DNS trasporta logici RR  
(record di risposta)

### Messaggio DNS

- Primi 12 byte: Sezione di Introduzione che contiene informazioni sul messaggio
- Sezione delle domande: ha informazioni sulle richieste effettuate
- Sezione delle risposte: ha gli RR in risposta alle query
- Sezione autorizzativa: ha i record di altri server autorizzativi
- Sezione aggiuntiva: ha altri record di aiuto

Il DNS può servire per monitorare il traffico

DNS non è molto sicuro, è soggetto ad attack  
e quando i dati in chiave

## SNMP: Simple Network Management Protocol

è un protocollo per la monitoraggio di un rete  
o altro.

Si basa sull'agent e manager che controllano  
ogni altra unità nel DB MIB.

Il manager può interrogare (get) ma anche  
modificare (set)

Gli agent interagiscono col SO e possono inviare  
anche trap

Questi sono per gestione / monitoraggio in remoto

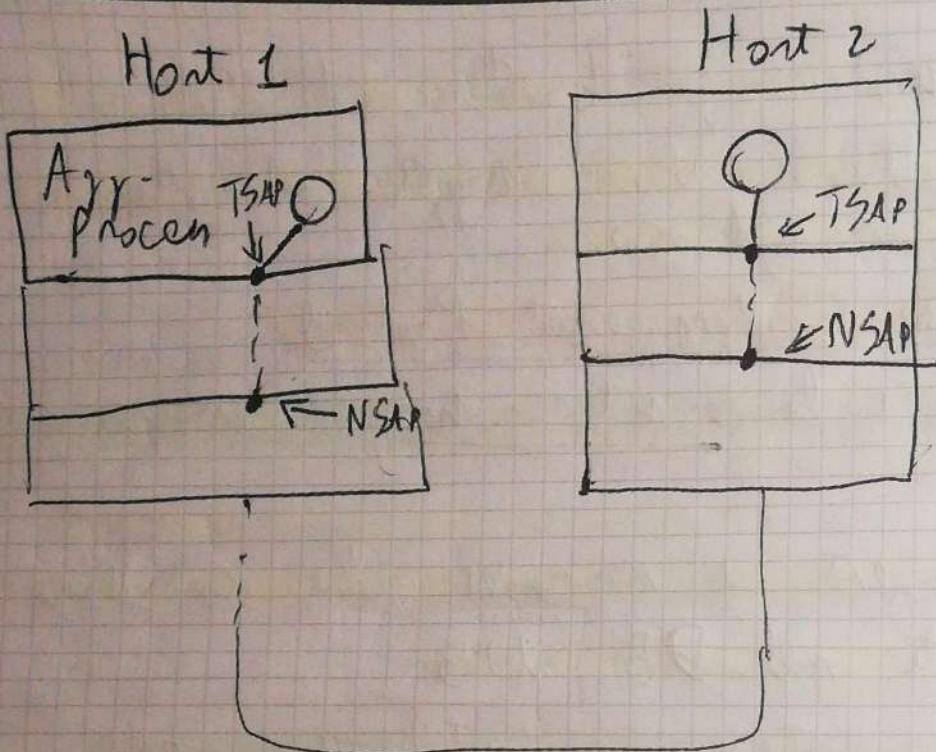
## Livello di Transporto

Unità / Segnale parte dalla LSI a parte  
bus e LSI

Il livello di trasporto facilita la comunicazione logica  
tra gli host, soltanto le problematiche della  
comunicazione di rete che coinvolge i host si risolvono  
nel mezzo nel segmento

