

# Reti di Calcolatori e Laboratorio

Federico Matteoni

## Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>	<b>12</b>	<b>HTTP</b>	<b>18</b>
2.1	Tipi di Rete . . . . .	4	12.1	HTTP URL . . . . .	18
2.2	Internetwork . . . . .	4	12.2	Caratteristiche . . . . .	18
2.3	Switching . . . . .	5	12.2.1	Modello . . . . .	18
2.4			12.2.2	Connessioni . . . . .	18
3.1	Enti Ufficiali . . . . .	6	12.3	Esempio HTTP . . . . .	19
3.2	Reti di accesso . . . . .	7	12.4	Messaggi HTTP . . . . .	19
3.3			12.5	Header . . . . .	20
3.4			12.5.1	Request header . . . . .	21
3.5			12.6	HTTP Response . . . . .	23
3.6			12.6.1	Response Headers . . . . .	23
3.7			12.7	Negoziazione del contenuto . . . . .	23
3.8			12.7.1	Entity Headers . . . . .	24
<b>4</b>	<b>Metriche di Riferimento</b>	<b>7</b>	<b>13</b>	<b>Web Caching</b>	<b>24</b>
<b>5</b>	<b>Modelli Stratificati</b>	<b>8</b>	<b>14</b>	<b>Cookies</b>	<b>24</b>
5.1	Perché stratificare . . . . .	8	<b>15</b>	<b>Telnet</b>	<b>25</b>
5.2	Smistamento Intermedio . . . . .	9	15.1	Introduzione . . . . .	25
5.3	Elementi fondamentali . . . . .	9	15.2	Protocollo Telnet . . . . .	25
5.4	Modalità di Servizio . . . . .	9	15.3	NVT . . . . .	26
5.5	Vantaggi . . . . .	9	15.4	Architettura . . . . .	27
5.6			15.5	Funzionamento . . . . .	28
<b>6</b>	<b>Protocolli</b>	<b>10</b>	<b>16</b>	<b>SSH</b>	<b>29</b>
6.1	Incapsulamento . . . . .	10	<b>17</b>	<b>TCP Port Forwarding</b>	<b>29</b>
<b>7</b>	<b>OSI RM (Open Systems Interconnection Reference Model)</b>	<b>10</b>	<b>18</b>	<b>FTP</b>	<b>30</b>
7.0.1	Pila di protocolli . . . . .	10	18.1	Modello FTP . . . . .	30
<b>8</b>	<b>Flusso dell'Informazione</b>	<b>11</b>	18.1.1	Connessione di Controllo . . . . .	30
<b>9</b>	<b>Stack protocollare TCP/IP</b>	<b>12</b>	18.1.2	Connessione Dati . . . . .	30
9.1	I Livelli . . . . .	12	18.2	Altri Dettagli . . . . .	31
<b>10</b>	<b>Livello Applicativo</b>	<b>13</b>	18.2.1	Due Modalità . . . . .	31
10.1	Protocollo a Livello Applicativo . . . . .	13	18.2.2	Stateful . . . . .	32
10.2	Paradigmi . . . . .	13	18.2.3	Modalità di trasmissione . . . . .	32
10.3	Componenti di un'Applicazione di Rete . . . . .	13	18.3	Anonymous FTP . . . . .	32
10.4	Terminologia . . . . .	14	<b>19</b>	<b>DNS</b>	<b>33</b>
10.5	Identificazione di un Processo . . . . .	14	19.1	Motivazioni . . . . .	33
10.6	Esempio di API: TCP . . . . .	14	19.2	Struttura . . . . .	33
10.7	Uso dei Servizi di Trasporto . . . . .	15	19.3	Servizi . . . . .	34
<b>11</b>	<b>Applicazioni Web e HTTP</b>	<b>16</b>	19.4	Spazio dei nomi . . . . .	34
11.1	Terminologia . . . . .	16	19.4.1	Indirizzi . . . . .	34
11.2	Uniform Resource Identifier . . . . .	16	19.4.2	Nomi . . . . .	34
11.2.1	Sintassi . . . . .	17	19.4.3	Top-Level Domains . . . . .	36
11.2.2	Assolute e Relative . . . . .	17	19.4.4	Struttura di un nome alfanumerico . . . . .	36
			19.5	Conversione . . . . .	37
			19.5.1	Name Servers . . . . .	37
			19.5.2	Root Name Servers . . . . .	38

19.5.3 Gerarchia dei server . . . . .	38	23.4 Trasferimento bufferizzato . . . . .	56
19.6 Risoluzione dei nomi . . . . .	39	23.5 Segmenti TCP . . . . .	56
19.7 Caching e Aggiornamento Record . . . . .	40	23.6 Numeri di sequenza e di riscontro . . . . .	57
19.8 Record DNS . . . . .	40	23.7 Segmento TCP . . . . .	57
19.9 Messaggi DNS . . . . .	40	23.7.1 Campi del segmento TCP . . . . .	58
<b>20 SMTP</b>	<b>42</b>	23.7.2 Formato del segmento TCP . . . . .	59
20.1 Agenti Utente . . . . .	42	23.8 Gestione della connessione . . . . .	59
20.2 Mail Server . . . . .	42	23.8.1 Three-Way Handshake . . . . .	59
20.3 Schema di principio . . . . .	43	23.8.2 Perché a tre vie . . . . .	60
20.4 Indirizzo . . . . .	43	23.8.3 Esempio . . . . .	60
20.5 Gestione Alias . . . . .	44	23.8.4 Chiusura della connessione con handshake . . . . .	61
20.6 Modello di riferimento . . . . .	45	23.9 Half-Close . . . . .	62
20.7 Simple Mail Transfer Protocol . . . . .	45	23.9.1 Scenario Half-Close . . . . .	63
20.8 SMTP Mail Relaying . . . . .	45	23.9.2 Stato Time-Wait . . . . .	63
20.9 Modello SMTP . . . . .	46	23.10 Stati del TCP . . . . .	64
20.9.1 Fallimenti nella consegna . . . . .	46	23.11 Trasferimento Dati Affidabile . . . . .	65
20.10 Protocollo . . . . .	46	23.11.1 Sequenza e riscontro . . . . .	65
20.11 Comandi SMTP . . . . .	47	23.11.2 Eventi lato mittente . . . . .	65
20.12 Esempio . . . . .	47	23.11.3 Eventi lato destinatario . . . . .	66
20.13 Formato messaggi mail . . . . .	48	23.11.4 Esempi . . . . .	67
20.14 Estensioni multimediali . . . . .	49	23.12 Calcolo del timeout . . . . .	71
20.14.1 MIME . . . . .	49	23.13 Finestra di trasmissione . . . . .	71
20.14.2 Tipi MIME . . . . .	50	23.14 Controllo della congestione . . . . .	73
20.15 Protocolli di accesso alla mail . . . . .	50	23.14.1 Algoritmo per il controllo della congestione . . . . .	74
<b>21 Livello di Trasporto</b>	<b>51</b>	23.14.2 cWnd . . . . .	74
21.1 Obiettivi . . . . .	51	23.14.3 AIMD . . . . .	74
21.2 Caratteristiche . . . . .	52	23.14.4 Slow Start . . . . .	75
21.3 Servizi offerti . . . . .	52	23.14.5 Politica Reno per il controllo della congestione . . . . .	76
<b>22 UDP</b>	<b>52</b>	23.14.6 Politica Tahoe per il controllo della congestione . . . . .	77
22.1 Proprietà . . . . .	52	23.15 Throughput . . . . .	78
22.2 Datagramma UDP . . . . .	53	23.16 Fairness . . . . .	78
22.3 Calcolo del checksum . . . . .	53	23.17 Transmission Control Block . . . . .	78
22.4 TCP vs UDP . . . . .	54		
<b>23 TCP</b>	<b>55</b>		
23.1 Proprietà . . . . .	55	<b>24 Esercizi</b>	<b>79</b>
23.2 Funzioni del segmento TCP . . . . .	55	24.1 SMTP . . . . .	79
23.3 Processi e Socket TPC . . . . .	56	24.2 DNS . . . . .	80
		24.3 TCP . . . . .	81

# 1 Introduzione

Appunti del corso di **Reti di Calcolatori** presi a lezione da **Federico Matteoni**.

Prof.: **Federica Paganelli**, federica.paganelli@unipi.it

Riferimenti web:

- [elearning.di.unipi.it/enrol/index.php?id=169](http://elearning.di.unipi.it/enrol/index.php?id=169)

Password: **RETI2019**

Esame: scritto (o compitini), discussione orale facoltativa + progetto con discussione (progetto + teoria di laboratorio, progetto da consegnare 7gg prima della discussione)

Libri e materiale didattico:

- Slide su eLearning
- IETF RFC
  - tools.ietf.org/rfc
  - www.ietf.org/rfc.html
- "Computer Networks: A Top-Down Approach" B. A. Forouzan, F. Mosharraf, McGraw Hill

Ricevimento: stanza 355 DO, II piano

Immagini: Copyright 1996–2005 J.F Kurose and K.W. Ross

## 2 Rete

**Definizione di rete** Interconnessione di dispositivi in grado di scambiarsi informazioni, come end system, router, switch e modem.

Gli end system possono essere di due tipi:

**Host**: una macchina, in genere di proprietà degli utenti, **dedicata ad eseguire applicazioni**. Esempi: desktop, portatile, smartphone, tablet...

**Server**: una macchina, tipicamente con elevate prestazioni, destinata ad eseguire programmi che **forniscono servizi** a diverse applicazioni utente. Esempi: posta elettronica, web, ...

Con il termine host si può anche indicare un server.

### 2.1 Tipi di Rete

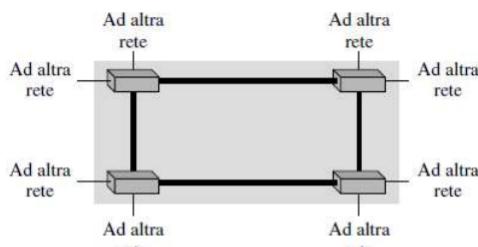
**Local Area Network** Una LAN è una rete di area geografica limitata: un ufficio, una casa ecc.. I dispositivi comunicano attraverso una determinata tecnologica: switch, BUS, HUB ecc..

In una rete locale tipicamente una serie di host comunicano tra loro attraverso, ad esempio, uno switch centrale.

**Wide Area Network** Alcuni esempi:



Legenda  
■ Dispositivo di comunicazione  
— Mezzo trasmissivo

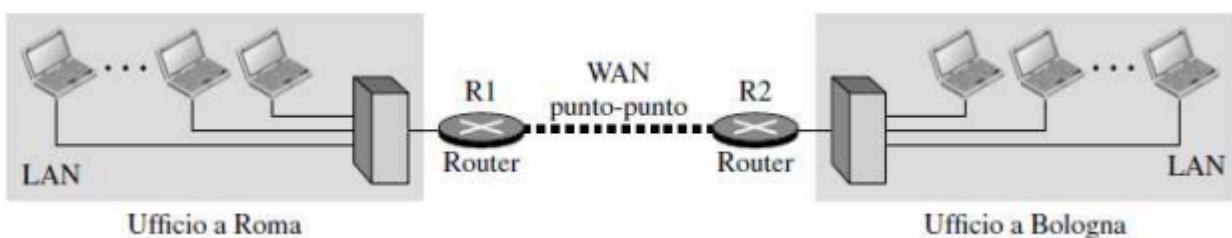


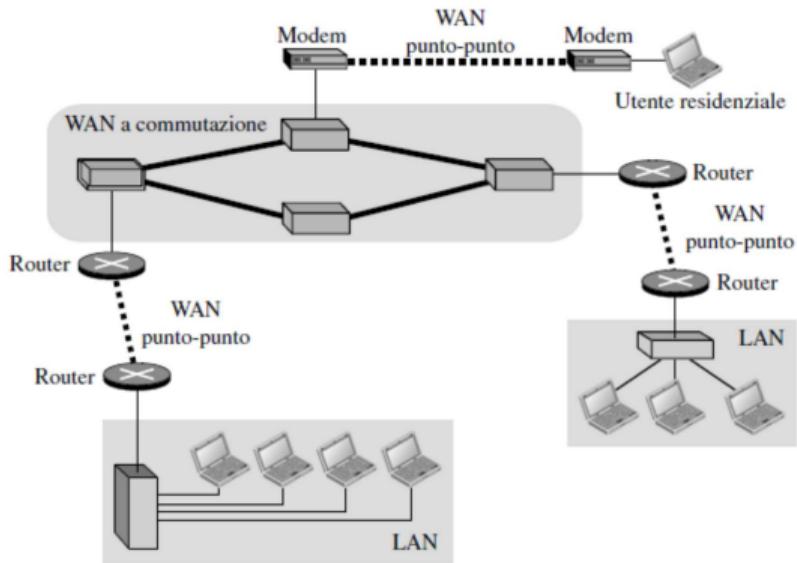
Legenda

■ Switch  
— Mezzo trasmissivo

### 2.2 Internetwork

Una internetwork si crea quando si interconnettono diverse reti. Alcuni esempi:





## 2.3 Switching

Una rete internet è formata dall'interconnessione di reti composte da link e dispositivi capaci di scambiarsi informazioni. In particolare, i sistemi terminali comunicano tra di loro per mezzo di dispositivi come switch, router ecc. che si trovano nel percorso tra i sistemi sorgente e destinazione.

**Switched Network** Reti a commutazione di circuito, tipico delle vecchie reti telefoniche

Le risorse sono riservate end-to-end per una connessione. Le risorse di rete (es. bandwidth) vengono suddivise in pezzi, e ciascun pezzo è allocato ai vari collegamenti. Le risorse rimangono inattive se non vengono utilizzate, cioè **non c'è condivisione**. L'allocazione della rete rende necessario un setup della comunicazione.

A tutti gli effetti vi è un circuito dedicato per tutta la durata della connessione. Ciò è rende poco flessibile l'utilizzo delle risorse (**overprovisioning**).

**Packet-Switched Network** Reti a commutazione di pacchetto, più moderno

Flusso di dati punto-punto suddiviso in pacchetti. I pacchetti degli utenti condividono le risorse di rete. Ciascun pacchetto utilizza completamente il canale.

**Store and Forward:** il commutatore deve ricevere l'intero pacchetto prima di ritrasmetterlo in uscita.

Le risorse vengono usate **a seconda delle necessità**. Vi è **contesa per le risorse**: la richiesta di risorse può eccedere la disponibilità e si può verificare **congestione** quando i pacchetti vengono accodati in attesa di utilizzare il collegamento. Si possono anche verificare perdite.

## 3 Internet

L'internet più famosa ed utilizzata è **internet**, ed è composta da migliaia di reti interconnesse. **Ogni rete** connessa ad internet **deve utilizzare il protocollo IP** e rispettare certe convenzioni su nomi ed indirizzi. Si possono aggiungere nuove reti ad internet molto facilmente.

**Dispositivi in internet** I **dispositivi** connessi ad internet possono essere host, end systems come PC, workstations, servers, pda, smartphones ecc...

I **link di comunicazione** possono essere fibre ottiche, doppini telefonici, cavi coassiali, onde radio... Le **entità software** in internet possono essere:

**Applicazioni** e processi

**Protocolli:** regolamentano la trasmissione e la ricezione di messaggi (TCP, IP, HTTP, FTP, PPP...)

**Interfacce**

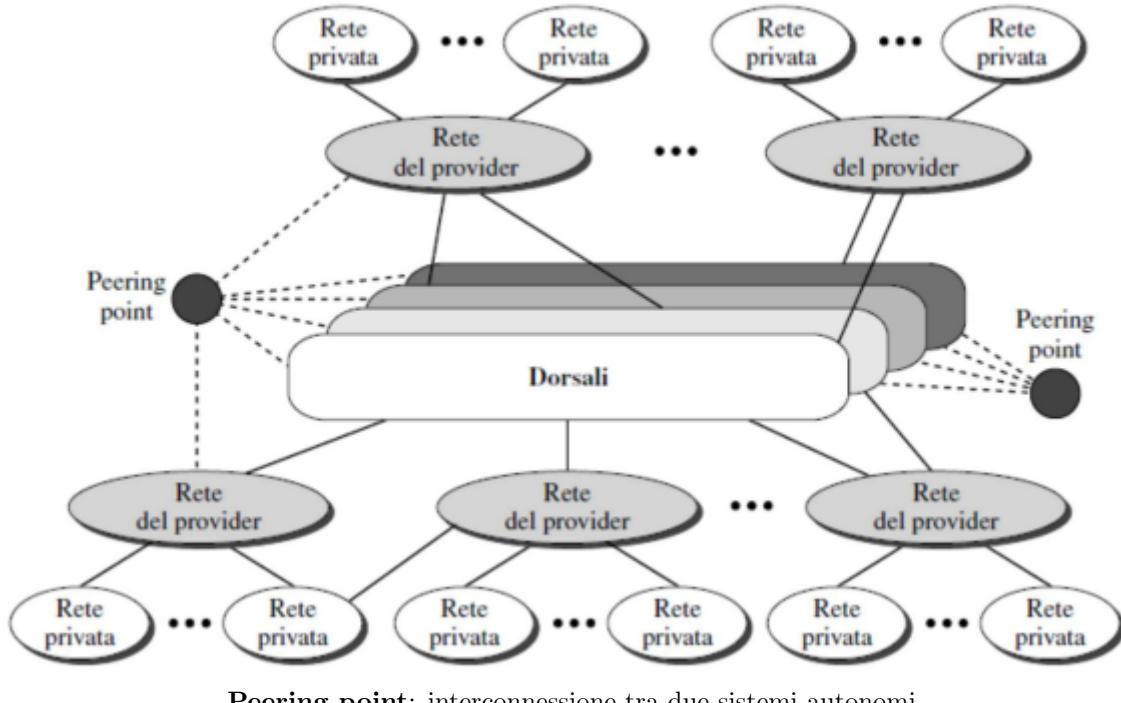
**Standard** di internet e del web: RFC (Request for Comments) e W3C.

**Internet è una visione dei servizi.** L'infrastruttura di comunicazione permette alle applicazioni distribuite di scambiare informazioni (WWW, e-mail, giochi, e-commerce, controllo remoto...) e fornisce loro **servizi di comunicazione connectionless** (senza garanzia di consegna) o **connection-oriented** (dati garantiti in integrità, completezza ed ordine).

### 3.1 Enti Ufficiali

L'**Internet Engineering Task Force** (IETF) è l'organismo che studia e sviluppa i protocolli in uso su internet. Si basa su gruppi di lavoro a cui chiunque può accedere. I documenti ufficiali che pubblica, dove descrivono i protocolli usati in internet, sono gli RFC/STD (Request for Comments/STanDards).

L'**Internet Corporation for Assigned Names and Numbers** (ICANN) si occupa di coordinare il sistema dei **nomi di dominio** (DNS) e assegna i gruppi di indirizzi, gli identificativi di protocollo.



### 3.2 Reti di accesso

Il collegamento tra l'utente ed il primo router di internet è detto **rete di accesso**. Può avvenire in 3 modi:

**Tramite rete telefonica:** servizio dial-up, ADSL...

**Tramite reti wireless**

**Collegamento diretto**, come collegamenti WAN dedicati ad alta velocità (aziende e università)

## 4 Metriche di Riferimento

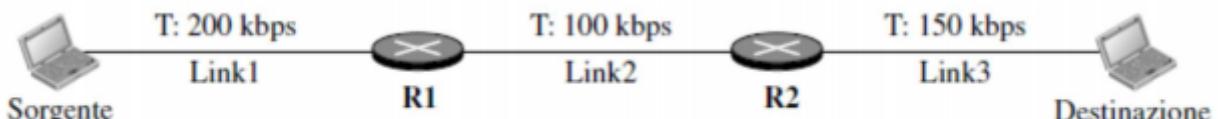
Come misurare le prestazioni della rete? Tramite una serie di metriche:

**Bandwidth** o ampiezza di banda: è la larghezza dell'intervallo di frequenze utilizzato dal sistema trasmissivo (Hz).

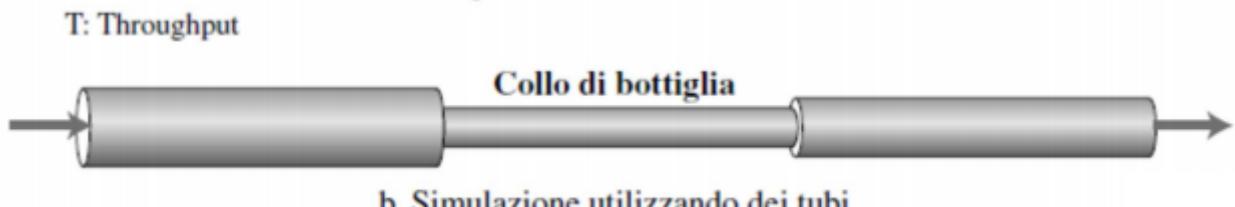
**Bitrate** o **transmission rate**: quantità di bit che possono essere trasmessi o ricevuti nell'unità di tempo (bit/secondo, bps)

Il bitrate dipende dalla bandwidth e dalla tecnica trasmissiva utilizzata.

**Throughput**: la quantità di traffico che arriva realmente a destinazione nell'unità di tempo (al netto di perdite sulla rete, funzionamento dei protocolli ecc...).



a. Un percorso attraverso tre link



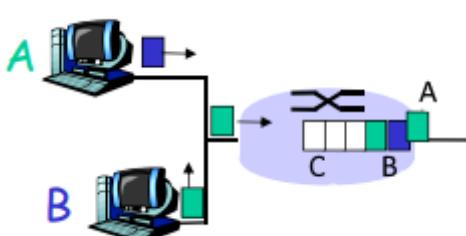
b. Simulazione utilizzando dei tubi

Non è detto che corrisponda alla bandwidth perché ci potrebbe essere un collo di bottiglia.

**Latenza** o ritardo: il tempo richiesto affinché un messaggio arrivi a destinazione dal momento in cui il primo bit parte dalla sorgente.

latenza = ritardo di propagazione + ritardo di trasmissione + ritardo di accodamento + ritardo di elaborazione

**Perdita di pacchetti**. Come si può verificare?



A → pacchetti **in attesa** di essere trasmessi (*ritardo*)

B → pacchetti **accodati** (*ritardo*)

C → buffer **liberi** (se non ci sono buffer liberi, i pacchetti in arrivo vengono scartati, *perdita*)

I pacchetti da spedire vengono accodati nei buffer dei router. Di solito, il tasso di arrivo dei pacchetti sul router eccede le capacità del router di evaderli, quindi i **pacchetti si accodano in attesa del proprio turno**.

Il **ritardo di elaborazione** è dato dal controllo sui bit e dalla determinazione del canale di uscita (trascutibile)

Il **ritardo di accodamento** è dato dall'attesa di un pacchetto di essere trasmesso (B)

Il **ritardo di trasmissione** è il tempo impiegato per trasmettere un pacchetto sul link.

$$R_{trasmissione} = R/L$$

R = rate di trasmissione del collegamento, in bps

$L$  = lunghezza del pacchetto in bit

Il **ritardo di propagazione** è il tempo impiegato da 1 bit per essere propagato da un nodo all'altro.

$R_{propagazione} = d/s$

$d$  = lunghezza del collegamento

$s$  = velocità di propagazione del collegamento (si usa la velocità della luce, circa  $3 \times 10^8$  m/s)

$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$

$d_{proc}$  = ritardo di elaborazione, pochi microsecondi

$d_{queue}$  = ritardo di accodamento, dipende dalla congestione

$d_{trans}$  = ritardo di trasmissione, L/R e significativo a lunga distanza

$d_{prop}$  = ritardo di propagazione, d/s, da pochi microsecondi a centinaia di millisecondi

## 5 Modelli Stratificati

[https://elearning.di.unipi.it/pluginfile.php/27387/mod\\_resource/content/1/L02\\_introduzione\\_protocolli.pdf](https://elearning.di.unipi.it/pluginfile.php/27387/mod_resource/content/1/L02_introduzione_protocolli.pdf)

**Perché usare un modello a strati** Per mandare dei dati da un host ad un altro comunicando su una rete, si devono eseguire una serie di operazioni: trovare il percorso di rete da attraversare, decidere in che modo spedire e codificare i dati, risolvere eventuali problemi di comunicazione e altro ancora.

Programmare ogni volta tutto il procedimento è un lavoro estremamente complesso e ripetitivo. Un modello a strati **astrae su più livelli il problema della trasmissione dati** in modo da fornire di volta in volta strumenti utili al programmatore per poter evitare di "reinventare la ruota".

**Definizioni generali** Nelle architetture di comunicazione a strati sono importanti una serie di definizioni:

- Stratificazione
- Information hiding
- Separation of concern
- Modello ISO/OSI
- Stack TCP/IP

Tali definizioni verranno viste durante il corso.

**Lo Strato** Uno strato è un **modulo interamente definito** attraverso i servizi, le interfacce e i protocolli che lo caratterizzano. Si indica anche col nome di livello.

Uno strato **n comunica direttamente** con lo strato **n** di un'altra unità tramite un **protocollo assegnato**. Lo stesso strato **n** può richiedere servizi allo strato **n-1** attraverso la **loro interfaccia**, e fornisce servizi allo strato **n+1** attraverso la **rispettiva interfaccia**.

**Es. modello stratificato: sistema postale** Vedi slide

[https://elearning.di.unipi.it/pluginfile.php/27387/mod\\_resource/content/1/L02\\_introduzione\\_protocolli.pdf](https://elearning.di.unipi.it/pluginfile.php/27387/mod_resource/content/1/L02_introduzione_protocolli.pdf), 48

Dal livello più alto al livello più basso per la spedizione, viceversa per la ricezione. Un problema importante che si incontra quando si manda una lettera, ad esempio, dall'Italia al Giappone è la traduzione. In una **serie di passi**, in cui in ognuno viene **eseguito un particolare compito su un messaggio**, che viene poi **trasferito ad un altro livello**. Nell'esempio, la segretaria prepara lettera (traduce in giapponese e imbusta) affinché il postino la possa prendere. Però il "messaggio" della segretaria è "scritto" per essere interpretato dalla segretaria giapponese, il direttore italiano scrive per il direttore giapponese. **Messaggi di un livello del sistema che spedisce sono scritti per essere interpretati dal medesimo livello del sistema ricevente.**

### 5.1 Perché stratificare

La stratificazione è molto utile per **scomporre il sistema complesso della gestione della comunicazione**. Prendo un sistema estremamente costoso da costruire per una singola coppia di aziende, quindi lo trasformo in strati così che il costo della singola lettera sia irrisorio.

**Definisco funzioni di base per effettuare trasferimento e agenti che le svolgono.** Principi di base:

### Separation of Concern

Far fare ad un determinato strato solo ciò che gli compete, delegando agli altri tutto ciò che è delegabile

### Information Hiding

Nascondo ad un determinato strato le informazioni non indispensabili allo svolgere della sua operazione.

**Esempio** Se traduco il modello postale nel modello a strati ho, ad esempio:



## 5.2 Smistamento Intermedio

Spedire un pacchetto che è destinato ad determinato livello intermedio. Quindi **arrivo fino al corrispondente livello intermedio per evitare che si possano esporre info sensibili**.

## 5.3 Elementi fondamentali

Gli **elementi fondamentali** del modello stratificato sono:

Flusso dati

**Servizio:** una **funzione** che uno strato offre allo strato superiore, attraverso un'interfaccia.

Protocollo

**Interfaccia:** insieme di regole che governano il formato e il significato dei frame, pacchetti o messaggi che vengono scambiati tra strati adiacenti della stessa entità.

I servizi indicano *cosa* si può fare, le interfaccie regolano *come* si può fare.

## 5.4 Modalità di Servizio

I **dati** possono essere scambiati in due modalità diverse:

**Connection-Oriented:** il livello di trasferimento stabilisce una **connessione logica** tra due sistemi. La connessione è quindi **gestita**:

- **Instaurazione** della connessione
- **Trasferimento** dei dati
- **Chiusura** della connessione

**Connectionless:** i dati vengono **trasferiti senza stabilire una connessione**.

## 5.5 Vantaggi

Il vantaggio più grosso è che **sviluppare il singolo strato è più semplice ed economico rispetto a sviluppare tutto il sistema complesso**. Questo perché i servizi degli strati inferiori vengono usati da più entità che implementano gli strati superiori.

## 6 Protocolli

**Cos'è un protocollo** Un **protocollo** è un **insieme di regole** che dice come comunicare ed esporre dati verso l'esterno. I protocolli **definiscono il formato e l'ordine dei messaggi inviati e ricevuti, con le azioni per trasmettere e ricevere tali messaggi.**

### 6.1 Incapsulamento

Processo in cui aggiungo strati, "involucri" al messaggio originale che vengono man mano tolti alla destinazione.

## 7 OSI RM (Open Systems Interconnection Reference Model)

Le prime reti erano chiuse, composte da tecnologie e protocolli proprietari. Alcuni esempi sono ARPANET, SNA (IBM), DNA (Digital). Non potevano intercomunicare tra loro, perché usavano **protocolli diversi**, erano costruite per servizi specifici (TELCO). Insorse quindi un obiettivo: creare un **modello di riferimento per sistemi aperti**, per permettere a qualsiasi terminale di poter comunicare mediante qualsiasi rete. C'era quindi necessità di **accordarsi sulle regole**.

**OSI** L'OSI è una **collezione di protocolli aperti**: questo significa che i loro **dettagli sono pubblici** e i **cambiamenti vengono gestiti da un'organizzazione con partecipazione aperta al pubblico**. Un sistema che implementa i protocolli aperti è un **sistema aperto**.

### 7.0.1 Pila di protocolli

L'OSI prevede **sette strati di protocolli**:

7. **Applicativo**: elaborazione dei dati
6. **Presentazione**: unificazione dei dati, preparazione del **pacchetto** da trasmettere/ricevere
5. **Sessione**: controllo del dialogo tra gli host sorgente e destinazione.
4. **Trasporto**: offre il vero e proprio trasferimento dati tra gli host terminali, cioè **astrae la logica** con la quale si scambiano i dati tra host e gestisce gli errori. Realizza il **dialogo end-to-end**.
3. **Rete**: instradamento del traffico (principalmente router, offre il servizio di consegna attraverso il sistema distribuito dei nodi intermedi).
2. **Datalink**: consegna il frame tra le interfacce, interpretato dalla scheda di rete.
1. **Fisico**: modulazione del segnale elettrico per trasmettere correttamente il flusso di bit sul mezzo fisico.

I livelli **7-5** si possono raggruppare in più modi, principalmente sono gli strati di supporto all'elaborazione e all'interazione con l'utente. Sono livelli **software**

I livelli **4-1** sono **software e hardware**. Sono strati di supporto alla rete e all'infrastruttura trasmittiva, cioè gestiscono la vera e propria trasmissione dei dati.

## 8 Flusso dell'Informazione

Per le reti, l'informazione ha origine al **livello applicativo**, che la genera per mandarla in remoto. Una volta generata, essa discende i vari livelli fino al canale fisico e ogni livello **aggiunge** all'informazione ricevuta dal livello superiore **una – o più – propria sezione informativa** sotto forma di **header**, contenente informazioni esclusive di quel livello. Per l'informazione ricevuta, si segue il **cammino inverso**, quindi dal basso verso il livello applicativo, e ogni livello "spacchetta" l'header del livello corrispondente, ne legge le informazioni esclusive e lo gestisce appositamente. Il **processo di encapsulamento** è quindi **reversibile**: ogni livello esegue una operazione di encapsulamento su dati già encapsulati dal livello precedente, in modo tale da garantire la possibilità di estrarre i dati precedentemente encapsulati.