TSAP: collega App a Transport, qui inizia la  
trasport connection

NSAP: collega Trasport a Network, qui inizia la  
network connection



Sostanzialmente il TSAP sostituisce la Porta, collega App. a Transport

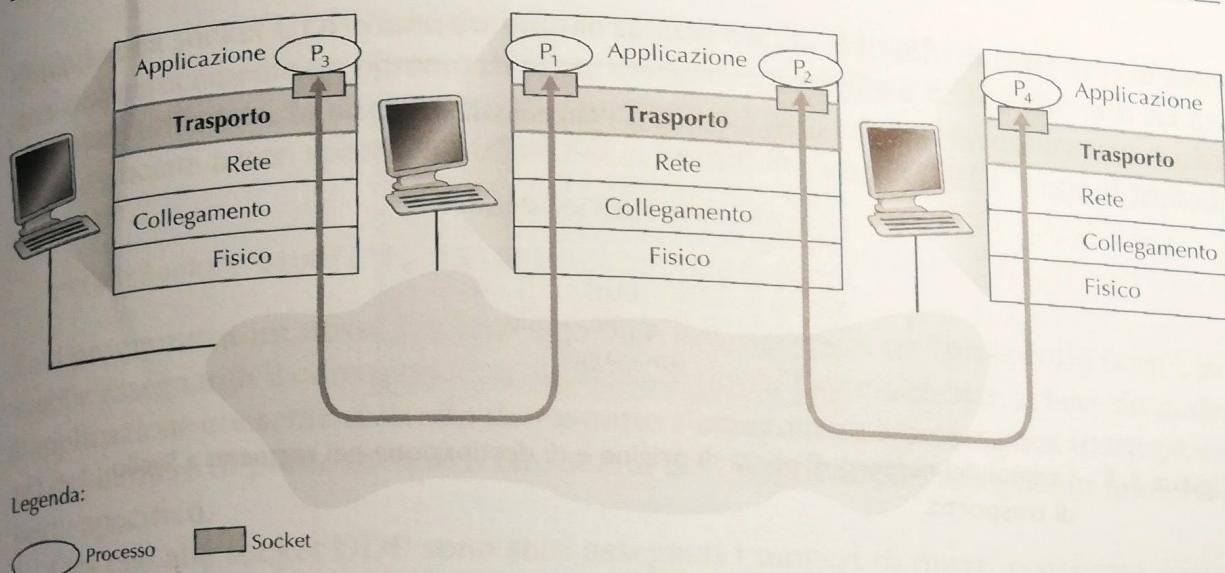
Dunque c'è una connessione, nel registro rete  
il livello di trasporto è una via pista di  
comunicazione che fa scambiare

Il Socket che ageva la connessione è individuato  
da 5 parametri (Connector-socket)

- Porta src .IP src
- Porta dest .IP dest
- Type socket

*Multiplexing e Demultiplexing  
e pag 180-185*

Con questi 5 parametri il web server può fare il  
multiplexing per la comunicazione in parallelo  
con più client

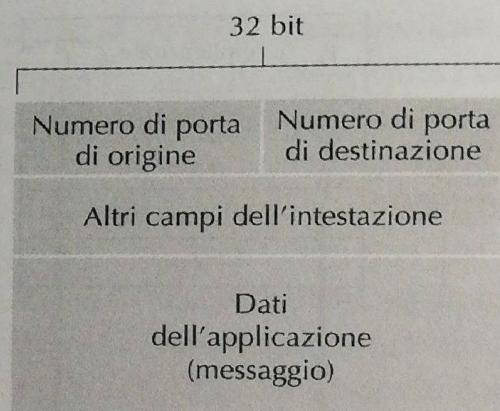


**Figura 3.2** Multiplexing e demultiplexing a livello di trasporto.

dal livello di rete sottostante, li deve indirizzare a uno di questi quattro processi. Esaminiamo come ciò venga realizzato.

Innanzitutto ricordiamo (Paragrafo 2.7) che un processo (come parte di un'applicazione di rete) può gestire una o più **socket**, attraverso le quali i dati fluiscono dalla rete al processo e viceversa. Di conseguenza (Figura 3.2) il livello di trasporto nell'host di ricezione in realtà non trasferisce i dati direttamente a un processo, ma piuttosto a una socket che fa da intermediario. Siccome, a ogni dato istante, può esserci più di una socket nell'host di ricezione, ciascuna avrà un identificatore univoco il cui formato dipende dal fatto che si tratti di socket UDP o TCP, come vedremo tra breve.

Consideriamo ora come l'host in ricezione indirizzi verso la socket appropriata il segmento a livello di trasporto in arrivo. Ciascun segmento a livello di trasporto ha vari campi deputati allo scopo. **Lato ricevente**, il livello di trasporto esamina questi campi per identificare la socket di ricezione e quindi vi dirige il segmento. Il compito di trasportare i dati dei segmenti a livello di trasporto verso la giusta socket viene detto **demultiplexing**. Il compito di radunare frammenti di dati da diverse socket sull'host di origine e incapsularne ognuno con intestazioni a livello di trasporto (che verranno poi utilizzate per il demultiplexing) per creare dei segmenti e passarli al livello di rete, viene detto **multiplexing**. Si noti che il livello di trasporto nell'host centrale della Figura 3.2 deve effettuare il demultiplexing dal livello di rete di segmenti che possono arrivare sia per il processo  $P_1$  sia per  $P_2$ ; ciò avviene indirizzando i dati del segmento in ingresso alla giusta socket. Il livello di trasporto nell'host centrale deve, inoltre, raccogliere i dati in uscita dalle socket dei due processi, creare i segmenti a livello di trasporto e passarli al livello di rete. Sebbene abbiamo introdotto il multiplexing e il demultiplexing nel contesto dei protocolli di trasporto Internet, è importante rendersi conto che queste funzioni hanno uno specifico interesse ogni volta che un protocollo a un qualsiasi livello (di trasporto o qualsiasi altro) è utilizzato da più entità al livello immediatamente superiore.