---

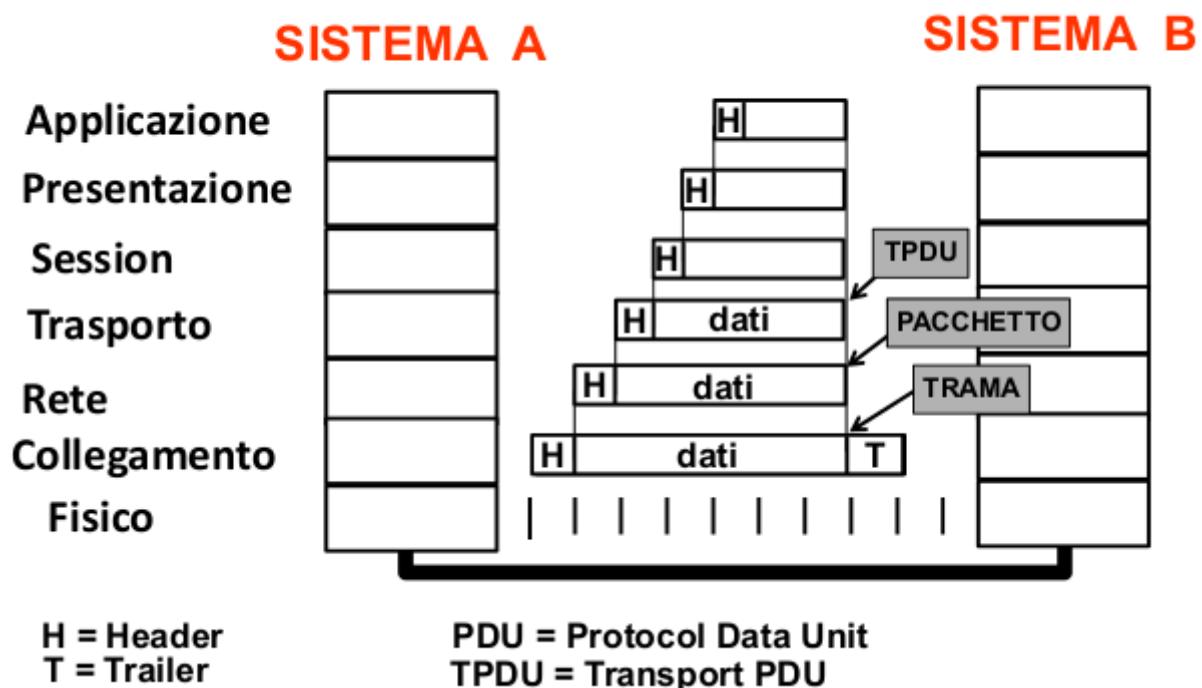
HEADER —— DATA —— TRAILER

---

**Header:** qualificazione del pacchetto per questo livello

**Data:** payload proveniente dal livello superiore

**Trailer:** generalmente usato in funzione del trattamento dell'errore



## 9 Stack protocollare TCP/IP

Il **TCP/IP** è una famiglia di protocolli attualmente utilizzati in internet. Si tratta di una **gerarchia di protocolli** costituita da **moduli interagenti**, ciascuno con funzioni specifiche.

**Gerarchia** Con il termine **gerarchia** s'intende che ciascun protocollo di livello superiore è **supportato dai servizi forniti dai protocolli di livello inferiore**. Cioè un protocollo a livello  $n$  realizza le sue funzionalità grazie ai protocolli a livello  $n-1$ .

### 9.1 I Livelli

Lo stack TCP/IP in origine era intesa come **quattro livelli software** sovrastanti **un livello hardware**. Oggi è intesa come semplicemente **composta da 5 livelli**



**Livello Applicativo** Il livello più alto, con il quale interagisce l'utente

Identificativi risorse: URL, URI, URN

Il web: user agents, http: request, response, connessioni persistenti, GET, POST, PUT, DELETE, status code, proxy server, caching

FTP: connessioni dati e di controllo, rappresentazione TELNET

Posta elettronica: SMTP, POP3, IMAP

DNS e risoluzioni nomi: gerarchia nomi, risoluzione iterativa e ricorsiva, formato messaggi, nslookup...

**Livello Trasporto** Livello al quale si definisce la codifica e il protocollo di trasporto

Servizi: mux demux, controllo errore, connectionless

TCP: formato segmenti, gestione connessione, controllo flusso e congestione

UDP: formato segmenti

**Livello Rete** Dove si gestisce l'indirizzamento dei vari host

Strato di rete e funzioni

Indirizzamento IP: classful IPv4, NAT, sottoreti e maschere, classless, CIDR

Risoluzione IP e MAC, ARP

IPv4: formato datagramma ip, frammentazione

Routing IP e istradamento

Introduzione IPv6

**Livello Link** Trasferimento dati tra elementi di rete vicini

Ethernet

**Livello Fisico** Bit sul filo

## 10 Livello Applicativo

**Applicazioni e processi** Le applicazioni possono essere composte da **vari processi distribuiti** comunicanti fra loro. Un **processo** sono programmi eseguiti degli host di una rete. Due processi possono anche comunicare all'interno dello stesso host attraverso la **comunicazione inter-processo** (definita dal S.O.).

Nella **comunicazione a livello applicativo fra due host di una rete**, due o più processi vengono eseguiti da ciascuno degli host comunicanti e **si scambiano messaggi**.

I livelli applicazione dei due host **si comportano come se esistesse un collegamento diretto** attraverso cui mandare e ricevere messaggi.

### 10.1 Protocollo a Livello Applicativo

Un protocollo di livello applicativo **definisce**:

**Tipi di messaggi** scambiati a quel livello, ad esempio di richiesta o di risposta

**Sintassi** dei vari tipi di messaggi, cioè i campi

**Semantica** dei campi

**Regole** per determinare quando e come un processo invia o risponde ai messaggi

### 10.2 Paradigmi

I due programmi applicativi devono essere entrambi in grado di richiedere e offrire servizi oppure ognuno deve occuparsi di uno dei due compiti?

**Client-Server** Un **numero limitato di processi server** che **offrono** un servizio, in esecuzione **in attesa di richieste** dai **processi client**, che **richiedono** servizi.

**Client** *Parla per primo*, cioè inizia il contatto con il server. Tipicamente richiede un servizio al server, ad esempio: per il web il client è implementato nel browser, per l'e-mail è implementato nel mail reader.

**Server** Fornisce al client il servizio richiesto e **rimane sempre attivo**. Ad esempio: un web server invia le pagine richieste, un mail server smista ed invia le mail.

**Peer-to-Peer** Host **peer** che possono **offrire servizi e inviare richieste**.

**Misto** Un misto tra i due paradigmi sopra.

### 10.3 Componenti di un'Applicazione di Rete

Due esempi

**Web** Composto da:

- Web Browser, sul **client**
- Web **Server**
- **Standard per il formato** delle risorse (pagine ecc.)
- **Protocollo HTTP**

**Posta Elettronica** Composta da:

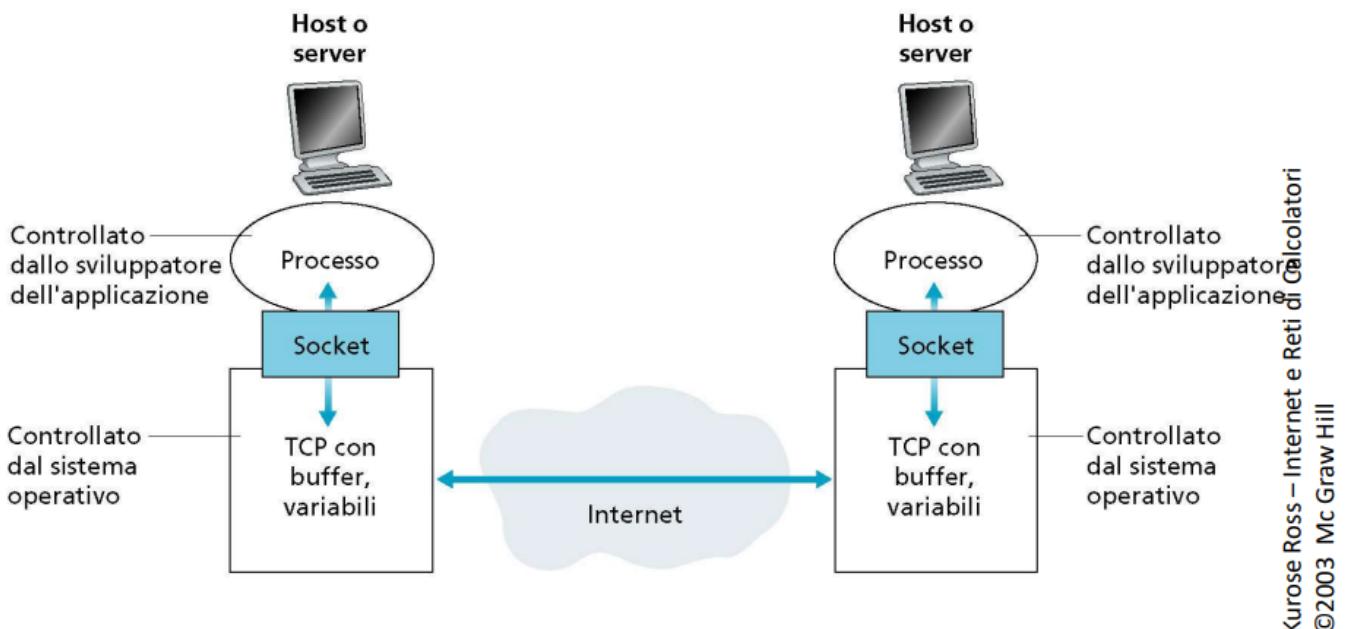
- Programmi di lettura e scrittura sul **client**
- **Server** di posta in internet
- **Standard per il formato** dei messaggi
- **Protocolli SMTP, POP3 ecc.**

## 10.4 Terminologia

**API Application Programming Interface:** si tratta di un **insieme di regole** che un programmatore deve rispettare per utilizzare delle risorse.

**Socket** Una **API** che funge da **interfaccia** tra gli strati di applicazione e di trasporto. A tutti gli effetti è la **API di internet per eccellenza**, due processi comunicano mandando dati sul socket e leggendoli da esso. Forma una **connessione logica**, l'invio e ricezione dei dati sono responsabilità del S.O. e del TCP/IP.

## 10.5 Identificazione di un Processo



I servizi di trasporto sono offerti al livello applicativo tramite le API. Ogni servizio di transport è **usato simultaneamente** da più processi application.

Come identifico i processi di livello application **di host diversi**? Serve un identificativo che identifichi sia l'host che il processo.

→ Coppie <Indirizzo IP, Numero di porta>



## 10.6 Esempio di API: TCP

```

Connection TCPopen(IPAddress, int)           //per aprire una connessione
void TCPsend(Connection, Data)                //per spedire dati su una connessione
Data TCPreceive(Connection)                  //per ricevere dati da una connessione
void TCPclose(Connection)                   //per chiudere una connessione
int TCPbind(int)                           //per richiedere l'assegnazione della porta su
                                         //cui attendere le richieste di connessione

void TCPunbind(int)                        //per liberare la porta assegnata
Connection TCPaccept(int)                 //per attendere le richieste di connessione

//Connection: identificata da una quadrupla
//Astraggo dalle possibili eccezioni sollevate e dal loro trattamento

```

## 10.7 Uso dei Servizi di Trasporto

Una coppia di processi fornisce servizi agli utenti di Internet, siano questi persone o applicazioni. La coppia di processi, tuttavia, **deve utilizzare i servizi offerti dal livello di trasporto** per la comunicazione, poiché non vi è una comunicazione fisica a livello applicativo. Le applicazioni di rete sono quindi **realizzate sopra ai servizi di trasporto dati**.

Nel livello trasporto dello stack protocollare TCP/IP sono previsti **due protocolli di trasporto principali**:

**TCP** Transfer Control Protocol

**Connection-Oriented**: è richiesto un setup tra client e server

Trasporto **affidabile** tra mittente e destinatario

**Controllo del flusso**: il mittente non *inonderà* di dati il destinatario

**Controllo di congestione**: *limita* il mittente quando la rete è satura

Non offre garanzie di timing né di ampiezza minima di banda

**UDP** User Datagram Protocol

**Connectionless**

Trasporto **non affidabile**

**NO** controllo del flusso

**NO** controllo di congestione

**NO** garanzie di timing o di ampiezza minima di banda

*Quindi quali applicazioni usano UDP, e perché?*

**Che tipo di trasporto richiede un'applicazione?**

**Throughput** Anche detta **banda**, è la frequenza alla quale il processo mittente può inviare i bit al processo ricevente. Alcune applicazioni (es. multimedia) richiedono una **banda minima** per essere efficaci, altri (**elastic apps**) usano la banda che trovano a disposizione.

Velocità di trasferimento  $\neq$  velocità di propagazione

**Perdita di dati** Alcune applicazioni (es. audio) possono tollerare alcune perdite, altre (es. telnet, trasferimento file) richiedono un trasferimento dati **affidabile** al 100%

**Timing** Alcune applicazioni (es. teleconferenze, videogame) richiedono un basso ritardo per essere efficaci

Applicazione	Tolleranza alla perdita dati	Throughput	Sensibilità al tempo
Trasferimento file	No	Variabile	No
Posta Elettronica	No	Variabile	No
Documenti Web	No	Variabile	No
Audio/Video in tempo reale	Si	Audio: 5Kbps – 1 Mbps Video: 10Kbps – 5MKbps	Si, centinaia di millisecondi
Audio/Video memorizzati	Si	Audio: 5Kbps – 1 Mbps Video: 10Kbps – 5MKbps	Si, pochi secondi
Videogame	Si	Fino a pochi Kbps	Si, centinaia di millisecondi
Messaggistica istantanea	No	Variabile	Si e no

Applicazione	Protocollo a Livello Applicativo	Protocollo di Trasporto
Posta Elettronica	SMTP (RFC 2821)	TCP
Accesso a terminali remoti	Telnet (RFC 854)	TCP
Web	HTTP (RFC 2616)	TCP
Trasferimento file	FTP (RFC 959)	TCP
Streaming multimediale	HTTP (es. YouTube), RTP (RFC 1889)	TCP o UDP
Telefonia internet	SIP, RTP, proprietario (es. Skype)	Tipicamente UDP

## 11 Applicazioni Web e HTTP

### 11.1 Terminologia

**WEB** Consiste di *oggetti* indirizzati da un **URL** (Uniform Resource Locator)

**Pagine Web** Solitamente formate da: *pagine WEB* (HTML, Javascript...) e diversi **oggetti referenziati** (altre pagine, immagini, script...)

**Browser** Lo user agent per il web, ad esempio: Chrome, Firefox, Netscape, Lynx

**Web server** Il server per il web, ad esempio: Apache, MS Internet Information Server

### 11.2 Uniform Resource Identifier

Una **URI** è una **forma generale per identificare una risorsa presente sulla rete** (IETF RFC 2396: *una Uniform Resource Identifier è una stringa compatta di caratteri usata per identificare una risorsa astratta o fisica*).

La sintassi di uno URI è stata progettata ponendo la **trascrivibilità globale** come uno degli obiettivi principali: utilizza caratteri da un **alfabeto molto limitato** (es. le lettere dell'alfabeto latino base, numeri e qualche carattere speciale).

Uno URI può essere **rappresentato in molti modi**, ad esempio: inchiostro su carta, pixel su schermo, sequenza di ottetti... L'**interpretazione di uno URI dipende soltanto dai caratteri utilizzati e non da come essi vengono rappresentati** nel protocollo di rete.

**Uniform** Uniformità della sintassi dell'identificatore, anche se i meccanismi per accedere alle risorse possono variare.

**Resource** Qualsiasi cosa abbia un'identità: documento, servizio, immagine, collezione di risorse...

**Identifier** Oggetto che può agire da riferimento verso qualcosa che ha identità

Esistono due tipi di URI:

**URL** Uniform Resource Locator: sottotipo di URI che identifica una risorsa attraverso il suo **meccanismo di accesso primario**, ad esempio la *posizione* nella rete.

Esempi:

URL <https://doi.org/10.1109/LCN.1988.10239>

URL <ftp://ftp.is.co.za/rfc/rfc1808.txt>

URL <https://www.apple.com/index.html>

**URN** Uniform Resource Name: sottotipo di URI che devono essere **globalmente univoci e persistenti**, anche quando la risorsa cessa di esistere o di essere disponibile.

Esempi:

URN <urn:oasis:names:specification:docbook:dtd:xml:4.1.2>:

URN <urn:doi:10.1109/LCN.1988.10239>

### 11.2.1 Sintassi

La sintassi di un URI è **organizzata gerarchicamente**, con le componenti **elencate in ordine decrescente** di importanza da sinistra a destra.

Una **URI assoluta** può essere formata da **quattro** componenti

```
<scheme>://<authority><path>?<query>
```

**<scheme> Obbligatorio**, schema per identificare la risorsa.

Lo URI scheme **definisce il namespace** dello URI, quindi potrebbe porre ulteriori vincoli su sintassi e semantica degli identificatori che usano quello schema. Nonostante molti URL scheme prendono il nome da protocolli, **questo non implica che l'unico modo di accedere la risorsa dello URL sia attraverso il protocollo specificato**.

**<authority>** Elemento gerarchico per richiamare un'authority così che la gestione del namespace definito sia delegato a quella authority. Il **nome di dominio** di un host o il suo **indirizzo IP** in notazione puntata decimale.  
**authority = [userinfo@]host[:port]**

**<path>** Contiene dati specifici per l'authority (o lo schema) e **identifica la risorsa nel contesto** di quello schema e di quella autorità. Può consistere in una sequenza di segmenti.

**<query>** L'interrogazione o i dati da passare alla risorsa richiesta

Esempi:

```
foo://example.com:8042/over/there?name=ferret#nose

    scheme = foo
    authority = example.com:8042
    path = /over/there
    query = name=ferret
    fragment = nose

urn:example:animal:ferret:nose

    scheme = urn
    path = example:animal:ferret:nose

http://maps.google.it/maps/place?q=larco+bruno+pontecorvo+pisa&hl=it

    scheme = http
    authority = maps.google.it
    path = /maps/place
    query = q=larco+bruno+pontecorvo+pisa&hl=it
```

### 11.2.2 Assolute e Relative

Le URI possono essere assolute o relative.

**URI Assoluta** Identifica una risorsa **indipendentemente dal contesto** in cui è usata.

**URI Relativa** Informazioni per **identificare una risorsa in relazione ad un'altra URL** (è priva di **scheme e authority**). **Non viaggiano sulla rete**, sono interpretate dal browser in relazione al documento di partenza.

**Esempio di URI relativa** Sia `http://a/b/c/d;p?q` il documento di partenza, allora

```
g = http://a/b/c/g
/g = http://a/g
//g = http://g
?y = http://a/b/c/?y
#s = documento corrente#s
g;x?y#s = http://a/b/c/g;x?y#s
.. = http://a/b/
.../..../g = http://a/g
```

## 12 HTTP

Lo HTTP è usato dal 1990 come **protocollo di trasferimento per il World Wide Web**. Definito nel seguente modo (RFC 2068, RFC 2616): *protocollo di livello applicazione per sistemi di informazione distribuiti, collaborativi ed impermediati*.

Protocollo **generico**, **stateless** e **object-oriented** che può essere usato per molte attività, come name server e sistemi distribuiti di gestione oggetti, attraverso l'estensione dei suoi **request methods** (comandi). Una funzionalità dell'HTTP è la rappresentazione del tipo di dati, consentendo al sistema di essere **costruito indipendentemente dai dati che vengono trasferiti**.

### 12.1 HTTP URL

Lo schema `http` è usato per accedere alla risorsa attraverso il protocollo HTTP.

**Sintassi** La sintassi per un URL `http` è:

```
http_URL = http://host[:port][path]
```

**Host**: un dominio, hostname o indirizzo IP in forma decimale puntata di Internet.

**Porta**: un numero, se omessa viene usata la porta 80.

La risorsa è localizzata nel server in ascolto per connessioni TCP su quella porta di quell'host. Il path specifica la **Request-URI**.

### 12.2 Caratteristiche

Il protocollo HTTP è un protocollo **request/response**: la **connessione** viene **iniziatata dal client**, che invia un **messaggio di request** al quale il server risponde con una **response**.

In quanto **generico e stateless** le coppie **richiesta/risposta sono indipendenti**.

#### 12.2.1 Modello

Il modello del protocollo HTTP è **client-server**:

**Client**: browser che richiede, riceve e visualizza oggetti web.

Stabilisce una connessione con il server e invia una **richiesta sotto forma di request-method, URI e versione di protocollo**, seguito da un messaggio.

**Server**: web server che invia oggetti in risposta ad una richiesta.

Accetta le connessioni e serve le richieste rispondendo con i dati richiesti.

#### 12.2.2 Connessioni

Una **connessione** è un **circuito logico di livello trasporto** stabilito tra due programmi applicativi per comunicare tra loro.

### Non-Persistent Connection http1.0: RFC 1945

Viene stabilita una **connessione TCP separata** per raggiungere ogni URL. **Aumenta il carico** sui server HTTP e **può causare congestioni su Internet** (questo perché, ad esempio, se vengono usate tante immagini allora si creano richieste multiple del solito server in un breve lasso di tempo).

### Persistent Connection http1.1: RFC 2616

Se non è indicato altrimenti, il client può **assumere che il server manterrà una connessione persistente**.

Lo standard specifica un **meccanismo con il quale** un client o un server **può segnalare la chiusura di una connessione TCP** (il campo connection nell'header). Una volta che la chiusura viene segnalata, il cliente **non deve più mandare richieste** su quella connessione.

## 12.3 Esempio HTTP

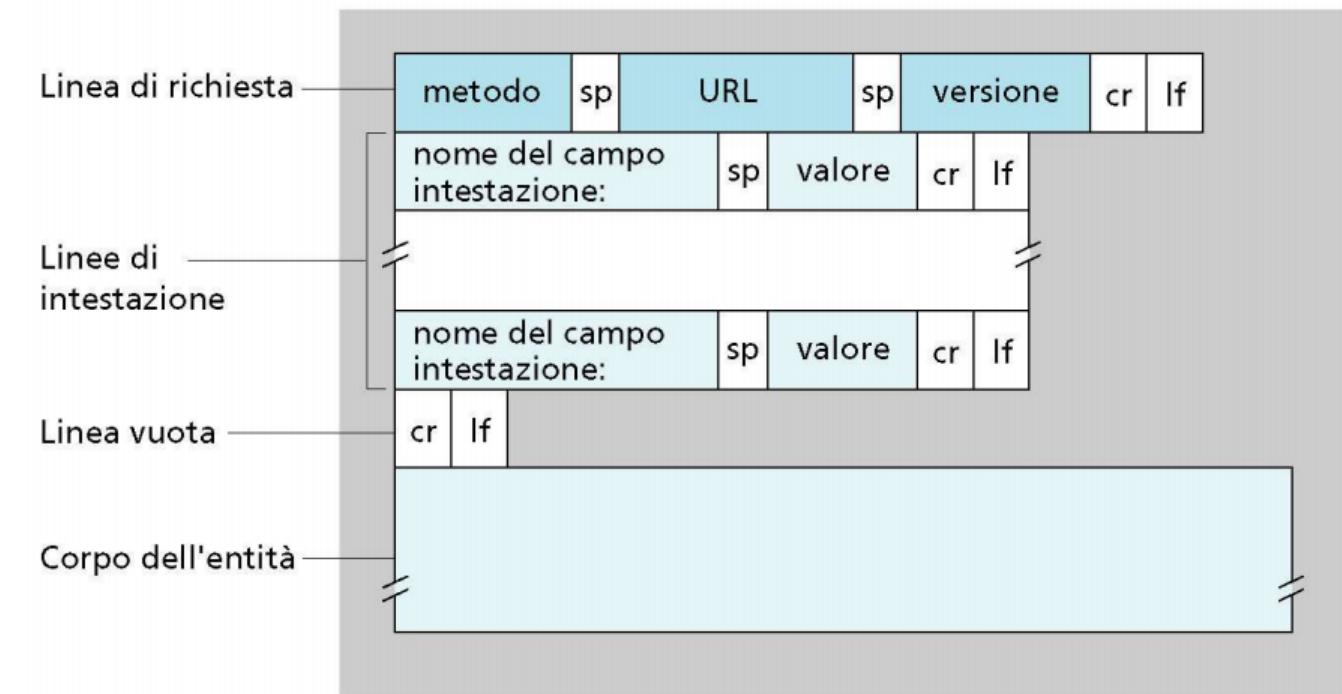
Vedi slide<sup>1</sup>

## 12.4 Messaggi HTTP

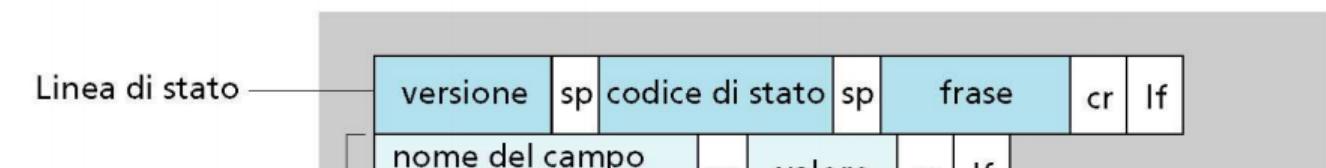
```
generic-message = start-line *message-header CRLF [message-body]
start-line = Request-Line | Status-Line
```

La **start-line** distingue **request** da **response**.

### HTTP Request Message



### HTTP Response Message



<sup>1</sup>[https://elearning.di.unipi.it/pluginfile.php/27477/mod\\_resource/content/2/L03\\_Applicativo\\_HTTP.pdf](https://elearning.di.unipi.it/pluginfile.php/27477/mod_resource/content/2/L03_Applicativo_HTTP.pdf), slide 34

**HTTP Request** Request-Line \*( general-header | request-header | entity-header ) CRLF [message-body]

Ad esempio:

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.w3.org
Connection: close
User Agent: Mozilla/4.0
Accept-language: it
```

(Body)

**HTTP Request Line** Request-Line = Method SP Request-URI SP HTTP-Version CRLF  
GET http://www.w3.org/pub/WW/TheProject.html HTTP/1.1

```
Method = GET
Request-URI = http://www.w3.org/pub/WW/TheProject.html
HTTP-Version = HTTP/1.1
Method = "OPTIONS" | "GET" | "HEAD" | "POST" | "PUT" | "DELETE" | "TRACE" | extension-method
```

**Method:** indica il **metodo** che deve essere eseguito sulla risorsa identificata dal **Request-URI**. Case sensitive.

**HTTP-Version:** indicare la versione del protocollo è pensato per consentire al mittente di indicare il formato di un messaggio e la sua capacità di capire il resto della comunicazione HTTP.

Le **URI** sono stringhe formattate in modo semplice che identificano una risorsa di rete.

## 12.5 Header

Gli **header** sono insiemi di coppie {nome:valore} che **specificano alcuni parametri** del messaggio trasmesso o ricevuto.

**General header** Relativi alla trasmissione

**Data:** data e ora di generazione del messaggio

**Connection:** consente al mittente di specificare le opzioni desiderate per quella particolare connessione. L'opzione "close" segnala che la connessione verrà chiusa al completamento della response

**Transfer-encoding:** indica quale (se presente) tipo di trasformazione è stata applicata al message body per trasferirlo correttamente dal mittente al destinatario (chunked, gzip...)

**Cache Control**

**Public** Indica che la response è cachable in qualsiasi cache

**Private** Indica che tutte o parti della response sono destinate ad un singolo utente e **non devono essere memorizzate** in una cache condivisa (shared cache). Una cache privata (non-shared) potrebbe memorizzare la response

**no-cache** Indica che tutte o parti della response **non devono essere memorizzate in nessuna cache**

```
general-header = Cache-Control | Connection | Date | Pragma | Transfer-Encoding | Upgrade
| Via
```

Questi header si applicano a **tutto il messaggio**. Esempi:

```
Date: Tue, 15 Nov 1994 08:12:31 GMT
Connection: close
Transfer-Encoding: chunked
```

**Entity header** Relativi all'entità trasmessa

Content-type, Content-length, data di scadenza...

### 12.5.1 Request header

Relativi alla richiesta

Chi fa la richiesta, a chi viene fatta, che tipo di caratteristiche è in grado di accettare il client, autorizzazione... consente al client di passare **informazioni aggiuntive** a proposito della richiesta o del client stesso al server. Questi campi agiscono come **modificatori di richiesta**, con **semantica equivalente a quella dei parametri** di un metodo.

```
request-header = Accept | Accept-Charset | Accept-Encoding | Accept-Language | Authorization  
| Proxy-Authorization | From | Host | If-Modified-Since | If-Unmodified-Since | If-Match |  
If-None-Match | If-Range | Max-Forwards | Range | Referer | User-Agent
```

**Accept** Specifica che tipi di media sono accettabili nella response. Parametro **q** per indicare un fattore di qualità relativo, default a 1.

**Accept-Charset** Indica il set di caratteri accettato per la risposta

**Accept-Encoding** Tipi di trasformazioni accettate (es. compressione)

```
Accept: text/plain;q=0.5, text/html, text/x-dvi;q=0.8, text/x-c  
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8  
Accept-Encoding: compress, gzip
```

(Alcuni) metodi request:

**OPTIONS** Richiede solo le opzioni di comunicazione associate ad un URL o al server stesso (capacità, metodi esposti ecc.). Un esempio:

```
OPTIONS http://192.168.11.66/manual/index.html HTTP/1.1  
host: 192.168.11.66  
Connection: close
```

```
HTTP/1.1 200 OK  
Date: Sun, 14 May 2000 19:52:12 GMT  
Server: Apache/1.3.9 (Unix) (Red Hat/Linux)  
Content-Length: 0  
Allow: GET, HEAD, OPTIONS, TRACE  
Connection: close
```

**GET** Richiede il trasferimento di una risorsa identificata da un URL o le operazioni associate all'URL stessa. Un esempio:

```
GET http://192.168.11.66 HTTP/1.1  
host: 192.168.11.66  
Connection: close
```

Response

```
HTTP/1.1 200 OK  
Date: Sun, 14 May 2000 19:57:13 GMT  
Server: Apache/1.3.9 (Unix) (Red Hat/Linux)  
Last-Modified: Tue, 21 Sep 1999 14:46:36 GMT  
ETag: "f2fc-799-37e79a4c"  
Accept-Ranges: bytes  
Content-Length: 1945  
Connection: close  
Content-Type: text/html
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2  
Final//EN">  
<HTML>...
```

Sono possibili **GET condizionali e parziali**. Esempio di GET condizionale:

```
GET http://192.168.11.66 HTTP/1.1  
Host: 192.168.11.66  
If-Modified-Since: Tue, 21 Sep 1999  
14:46:36 GMT
```

Response:  
HTTP/1.1 304 Not Modified  
Date: Wed, 22 Sep 1999 15:06:36 GMT  
Server: Apache/1.3.9 (Unix) (RedHat/Linux)

**HEAD** Simile al GET, ma il server **non trasferisce il body** nella response. Utile per controllare lo stato dei documenti (validità, modifiche...). Un esempio:

```
HEAD http://192.168.11.66 HTTP/1.1
host: 192.168.11.66
Connection: close
```

Response (notare la somiglianza con GET, esclusa la mancanza qua del body):

```
HTTP/1.1 200 OK
Date: Sun, 14 May 2000 20:02:41 GMT
Server: Apache/1.3.9 (Unix) (Red Hat/Linux)
Last-Modified: Tue, 21 Sep 1999 14:46:36 GMT
ETag: "f2fc-799-37e79a4c"
Accept-Ranges: bytes
Content-Length: 1945
Connection: close
Content-Type: text/html
```

**POST** Serve per **inviare dal client al server** informazioni inserite nel body del messaggio.

In teoria lo standard dice che il metodo POST è usato per richiedere che il server **accetti l'entità racchiusa nella richiesta come nuovo subordinato della risorsa identificata** dallo Request-URI nel Request-Line.

**Nella pratica, la funzionalità effettiva del metodo POST è determinata dal server** e solitamente dipende dalla Request-URI.

**DELETE** Il client **chiede di cancellare una risorsa identificata** dalla Request-URI.  
Solitamente non attivo su server pubblici.

**PUT** Il client **chiede di creare/modificare una risorsa identificata** dalla Request-URI. Dopo posso usare una GET per recuperarla.  
Solitamente non attivo su server pubblici.

**Response header** Nel messaggio di risposta  
Server, autorizzazione richiesta...

**Safe Methods** Metodi che **non hanno effetti collaterali** (es. non modificano la risorsa): GET, HEAD, OPTIONS, TRACE

**Idempotent Methods** Metodi che **non hanno effetti ulteriori se vengono fatti N > 0 richieste identiche**: GET, HEAD, PUT, DELETE, OPTIONS, TRACE

## 12.6 HTTP Response

```
Response = Status-Line *( general-header | response-header | entity-header ) CRLF [message-body]
```

Un esempio:

```
HTTP/1.1 200 OK
Date: Sun, 14 May 2000 23:49:39 GMT
Server: Apache/1.3.9 (Unix) (Red Hat/Linux)
Last-Modified: Tue, 21 Sep 1999 14:46:36 GMT
```

**Status-Line** La prima linea del messaggio di risposta.

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

Esempio: HTTP/1.1 200 OK

**Status-Code** Intero a 3 cifre, risultato del tentativo di comprendere e soddisfare la richiesta.

<b>1xx: Informational</b> - Request received, continuing process
<b>2xx: Success</b> - The action was successfully received, understood, and accepted
<b>3xx: Redirection</b> - Further action must be taken in order to complete the request
<b>4xx: Client Error</b> - The request contains bad syntax or cannot be fulfilled
<b>5xx: Server Error</b> - The server failed to fulfill an apparently valid request

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

**Reason-Phrase** Ha l'obiettivo di fornire una breve descrizione testuale dello Status-Code. Lo Status-Code è indirizzato ai computer mentre la Reason-Phrase è per gli umani.

### 12.6.1 Response Headers

Il campo response-header consente al server di passare ulteriori informazioni sulla response. Questi campi dell'header forniscono informazioni sul server e sull'accesso alla risorsa identificata dallo Request-URI.

response-header = Age | Location | Proxy-Authenticate | Public | Retry-After | Server | Vary | Warning | WWW-Authenticate

Esempio:

```
Age: 150 // età del doc. se tramite Proxy
Location: http://www.w3.org/pub/WWW/People.html
Server: CERN/3.0 libwww/2.17
```

**Age** Una stima in secondi del tempo passato dalla generazione della risposta dal server di origine

**Location** Usato per reindirizzare il ricevente verso una destinazione diversa dalla Request-URI per il completamento della richiesta o l'identificazione di una nuova risorsa

**Server** Informazioni sul software usato dal server d'origine per gestire la richiesta

## 12.7 Negoziazione del contenuto

Le risorse possono essere **disponibili in multiple rappresentazioni**, ad esempio più lingue, formati, dimensioni e risoluzioni, o variare in altri modi ancora. La **content negotiation** è il meccanismo usato per **selezionare l'appropriata rappresentazione** quando si serve una richiesta. Ogni entità è costituita da un **entity body** e da una serie di **entity headers** che ne definiscono il contenuto e le proprietà. Gli entity header sono **informazioni sulle informazioni**, cioè **metadati**.

### 12.7.1 Entity Headers

entity-header = Allow | Content-Base | Content-Encoding | Content-Language | Content-Length | Content-Location | Content-MD5 | Content-Range | Content-Type | ETag | Expires | Last-Modified | extension-header

**Content-Base** URI assoluta da usare per risolvere le URL relative contenute nell'entity-body

**Content-Encoding** Codifica dell'entity-body (es. gzip)

**Content-Language** Lingua dell'entity-body (es. en, it)

**Content-Type** Tipo dell'entity-body (es. text/html)

**Expires** Valore temporale dell'entity-body (utile nel caching)

**Last-Modified** Data dell'ultima modifica sul server (utile nel caching)

## 13 Web Caching

L'obiettivo è **soddisfare una richiesta** del cliente **senza contattare il server**. Si memorizzano copie temporanee delle risorse web (es. pagine HTML e immagini) e si servono al client per ridurre l'uso di risorse (es. banda e workload sul server), diminuendo anche il tempo di risposta.

**User Agent Cache** lo user agent (il browser) mantiene una **copia delle risorse visitate dall'utente**.

**Proxy Cache** Il proxy intercetta il traffico e **mette in cache le risposte**. Le successive richieste alla stessa Request-URI **possono essere servite dal proxy senza inoltrare la richiesta** al server.

**Proxy** Programma intermediario che agisce sia da server che da client, con l'obiettivo di fare richieste per conto di altri client. Le richieste sono servite internamente o passandole oltre, anche traducendole, ad altri server.

## 14 Cookies

L'HTTP è **stateless**, per cui non mantiene info sui client. Come posso riconoscere il cliente di un'applicazione web (es. Amazon)? Come posso realizzare applicazioni web con stato (es. carrello della spesa)? Ricordiamo che **tipicamente l'utente si connette ogni volta con un indirizzo IP e porta diversi**.

**Soluzione:** numerare i client e obbligarli a farsi riconoscere ogni volta presentando un cookie.

**Funzionamento** Il client C invia al server S una normale richiesta HTTP.

Il server invia la normale risposta + una linea **Set-Cookie: 1678453**

Il client memorizza il cookie in un file associato a S, e aggiunge una linea **Cookie: 1678453** a tutte le successive richieste verso quel sito.

Il server confronta il cookie presentato con l'informazione che ha associato a quel cookie.

**Utilizzi** I cookie vengono utilizzati per:

- Autenticazione
- Ricordare il profilo utente e le scelte precedenti (alla carta-socio)
- Creare sessioni sopra un protocollo stateless (es. carrelli della spesa)

**Non accettare dolci dagli sconosciuti:** cookiecentral.com

## 15 Telnet

**T**ERminaL NETwork Protocollo di terminale remoto che permette l'**uso interattivo** di macchine remote: **accesso remoto, accesso multiplo ad un singolo computer**.

Realizza coppie client-server generiche per login remoto, non specializzate per tipo di applicativo. Invece di offrire server specializzati per servizi interattivi, l'approccio consiste nel permettere all'utente di **effettuare una sessione login nella macchina remota** e quindi **inviare i comandi**. Tramite il login remoto gli utenti hanno accesso ai comandi e ai programmi disponibili nella macchina remota.

*Puoi mandare comandi attraverso il programma Telnet e verranno eseguiti come se fossi a scriverli direttamente sulla server console. Questo ti consente di controllare il server e comunicare con gli altri server sulla rete*  
**Non è un compito facile**, per realizzarlo il Telnet:

Maschera sia la rete che i S.O.

Utilizza un'**interfaccia minima ma veloce**, tipicamente a caratteri

### 15.1 Introduzione

Telnet permette ad un utente su una macchina di **stabilire una connessione con un login su server remoto**. In seguito, passa la **battute dei tasti** della macchina locale **alla macchina remota**: i comandi vengono **eseguiti come se fossero stati battuti al terminale della macchina remota**.

Dopodiché, l'**output** della macchina remota **viene trasportato al terminale utente**.

Questo è un **servizio trasparente**: il terminale dell'utente *sembra* essere **connesso direttamente** alla macchina remota

Il modello di Telnet include:

**Server** che **accetta** le richieste

**Client** che **effettua** le richieste

**Il programma Telnet** che svolge due funzioni:

**Interagisce col terminale utente** sull'host locale

**Scambia messaggi** con il Telnet server

### 15.2 Protocollo Telnet

**RFC 854** Comunicazione generale, bidirezionale e orientata a blocchi di 8bit. L'obiettivo primario è di fornire un metodo standard per **interfacciare dispositivi terminali e processi terminal-oriented tra loro**.

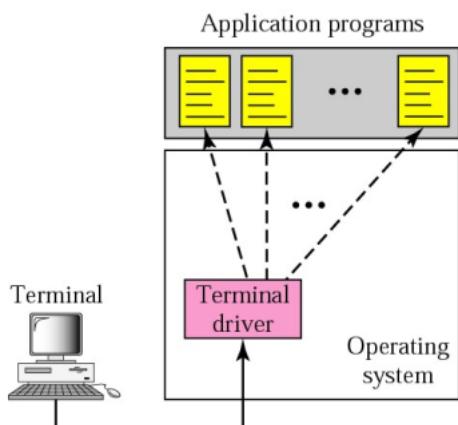
Usa il TCP, con una connessione TCP persistente per tutta la durata della sessione di login, sulla porta 23 del server.

Client **stabilisce una connessione TCP** con il server

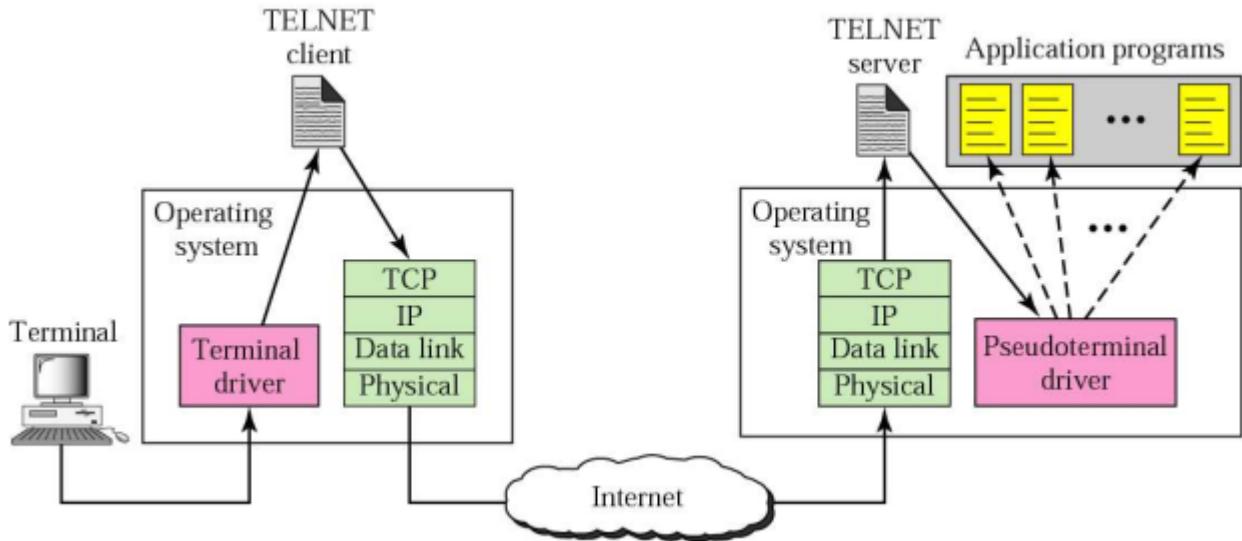
Client **accetta le battute di tasti** sul terminale e **le invia al server**.

**Accetta i caratteri** che il server manda indietro e **li visualizza** sul terminale utente.

Server **accetta la connessione TCP** e trasmette i dati al S.O. locale.



In **Local Login** il S.O. assume che gli input ad un processo vengano forniti dallo standard input (tastiera) e che gli output siano inviati allo standard output (monitor).



In **Remote Login** lo **Pseudo Terminal Driver** è l'entry point del S.O. che consente di trasferire caratteri ad un processo **come se provenissero dal terminale**. Ha il compito di **accettare i caratteri** dal server e **trasmetteri al S.O.** che consegnerà all'applicazione opportuna.

### 15.3 NVT

Il Telnet deve **poder operare con il numero massimo di sistemi**, quindi gestire **dettagli di S.O. eterogenei** che possono differire per:

**Set di codifica** dei caratteri

**Lunghezza** della linea e della pagina

**Tasti funzione** individuati da diverse sequenze di caratteri (**escape sequence**)

Es. diverse combinazioni per interrompere un processo(CTR+C, ESC), caratteri ASCII diversi per la terminazione di righe di testo.

Per risolvere questo problema si definisce **un ambiente virtuale**. Sulla rete si considera un **unico terminale standard** e in corrispondenza di ogni stazione di lavoro si effettuano le **conversioni da terminale locale a terminale virtuale e viceversa**.

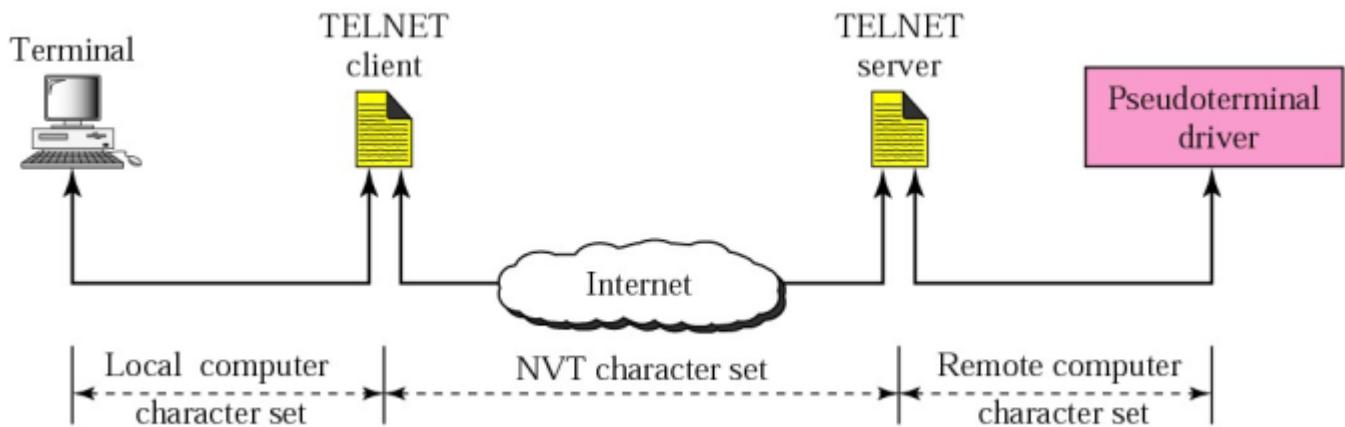
**Network Virtual Terminal** Telnet assume che sui due host sia in esecuzione un **Network Virtual Terminal**, la connessione TCP è stabilita tra i due terminali NVT.

L'NVT è un dispositivo *immaginario* che **fornisce una rappresentazione astratta di un terminale canonico**. Gli host, sia client che server, traducono le loro caratteristiche locali così da **apparire esternamente come un NVT** e assumo che l'host remoto sia un NVT.

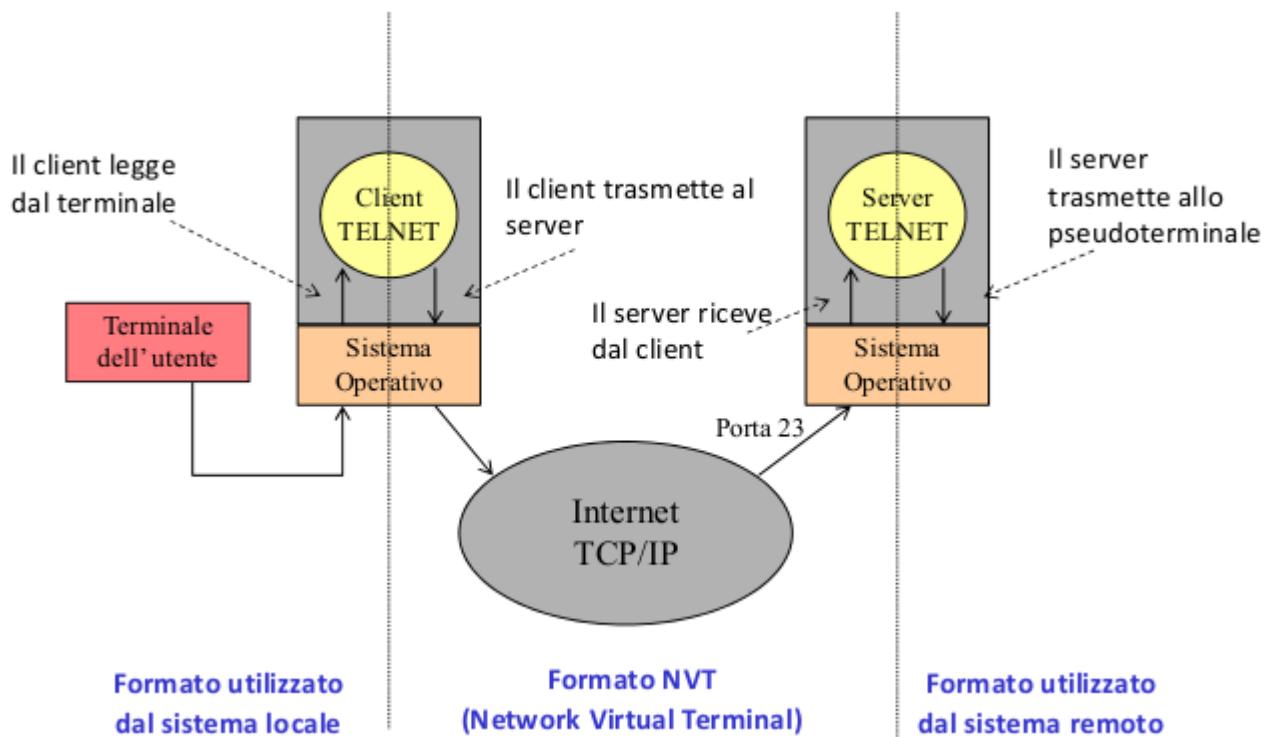
Il Network Virtual Terminal definisce un **set di caratteri e di comandi universale**, che permette di:

**Trasformare il set di caratteri in uso** localmente **in un set di caratteri universale** (lettere accentate, tasti freccia, backspace...), includendo anche i caratteri di controllo più importanti (break...)

**Inviare i caratteri di controllo** in maniera **privilegiata** (meccanismo URGENT del TCP)



#### 15.4 Architettura



## 15.5 Funzionamento

Rispetto alla esecuzione in locale, sono aggiunti una **serie di intermediari**:

- Il client Telnet **trasforma in NVT** ed usa il S.O. per inviare sulla rete;
- la rete ed il S.O. server portano i dati al server Telnet;
- il server Telnet **traduce da NVT a S.O.** remoto;
- il S.O. remoto **esegue** il dovuto;
- si percorre il cammino inverso con gli stessi soggetti.

Un esempio:

1. L'utente digita INVIO sulla tastiera
2. Il S.O. passa al client il carattere CR
3. Il client converte CR in CR-LF e lo invia sulla connessione TCP
4. Il server riceve CR-LF, lo converte in LF e lo passa allo pseudoterminale
5. Lo pseudoterminale passa all'applicazione LF
6. L'applicazione esegue l'operazione di INVIO

**Formato** I terminali NVT si scambiano i dati in **formato 7bit US-ASCII**. Ogni carattere è inviato come un ottetto con il primo bit settato a 0. I byte con il bit più significativo a 1 sono usati per le **sequenze di comandi**. I comandi (es. end-of-line trasmesso come la sequenza CR-LF) iniziano con un ottetto speciale (**Interpret as Command** o **IAC**) di 1 → **Inband Signalling** (comandi e dati sulla stessa connessione).

I messaggi di controllo iniziali sono usati per scambiare informazioni sulle caratteristiche degli host (**Telnet Option Negotiation**).

**Comandi** Qualche esempio

Comando	Codifica Decimale	Significato
IAC	255	Interpreta come comando l'ottetto successivo
EL	248	Erase Line
EC	247	Erase Character
IP	244	Interrupt Process
EOR	239	End of Record

**NVT conviene?** Suppongo di voler far interoperare N sistemi.

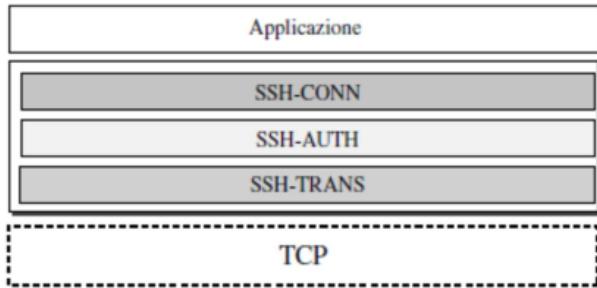
Senza usare NVT, ho bisogno di scrivere N-1 client per ogni sistema, e N server (uno per sistema) = **devo scrivere N(N-1)+N applicazioni**.

Usando NVT invece devo scrivere soltanto N server e N client, cioè **2N applicativi**.  
Per N > 2 conviene NVT.

## 16 SSH

Poiché Telnet passa tutto in chiaro, **anche le password**, con il tempo si è resa necessaria una maggiore sicurezza.

**Secure Shell** SSH è un'applicazione nata per **sostituire Telnet e risolvere i suoi problemi di sicurezza**. Facilita la comunicazione sicura tra client e server e permette la login remota, resa sicura attraverso **tecniche di cifratura**. Nella realtà SSH offre **funzionalità molto superiori** a quelle di Telnet<sup>2</sup>.



**SSH-TRANSPORT** Realizza la **connessione sicura** tra due host:

Autenticazione del server

Negoziazione degli algoritmi di cifratura

Scambio di chiavi

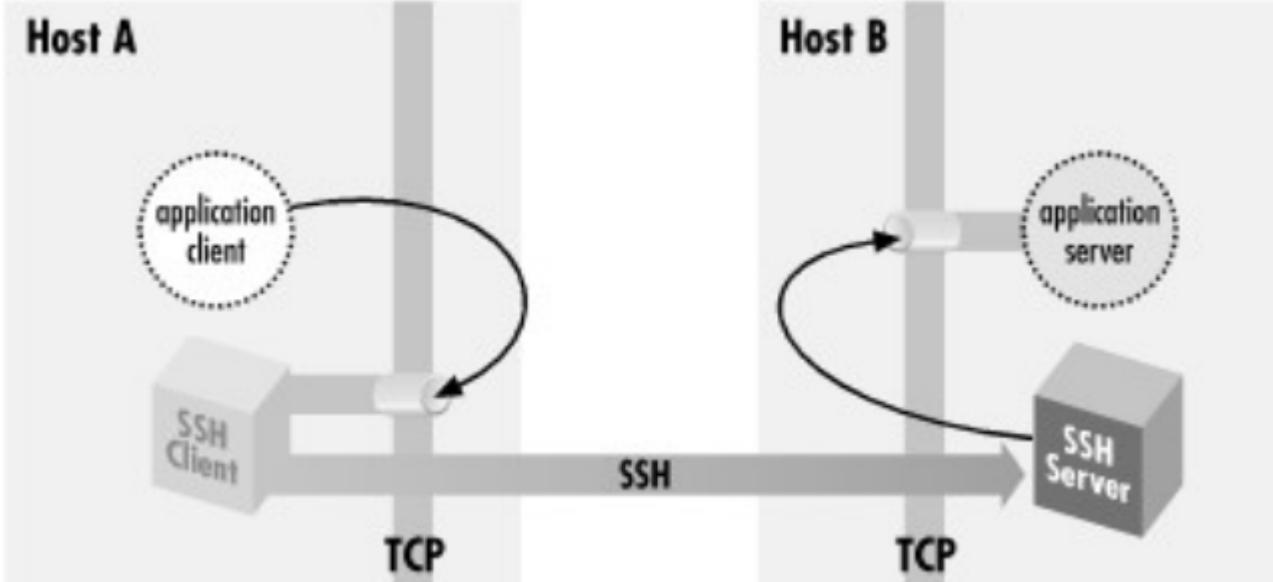
**SSH-AUTHENTICATION** Meccanismi per autenticare l'utente

**SSH-CONNECTION** Sessioni di login remoto, tunneling

## 17 TCP Port Forwarding

**Port Forwarding** L'inoltro delle porte è un meccanismo che permette di creare un **canale di comunicazione sicuro** attraverso il quale veicolare qualsiasi tipo di connessione TCP.

Viene **creato un canale di comunicazione cifrato tra la porta all'indirizzo remoto** al quale ci si vuole collegare **ed una porta locale libera**. In questo modo le applicazioni punteranno il collegamento alla porta locale e la connessione verrà **inoltrata automaticamente** all'host remoto tramite un canale sicuro.



`ssh -L 123:localhost:456 remotehost`

-L specifica che la data porta sull'host locale è da inoltrare alla data porta sull'host remoto.

`<porta locale>:<host>:<porta remota di host>`

<sup>2</sup>Vedi su Forouzan

# 18 FTP

Il **File Transfer Protocol** (RFC 959) è usato per il **trasferimento di file da/a un host remoto**. Segue il modello client server:

Client: il lato che **chiede** il trasferimento

Server: l'host remoto

**Standard** Nelle reti TCP/IP lo FTP è lo **standard per il trasferimento di file**. Questo è un servizio **diverso** dall'accesso condiviso online (che è un **accesso simultaneo** da parte di più programmi **ad un singolo file**). L'FTP fornisce anche **funzionalità aggiuntive** oltre al semplice trasferimento di file:

**Accesso interattivo:** l'utente può **navigare e cambiare/modificare** l'albero di directory nel file system remoto

**Specifiche del formato** dei dati da trasferire (es. file di testo o file binari)

**Autenticazione:** il client può specificare username e password

## 18.1 Modello FTP

L'FTP ha **due tipi di connessione**:

**Control connection:** scambio di comandi e risposte tra client e server, segue il protocollo Telnet

**Data connection:** connessione su cui i dati sono trasferiti con modi e tipi specificati. I dati trasferiti possono essere **parte** di file, **un file** o **un set** di file.

### 18.1.1 Connessione di Controllo

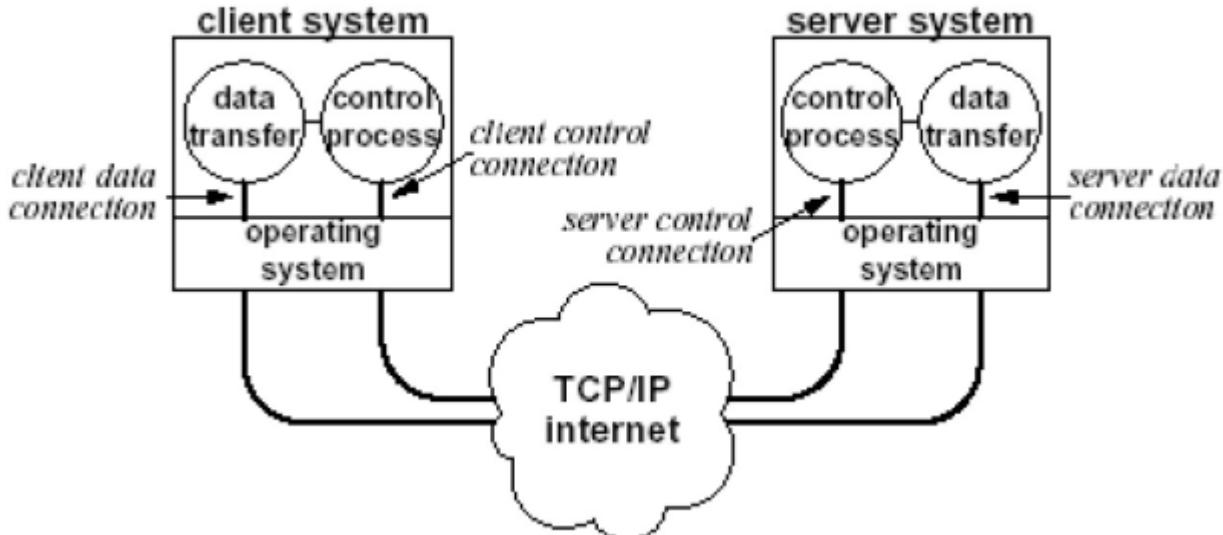
Il client FTP **contatta il server FTP alla porta 21** usando il TCP come protocollo di trasporto. Il client **ottiene l'autorizzazione** sulla connessione di controllo (es. cambio directory, invio file ecc...).

La **connessione è persistente**.

### 18.1.2 Connessione Dati

Quando il server riceve un comando per trasferire file (da o verso il client), **apre una connessione TCP** con il client.

**Active Mode:** una connessione per ciascun trasferimento e dopo il trasferimento di un file il server chiude la connessione.



## 18.2 Altri Dettagli

Quando un client attiva la connessione di controllo con il server usa un **numero di porta assegnato localmente in modo casuale** e contatta il server ad una porta nota, cioè la 21

FTP usa la connessione di controllo per permettere a client e server di **coordinare l'uso delle porte assegnate dinamicamente** per il trasferimento dati

La connessione di controllo FTP si basa sul protocollo Telnet

### 18.2.1 Due Modalità

Per creare la connessione TCP per il trasferimento dati sono possibili due modalità:

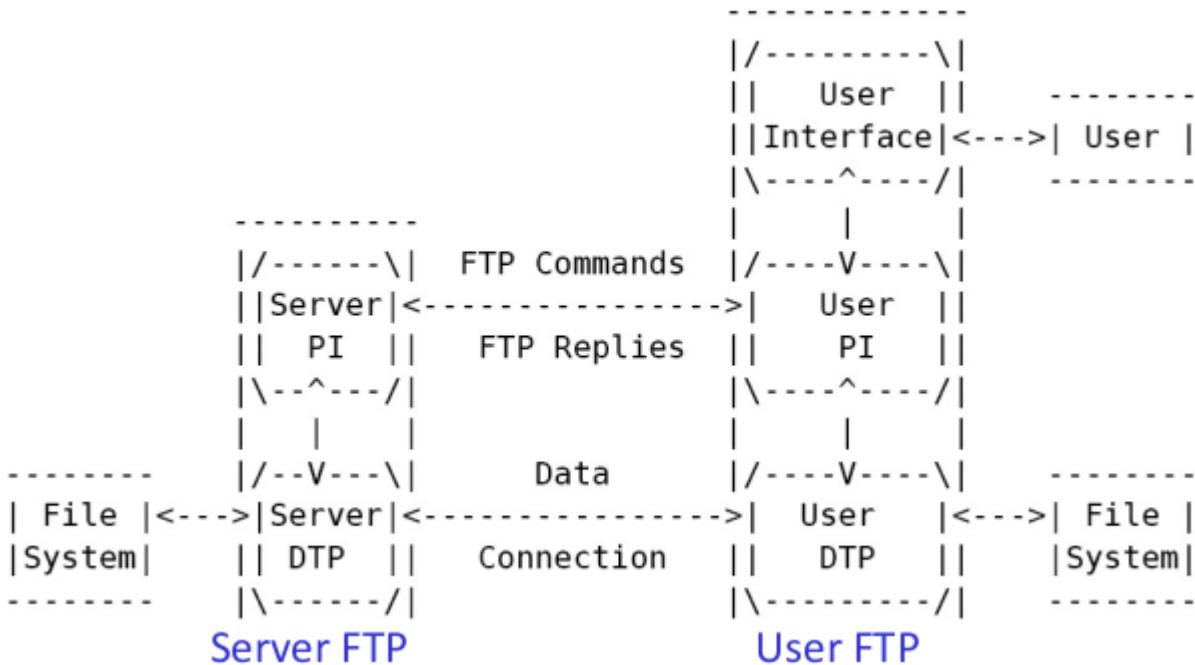