**Figura 3.3** I campi del numero di porta di origine e di destinazione nei segmenti a livello di trasporto.

Per mostrare il compito del multiplexing e del demultiplexing richiamiamo l'analogia del paragrafo precedente. Anna effettua un'operazione di multiplexing quando raccolge le lettere dai mittenti e le imbuca. Nel momento in cui Andrea riceve le lettere dal postino, effettua un'operazione di demultiplexing leggendo il nome riportato sopra la busta e consegnando ciascuna missiva al rispettivo destinatario.

Ora che abbiamo compreso i ruoli del multiplexing e del demultiplexing a livello di trasporto esaminiamo come vengono realizzati negli host. Da quanto visto in precedenza, sappiamo che il multiplexing a livello di trasporto richiede (1) che le socket abbiano identificatori unici e (2) che ciascun segmento presenti campi che indichino la socket cui va consegnato il segmento. Questi (Figura 3.3) sono il **campo del numero di porta di origine** e il **campo del numero di porta di destinazione**. I segmenti UDP e TCP presentano anche altri campi, come vedremo più avanti. I numeri di porta sono di 16 bit e vanno da 0 a 65535, quelli che vanno da 0 a 1023 sono chiamati **numeri di porta noti** (*well-known port number*) e sono riservati per essere usati da protocolli applicativi ben noti quali HTTP (porta 80) e FTP (porta 21). L'elenco dei numeri di porta noti è fornito nell'RFC 1700: la sua versione aggiornata è consultabile tramite <http://www.iana.org> [RFC 3232]. Quando si sviluppa una nuova applicazione, è necessario assegnarle un numero di porta.

Ora dovrebbe essere chiaro come il livello di trasporto possa implementare il servizio di demultiplexing: ogni socket nell'host deve avere un numero di porta, e quando un segmento arriva all'host il livello di trasporto esamina il numero della porta di destinazione e dirige il segmento verso la socket corrispondente. I dati del segmento passano, quindi, dalla socket al processo assegnato. Come vedremo, questo è fondamentalmente il modo in cui agisce UDP, mentre il multiplexing/demultiplexing TCP è ancora più raffinato.

### Multiplexing e demultiplexing non orientati alla connessione

Ricordiamo, dal Paragrafo 2.7.1, che i programmi Python in esecuzione possono creare una socket UDP con l'istruzione

```
clientSocket = socket(AF_INET, SOCK_DGRAM)
```

Quando una socket UDP viene definita in questo modo, il livello di trasporto le assegna automaticamente un numero di porta compreso tra 1024 e 65535 che non sia ancora stato utilizzato. In alternativa, un programma Python potrebbe creare una socket UDP associata a uno specifico numero di porta (per esempio, 19157) con il metodo `bind()`:

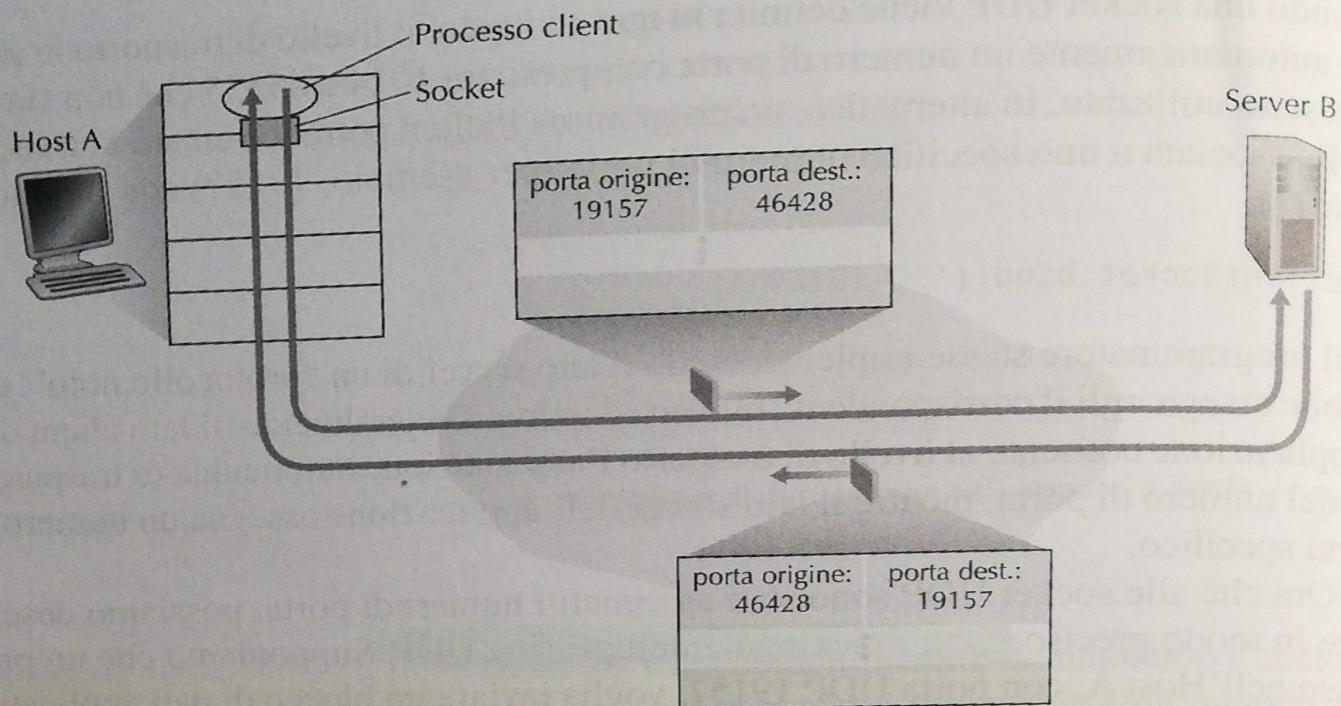
```
clientSocket.bind(('', 19157))
```

Se il programmatore stesse implementando il lato server di un “protocollo noto”, dovrebbe assegnargli il corrispondente numero di porta. Generalmente, il lato client dell’applicazione consente al livello di trasporto l’assegnazione automatica (e trasparente) del numero di porta, mentre il lato server dell’applicazione assegna un numero di porta specifico.

Ora che alle socket UDP sono stati assegnati i numeri di porta, possiamo descrivere in modo preciso il multiplexing/demultiplexing UDP. Supponiamo che un processo nell’Host A, con porta UDP 19157, voglia inviare un blocco di dati applicativi a un processo con porta UDP 46428 nell’Host B. Il livello di trasporto di A crea un segmento che include i dati applicativi, i numeri di porta di origine (19157) e di destinazione (46428) e due altri valori (che non sono importanti per l’attuale discussione e verranno trattati più avanti). Il livello di trasporto passa, quindi, il segmento risultante al livello di rete, che lo incapsula in un datagramma IP, ed effettua un tentativo best-effort di consegna del segmento all’host in ricezione. Se il segmento arriva all’Host B, il suo livello di trasporto esamina il numero di porta di destinazione del segmento (46428) e lo consegna alla propria socket identificata da 46428. Osserviamo che l’Host B potrebbe avere in esecuzione più processi, ciascuno con la propria socket UDP e relativo numero di porta. Quando i segmenti UDP giungono dalla rete, l’Host B dirige ciascun segmento (ossia ne esegue il demultiplexing) alla socket appropriata esaminando il numero di porta di destinazione del segmento.

È importante notare che una socket UDP viene identificata completamente da una coppia che consiste di un indirizzo IP e di un numero di porta di destinazione. Di conseguenza, se due segmenti UDP presentano diversi indirizzi IP e/o diversi numeri di porta di origine, ma hanno lo stesso indirizzo IP e lo stesso numero di porta di destinazione, saranno diretti allo stesso processo di destinazione tramite la medesima socket.

Per quanto riguarda il numero di porta di origine, osserviamo la Figura 3.4 in cui, nel segmento che va da A verso B, il numero di porta di origine serve come parte di un “indirizzo di ritorno”: quando B vuole restituire il segmento ad A, la porta di destinazione del segmento da B verso A assumerà il valore della porta di origine del segmento da A verso B. L’indirizzo di ritorno completo è costituito dall’indirizzo IP di A più il numero di porta di origine. Per esempio, vediamo il programma server UDP studiato nel Paragrafo 2.7. In `UDPServer.py`, il server utilizza il metodo `recvfrom()` per estrarre il numero di porta di origine dal segmento ricevuto dal client; poi invia un nuovo segmento al client, in cui il numero di porta di origine estratto viene usato come numero di porta di destinazione del nuovo segmento.



**Figura 3.4** Inversione dei numeri di porta di origine e di destinazione.

### Multiplexing e demultiplexing orientati alla connessione

Per comprendere il **demultiplexing TCP** dobbiamo analizzare da vicino le socket TCP e il modo in cui si stabiliscono le connessioni TCP. Una sottile differenza tra una socket TCP e una socket UDP risiede nel fatto che la prima è identificata da quattro parametri: indirizzo IP di origine, numero di porta di origine, indirizzo IP di destinazione e numero di porta di destinazione. Pertanto, quando un segmento TCP giunge dalla rete in un host, quest'ultimo utilizza i quattro valori per dirigere (fare demultiplexing) il segmento verso la socket appropriata. In particolare, e al contrario di UDP, due segmenti TCP in arrivo, aventi indirizzi IP di origine o numeri di porta di origine diversi, vengono diretti a due socket differenti, anche a fronte di indirizzo IP e porta di destinazione uguali, con l'eccezione dei segmenti TCP che trasportano la richiesta per stabilire la connessione. Per chiarire meglio questo aspetto riconsideriamo l'esempio del Paragrafo 2.7.2.

- L'applicazione server TCP presenta una “socket di benvenuto” che si pone in attesa di richieste di connessione da parte dei client TCP (Figura 2.29) sulla porta numero 12000.
- Il client TCP crea una socket e genera un segmento per stabilire la connessione tramite le seguenti linee di codice:

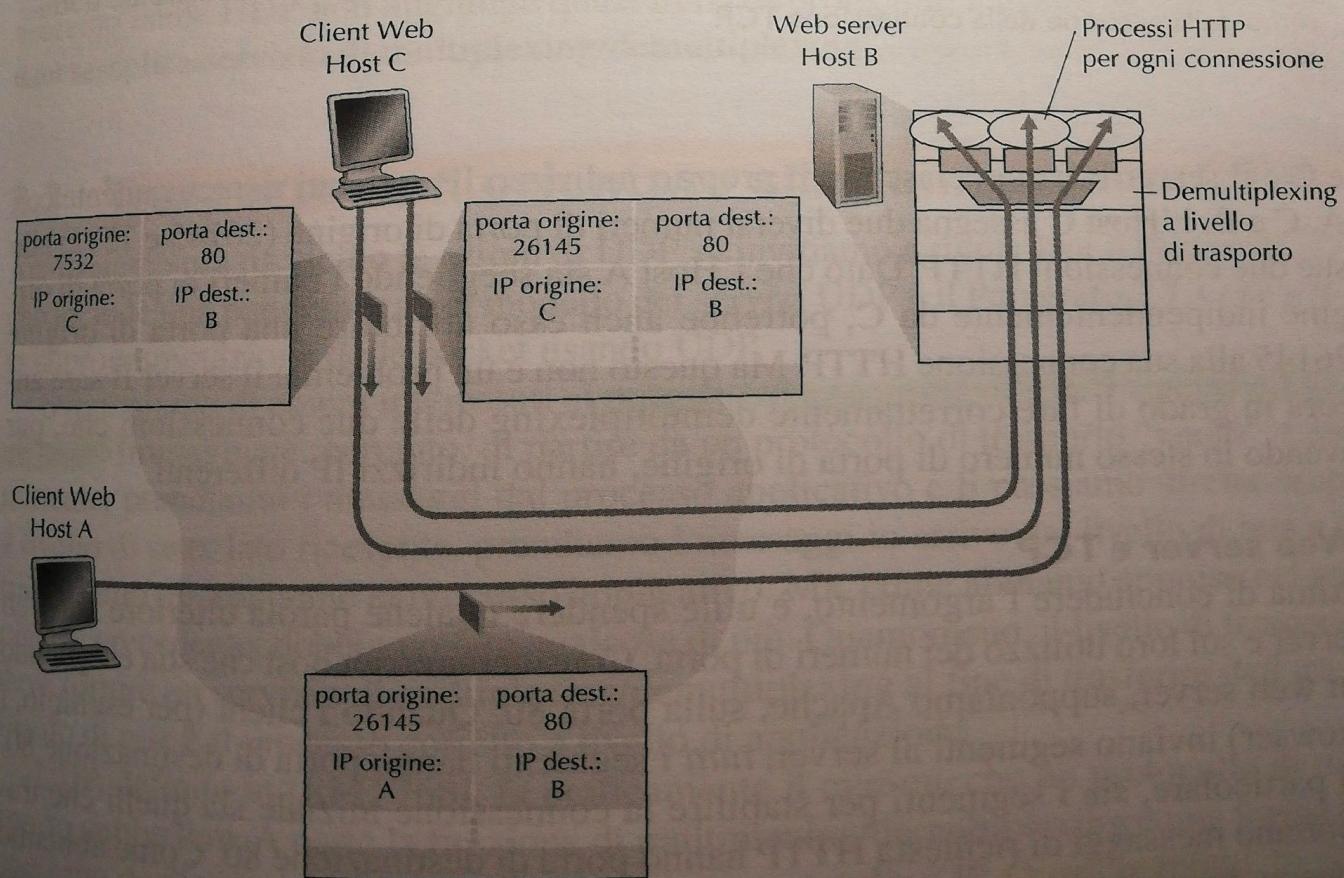
```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, 12000))
```