**Active Mode:** vedi sopra, una connessione per ciascun trasferimento e dopo il trasferimento di un file il server chiude la connessione. Il server deve conoscere il numero di porta lato client (glielo comunica)

**Passive Mode:** il **client ottiene un numero di porta dal server** (porta 20, non necessariamente). Il server non deve accettare connessioni da un processo arbitrario.

Da RFC 959 – PASV: questo comando richiede che il server-DTP "ascolti" su una porta dati (che non è quella predefinita) e di aspettare una connessione piuttosto che iniziare una alla ricezione del comando di trasferimento. La risposta a questo comando include l'host e l'indirizzo e porta su cui il server sta ascoltando.

**DTP:** Data Transfer Process

**PI:** Protocol Interpreter



I dispositivi dove risiedono client e server FTP sono diversi: S.O., diverse strutture per gestire i file, diversi formati dei file...

Per effettuare il trasferimento di file, il client deve **definire il tipo di file, la struttura dati e la modalità di trasmissione** al fine di risolvere i problemi di eterogeneità tra client e server.

Il trasferimento file viene **preparato attraverso uno scambio di informazioni lungo la connessione di controllo**

La comunicazione sulla connessione di controllo avviene per mezzo di caratteri con una codifica standard NVT ASCII, sia per i comandi che per le risposte

### 18.2.2 Stateful

FTP è un protocollo stateful. Il server deve **tenere traccia dello stato** dell'utente: tra le altre cose anche della connessione di controllo associata ad un account e della directory attuale.

### 18.2.3 Modalità di trasmissione

**Stream mode:** FTP invia i dati a TCP con un **flusso continuo di bit**

**Block mode:** FTP invia i dati a TCP **suddivisi in blocchi**, ognuno dei quali **preceduto da un header**

**Compressed mode:** si trasmette il **file compresso**

## Comandi di Controllo

USER username

PASS password

LIST, elenca i file della directory corrente

NLST, richiede l'elenco di file e directory (ls)

RETR filename, recupera (get) un file dalla directory corrente

STOR filename, memorizza (put) un file nell'host remoto

ABOR, interrompe l'ultimo comando ed i trasferimenti in corso

PORT, indirizzo e numero di porta del client

SYST, il server restituisce il tipo di sistema

QUIT, chiude la connessione

## Codici di Ritorno

I codici di ritorno sono composti da un codice di stato e da un'espressione, come in HTTP:

331 Username OK, password required

425 Can't open data connection

452 Error writing file

200 Comando OK

125 Data connection already open, transfer starting

225 Data connection open

225 Closing data connection. Requested file action succesful (es. trasferimento file o ABOR)

426 Connection closed, transfer aborted

227 Entering passive mode

## 18.3 Anonymous FTP

Server che supportano connessioni FTP senza autenticazione, spesso consentendo operazioni limitate.  
ftp.ed.ac.uk, user: ftp e password la mail.

## 19 DNS

**Identificare il processo** Ogni processo di livello applicativo ha necessità di **individuare il processo omologo** con il quale vuole comunicare. Il processo omologo **risiede su una** particolare **macchina remota** anch'essa da **individuare** (sappiamo che usa lo stesso protocollo).

**Nome** Uno **nome identifica un oggetto**. Consiste in una sequenza di caratteri scelti da un alfabeto finito, scelta per scopo mnemonico.

Nome significativo di alto livello (alfanumerico), vero e proprio identificativo di livello applicativo.

**Indirizzo** Un **indirizzo identifica dove tale oggetto è situato**.

Host di internet usano indirizzi IP (32bit) per instradare i datagrammi (livello di rete). Il formato è scelto per garantire l'efficienza dell'instradamento.

**Disaccoppiamento:** ad un nome possono essere associati più indirizzi.

Come associare indirizzo IP e nome?

### 19.1 Motivazioni

**Inizi** All'inizio l'associazione fra nomi logici e indirizzi IP era **statica**: tutti i nomi logici ed i relativi indirizzi erano contenuti in un file (**host file**) periodicamente aggiornato da un server ufficiale. Questo approccio è infattibile nella rete internet attuale:

Non è possibile che **ogni host mantenga una copia aggiornata** dell'elenco. Questo per dimensioni dell'elenco, per i volumi di traffico per trasferire i cambiamenti...

Non è possibile **centralizzare un elenco del genere**: si avrebbe così un unico punto di fallimento, un gigantesco volume di traffico sul server...

Questa soluzione non è scalabile.

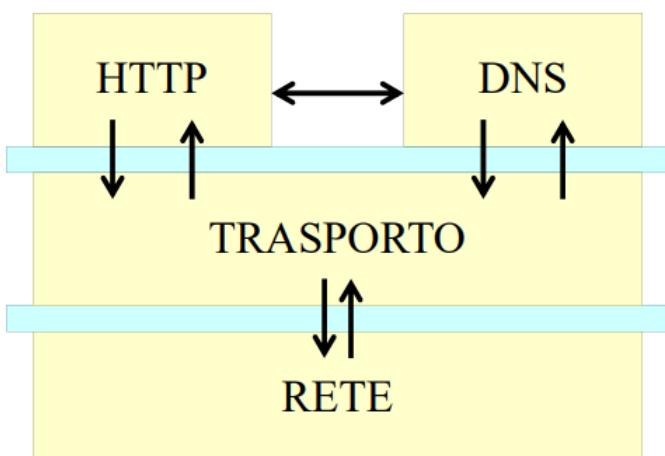
Per questo si utilizza il **Domain Name System**

### 19.2 Struttura

Il DNS è **posizionato nel livello applicativo**: gira sui terminali a paradigma client-server. Si affida al sottostante protocollo di trasporto punto-punto per trasferire i messaggi tra i terminali.

**Non interagisce direttamente con utenti**

**La complessità è spostata alle estremità della rete**



Il DNS è un meccanismo che deve:

**Specificare la sintassi** dei nomi e le regole per gestirli

**Consentire la conversione** nomi → indirizzi e viceversa

Il DNS è costituito essenzialmente da:

**Schema di assegnazione dei nomi**, gerarchico e basato su domini

**Database distribuito** contenente i **nomi** e le **corrispondenze** con gli indirizzi

**Protocollo per la distribuzione** delle informazioni sui nomi **tra i vari name server**

### 19.3 Servizi

I servizi offerti dal DNS sono molteplici:

**Risoluzione** di nomi di alto livello (**hostname**) in indirizzi IP

**Host aliasing**: un host può avere più nomi, solitamente il nome canonico + sinonimi.

Esempio: `realy1.west-coast.enterprise` può avere due alias `enterprise.com` e `www.enterprise.com`.

**Mail Server aliasing**: ci possono essere domain name identici per mail server e web server

**Distribuzione del carico** tra vari server replicati.

Ad un hostname canonico possono corrispondere diversi indirizzi IP. La lista di indirizzi viene ordinata in modo diverso in ogni risposta alla richiesta di risoluzione del nome, così che ogni server replicato possa essere scelto con uguale probabilità, distribuendo così efficientemente le richieste.

### 19.4 Spazio dei nomi

Dato che internet è una rete di proporzioni enormi, si presentano i problemi di **identificazione delle macchine** e **intradamento dei pacchetti**. Si adotta un **approccio stratificato** poiché è un sistema complesso.

#### 19.4.1 Indirizzi

In internet esistono più indirizzi

Indirizzi **MAC**, quello della scheda di rete.

Soltanente prefissato

**Indirizzo di rete (IP)**

Esempio: `150.217.8.21`

Assegnato dal gestore di rete in base al tipo di rete a cui si appartiene (classe di sottorete)

**Indirizzo di trasporto**

Coppia <indirizzo IP, porta>

La porta è scelta a livello applicativo con regole appropriate

**Indirizzo alfanumerico**

Esempio: `medialab.det.unifi.it`

Libero, basta che sia mappato in un NameServer

#### 19.4.2 Nomi

Lo **spazio dei nomi** deve permettere di **identificare in modo univoco** un host.

**Struttura flat**: sequenza di caratteri senza alcuna ulteriore struttura. Poco applicabile.

**Struttura gerarchica**: un nome è **costituito da diverse parti**.

Requisiti per la partizione dello spazio dei nomi:

**Conversione efficiente**

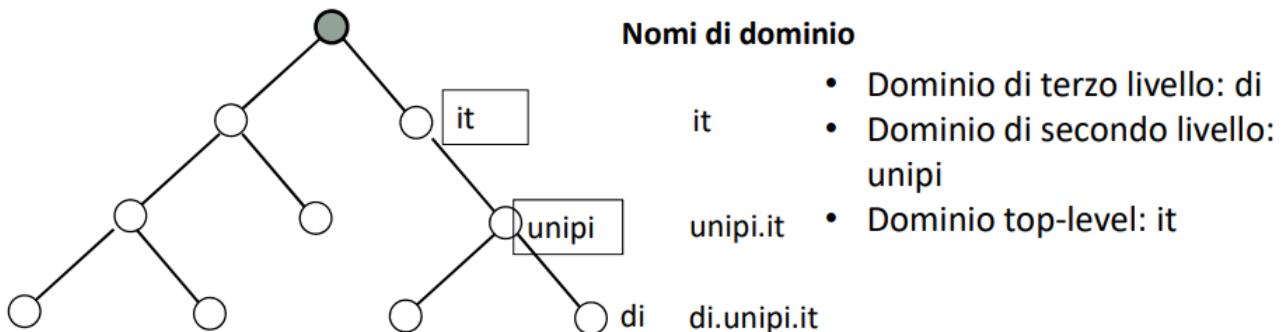
**Controllo decentralizzato** dell'assegnazione dei nomi.

Delega dell'autorità per le varie parti dello spazio dei nomi e **distribuzione della responsabilità** della conversione tra nomi e indirizzi

Vantaggi e svantaggi

- Minore velocità
- + Maggiore flessibilità
- + Riconfigurazione più veloce
- + Possibilità di aggiornamenti decentrati

**Nomi di dominio** Spazio dei nomi con **struttura gerarchica**. I nomi hanno una **struttura ad albero** con un numero di livelli variabile. **Ogni nodo definisce un livello gerarchico** ed è individuato da un'etichetta (max 63 caratteri). Alla radice è associata un'etichetta vuota.



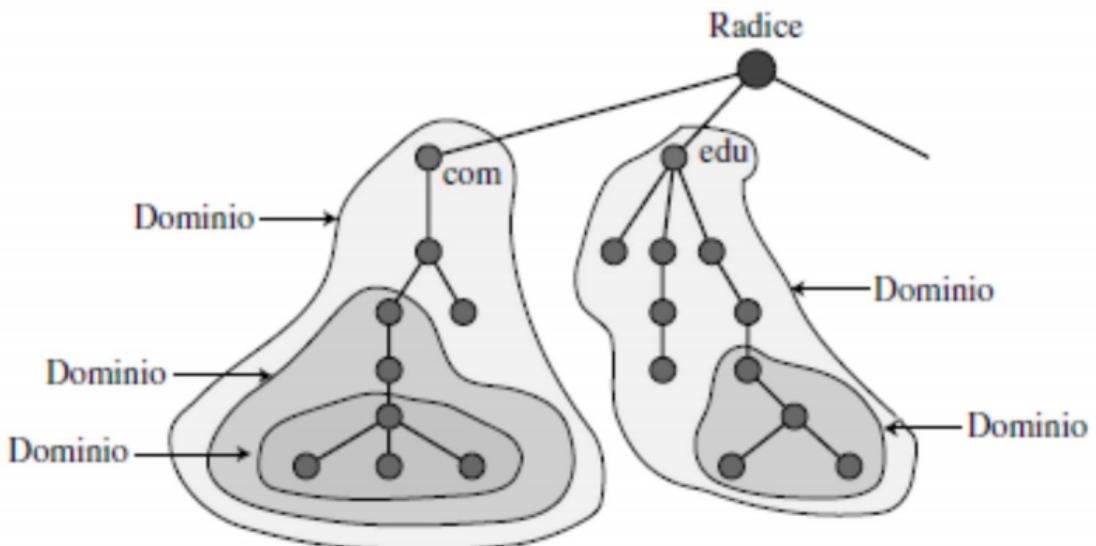
Ogni nodo dell'albero ha un nome di dominio, ovvero una **sequenza di etichette separate da punti** ovvero il cammino foglia → radice.

**Dominio** Sottoalbero nello spazio di nomi di dominio che viene identificato dal nome di dominio del nodo in cima al sottoalbero.

Può essere suddiviso in ulteriori domini, detti **sottodomini**.

**In Internet** Su internet i nomi gerarchici delle macchine sono **assegnati in base alla struttura delle organizzazioni** che ottengono l'autorità per porzioni dello spazio dei nomi. La struttura gerarchica permette **autonomia nella scelta dei nomi all'interno di un dominio** perché l'univocità è comunque garantita.  
Ad es. server1.di.unipi.it e server1.cs.cornell.edu sono due nomi diversi.

Internet è divisa in diverse centinaia di domini ognuno dei quali **partizionato in sottodomini** e così via.



#### 19.4.3 Top-Level Domains

Internet Assigned Number Authority IANA – [iana.org](http://iana.org)

com Organizzazioni commerciali

edu Istituti d'istruzione (università, scuole...)

mil Gruppi militari

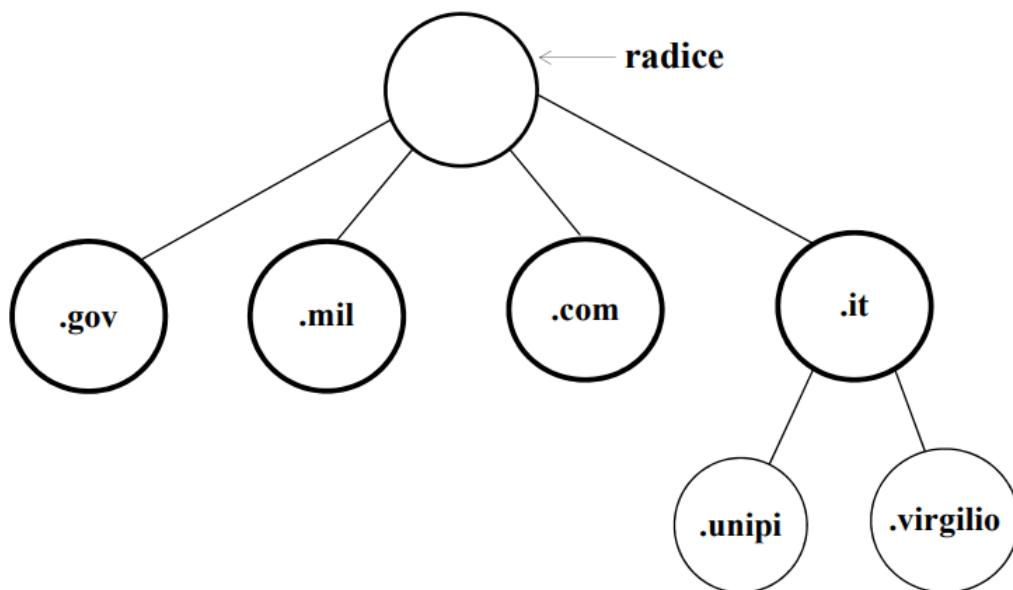
gov Istituzioni governative

net Principali centri di supporto alla rete

org Organizzazioni diverse dalle precedenti

Codici geografici, schema geografico per nazioni

Es. ir, uk, us, fr...



#### 19.4.4 Struttura di un nome alfanumerico

mmedia5.di.unipi.it

mmedia5 nome locale della macchina (etichetta **più specifica**)

di.unipi.it nome del dominio (.it è l'etichetta **meno specifica**)

La parte dominio **può essere ulteriormente suddivisa**, creando così una **struttura logica gerarchica** (di nome locale, unipi.it dominio e così via...)

## 19.5 Conversione

**Indirizzi IP** Gli indirizzi IP sono **interi a 32bit**. Vengono rappresentati nella **Decimal Dotted Notation** che divide l'indirizzo IP in 4 **ottetti**, ovvero 4 gruppi di 8bit. Ad es. 150.217.8.21.

**Gerarchia** L'indirizzo IP è strutturato in una gerarchia a due soli livelli.

**Indirizzi alfanumerici** Gli indirizzi alfanumerici sono del tipo "medialab.di.unipi.it", sono **logici**, gerarchici e **NON indicano in assoluto la locazione geografica** di un host.

Vengono convertiti da un **Domain Name Server** in un **indirizzo IP**, eventualmente usando un sistema ricorsivo di ricerca.

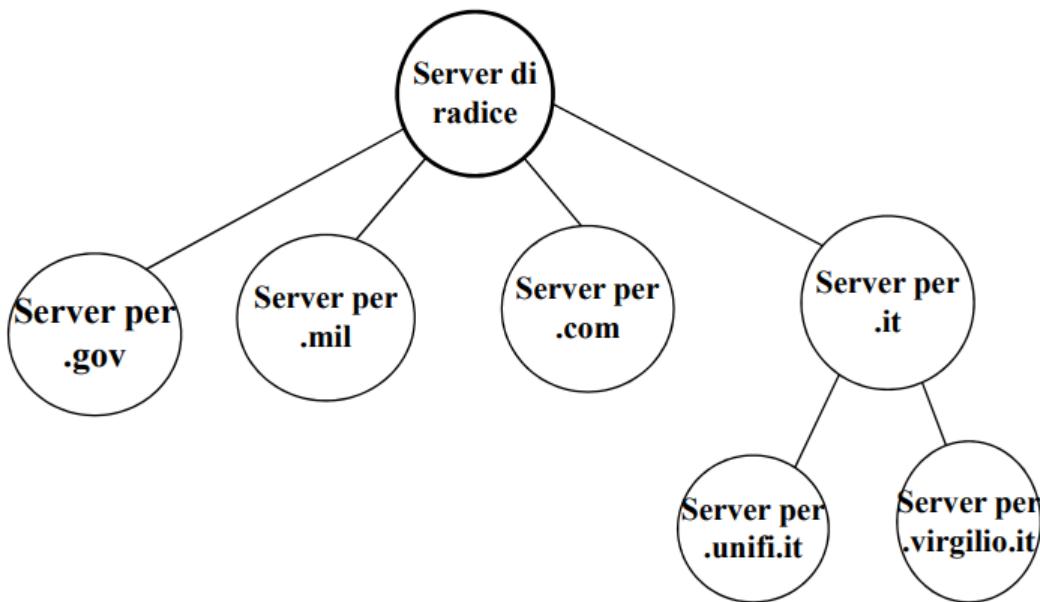
**Database DNS** Il database DNS è **distribuito** ed implementato in una gerarchia di più name servers.

**Protocollo** Il protocollo per la **risoluzione dei nomi alfanumerici** in indirizzi IP è dello **strato applicativo**.

**NB** Una funzione fondamentale di internet è implementata come protocollo dello strato applicazione → una parte della complessità della rete è **gestita alle estremità** della rete stessa.

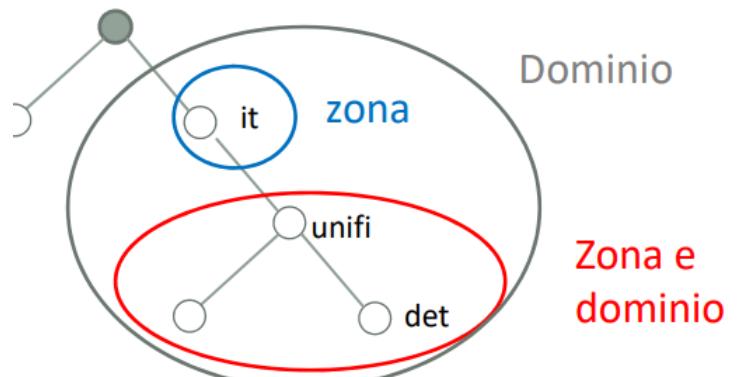
### 19.5.1 Name Servers

Un **Name Server** è un **programma che gestisce la conversione da nome di dominio ad indirizzo IP**. I Name Server sono strutturati gerarchicamente.



Spesso i DNS sono **accorpati e duplicati (sicurezza)**. Inoltre per diminuire il traffico di rete ed il carico dei DNS, ogni server usa **una cache per le ultime richieste espletate**.

**Informazioni** Le informazioni sui domini sono ripartite su più server



**Zona** Una zona è una **regione di cui è responsabile un name server**, tipicamente una parte contigua dell'albero. Zona e dominio non necessariamente coincidono.

Il server **immagazzina le informazioni relative alla propria zona**, inclusi i riferimenti ai server dei domini di livello inferiore.

### 19.5.2 Root Name Servers

I **root name servers** vengono contattati dai **local name servers** che non possono risolvere un determinato nome.  
Il root name server:

Contatta gli **authoritative name server** se quella traduzione non è nota

**Recupera la traduzione**

**Invia la traduzione** al local name server

Ce ne sono centinaia in tutto il mondo.



[root-servers.org](http://root-servers.org)

### 19.5.3 Gerarchia dei server

**Root Name Server**, server radice

**Top-Level Domain Server**: si occupano dei domini .com, .org, .edu...  
Ad es. Network Solution gestisce i TLD server per il dominio .com

**Authoritative Name Server**, server di competenza

Per un host archivia l'indirizzo IP di quello stesso host. Capace di risolvere la traduzione name/address per quell'hostname.

Ogni organizzazione dotata di host internet pubblicamente accessibili (es. web server, server di posta) deve **fornire i record DNS di pubblico dominio** che mappano i nomi di tali host in indirizzi IP (server mantenuti dall'organizzazione o ISP).

Per una certa zona ci possono essere **server primari** e **secondari**.

**Server primari** mantengono il file di zona

**Server secondari** ricevono il file di zona e offrono il servizio di traduzione

**Local Name Server**

Non appartengono strettamente alla gerarchia dei server. Ogni ISP (Università, Società, ISP) ha il **proprio (default) name server locale**. Le query DNS passano prima dal name server locale.

## 19.6 Risoluzione dei nomi

Esempio: il client vuole l'IP di `www.amazon.com`. Prima approssimazione:

Client **interroga il server radice** per trovare il server DNS `com`

Client **interroga il server DNS `com`** per ottenere il server DNS `amazon.com`

Client **interraga il server DNS `amazon.com`** per ottenere l'indirizzo IP di `www.amazon.com`

### Query Ricorsiva

Host `www.tintin.fr` cerca l'IP di `www.topolino.it`.

Contatta il suo DNS locale `dns.tintin.fr`

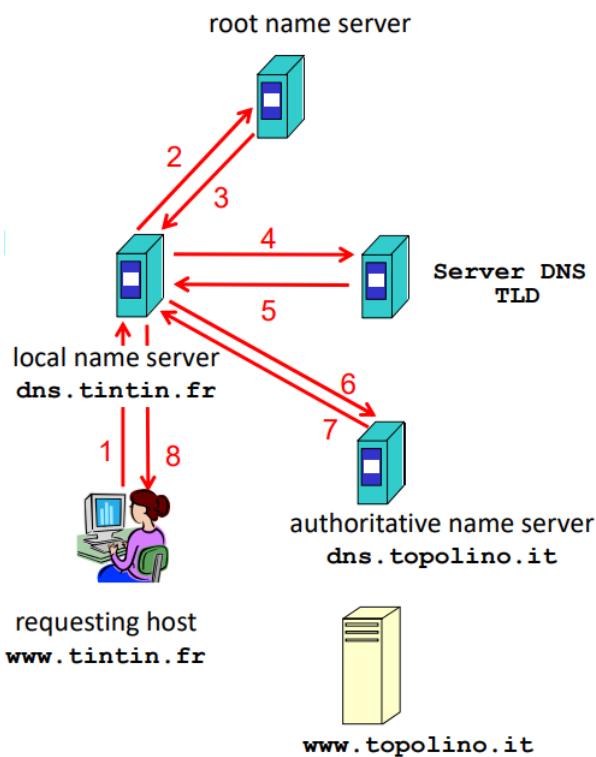
`dns.tintin.fr` contatta il root name server se necessario

Il root name server contatta l'autoritative name server `dns.topolino.it` se è necessario

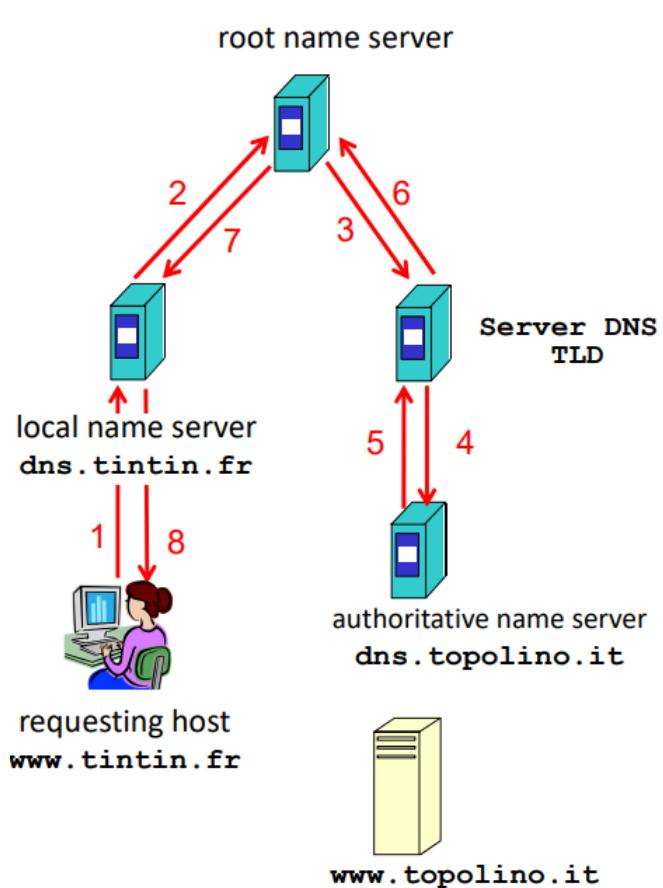
Il local name server ha richiesto una conversione completa

**Query Iterativa** Le risposte vengono restituite direttamente al client, cioè i name server rispondono con il riferimento al name server da contattare.

### Query Iterativa



### Query Ricorsiva



## 19.7 Caching e Aggiornamento Record

Una volta che un name server ha appreso un'associazione, la mette nella **cache**. I record nella cache vengono cancellati dopo un certo tempo (timeout, TTL). L'utilizzo della cache **migliora il ritardo e riduce il numero di messaggi DNS**.

I meccanismi di update/notifica sono descritti nella RFC 2136.

## 19.8 Record DNS

Il DNS è un database distribuito di "resource records" (RR).

Formato RR: (name, value, type, TTL)

**TTL**: quando la risorsa dovrà essere rimossa dalla cache

I significati di **name** e **value** dipendono da **type**