- Una richiesta di connessione non è nient'altro che un segmento TCP con numero di porta di destinazione 12000 e uno speciale bit di richiesta di connessione posto a 1 nell'intestazione (Paragrafo 3.5). Il segmento include anche un numero di porta di origine, scelto dal client.

- Il sistema operativo dell'host che esegue il processo server, quando riceve il segmento con la richiesta di connessione con porta di destinazione 12000, localizza il processo server in attesa di accettare connessioni sulla porta 12000. Il processo server crea quindi una nuova connessione:  
`connectionSocket, addr = serverSocket.accept()`
- Inoltre il livello di trasporto sul server prende nota dei seguenti valori nel segmento con la richiesta di connessione: (1) numero di porta di origine nel segmento, (2) indirizzo IP dell'host di origine, (3) numero di porta di destinazione nel segmento e (4) il proprio indirizzo IP. La socket di connessione appena creata viene identificata da questi quattro valori. Tutti i segmenti successivi la cui porta di origine, indirizzo IP di origine, porta di destinazione e indirizzo IP di destinazione coincidono con tali valori verranno diretti verso questa socket. Ora che la connessione TCP è attiva, client e server possono scambiarsi dati.

L'host server può ospitare più socket TCP contemporanee collegate a processi diversi, ognuna identificata da una specifica quaterna di valori. Quando il segmento TCP arriva all'host, i quattro campi citati prima vengono utilizzati per dirigere (fare demultiplexing) il segmento verso la socket appropriata.

La situazione è schematizzata nella Figura 3.5 dove l'Host C dà inizio a due sessioni HTTP verso il server B, mentre l'Host A apre una sessione verso B. Gli Host A



**Figura 3.5** Due client che usano lo stesso numero di porta di destinazione (80) per comunicare con la stessa applicazione sul web server.

UDP è un Protocollo di Transporto

- Il protocollo plauso dati è lo proto collo applicativo
- Non gestisce la connessione, non occupa slot di memoria pacchetto. Si può tuttavia fare a livello applicativo
- Non c'è stato
- Non c'è overhead di gestione

Riuso uno stesso numero di insieme Source port e Destination port UDP è un servizio bidirezionale alle applicazioni cioè i segnali sono passati immediatamente al livello di rete e UDP non ha protocollo End-to-end

Dunque UDP non è orientato alla connessione

### Struttura Segmento UDP

Num src-port	Num dest-port
lung	Checksum
Messaggio	

Intestazione  
di campo, ognuno 2 byte

- I numeri di porta generiamo di fare nel modo corretto il Demand-pairing
- Lunghezza: definisce in byte la lung. del segmento
- Checksum: usato per verificare se ci sono errori nel segmento (il checksum è calcolato)

Il checksum controlla forse per no XOR per il controllo (superficie) degli errori

Il checksum serve per il rilevamento errori, e questo per vedere se i bit delle regole sono stati alterati durante il trasferimento.

Collegamento

Client