```
type = A  
name = hostname  
value = indirizzo IP  
  
type = CNAME  
name = hostname (sinonimo)  
value = nome canonico dell'host  
  
type = NS  
name = dominio (es. unifi.it)  
value = hostname dell'autoritative name server per quel dominio  
  
type = MX  
name = nome di dominio  
value = nome canonico del server di posta associato a name  
  
Ci sono altri type...
```

Esempi di record DNS

```
(hostname, indirizzoip, A, ...)  
(www.cnn.com, 157.166.224.25, A, ...)  
(www.cnn.com, 157.166.224.26, A, ...)  
  
(dominio, nomeDiAuthoritativeServerPerDominio, NS, ...)  
(cli.di.unipi.it, nameserver.cli.di.unipi.it*, NS, ...)  
[* ANSWER conterrà anche (nameserver.cli.di.unipi.it, 131.114.120.2,A, ...)]  
  
(alias, hostnameMailServerConTaleAlias, MX, ...)  
(cli.di.unipi.it, mailserver.cli.di.unipi.it*, MX, ...)  
[* ANSWER conterrà anche (mailserver.cli.di.unipi.it, 131.114.11.39, A, ...)]
```

## 19.9 Messaggi DNS

Query e risposte hanno lo stesso formato.

Le query contengono **QName** e **QType** nella sezione "question".

Le risposte contengono 0 o più RR nella sezione "answer" e 0 o più RR nella sezione "additional".

Per esempio:

```
dns.poly.edu → TLD server  
Question: QName=gaia.cs.mass.edu QType=A  
  
dns.poly.edu ← TLD server  
Answer: (umass.edu, dns.umass.edu, NS)  
Additional: (dns.umass.edu, 128.115.40.41, A)
```

## Header dei messaggi DNS

**Identification:** 16 bit. Identification usato dalle query, la reply ad una query usa lo stesso identification.

### Flags

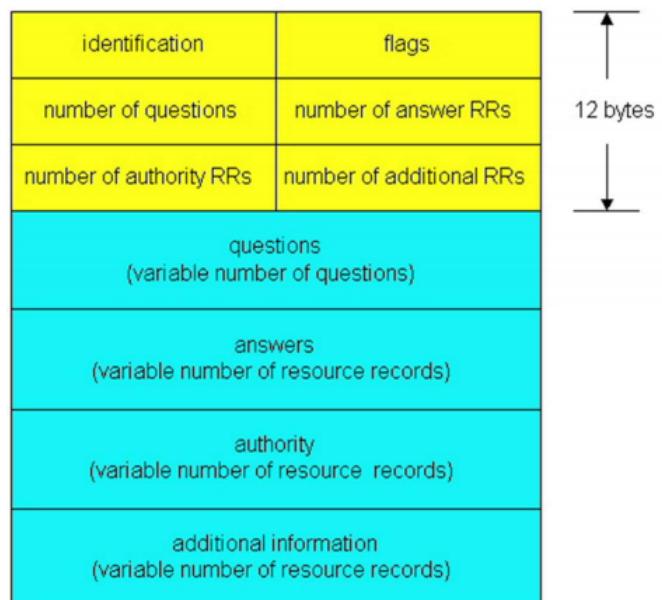
Query (0) o reply (1)

Recursion desired

Recursion available

Reply is authoritative

Il protocollo DNS usa tipicamente **UDP sulla porta 53 per le query DNS** (dimensioni del messaggio inferiori a 512B). Possibile usare TCP, ad esempio per trasferire i dati tra DNS server – Zone Transfers)



**Domande:** campi per il nome richiesto ed il tipo di domanda

**Risposte:** RR nella risposta della domanda

**Competenza:** record per i server di competenza

Informazioni aggiuntive

### Questions

0	31
Nome di dominio dell'interrogazione	
Tipo di interrogazione	Classe di interrogazione
...	

**Tipo di Interrogazione:** tipo di domanda (nome di una macchina o indirizzo di posta)

Il **client** riempie la sezione di **domanda**

Il **server**, nel proprio messaggio di **risposta**:

ricopia la sezione di **domanda**

riempie le altre sezioni (risposta, autorità, altre info) con RR

Le **RFC di riferimento** sono:

RFC 1034: DOMAIN NAMES – CONCEPTS AND FACILITIES  
Definisce i seguenti tipi di record:

**A** indirizzo dell'host

**CNAME** nome canonico

**HINFO** CPU e S.O. usato dall'host

**MX** mail exchange

**NS** Authoritative Name Server

**PTR** name space pointer

**SOA** Inizio di una zona d'autorità

RFC 1035: DOMAIN NAMES – IMPLEMENTATION AND SPECIFICATION

## 20 SMTP

**Posta Elettronica** Uno dei primi servizi nati su internet è la posta elettronica (1982). Essa consiste nel **trasferimento di un messaggio tra** uno user **mittente** e uno user **destinatario**. Il **destinatario potrebbe non essere disponibile** in quel momento, e quindi non poter accettare i messaggi: utente impegnato, **computer spento**.

Il servizio di posta elettronica si **basa su componenti intermediari** per trasferire i messaggi: **disaccoppiamento** lato mittente e destinatario, analogia con posta tradizionale.

Il trasferimento di messaggi di posta elettronica si basa sul **Simple Mail Transfer Protocol** (RFC 821, RFC 2821, RFC 5321).

### Componenti principali

**Agenti utente**

**Mail Server**

**Protocolli**

#### 20.1 Agenti Utente

Detti anche **mail reader**, usati per la **composizione**, editing, **lettura** di messaggi di posta. Ad es:

Eudora

Outlook

Thunderbird

I messaggi in entrata e uscita **vengono archiviati sul server**

#### 20.2 Mail Server

Le **mailbox** (casella) **contengono**

i messaggi in ingresso diretti all'utente ancora da leggere

una coda di messaggi in uscita ancora da inviare

il **protocollo SMTP** per il dialogo tra mail server, allo scopo di scambiare i messaggi

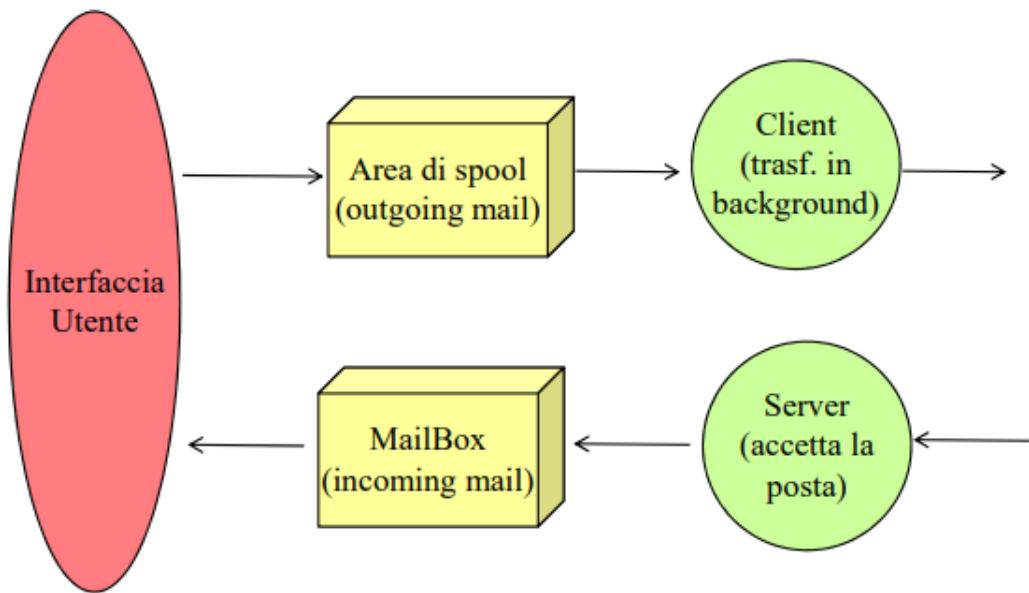
**Client:** mail server che invia

**Server:** mail server che riceve

Esempio di scenario: Alice invia un messaggio di posta a Bob.

1. Alice, grazie al suo mail user agent **compone il messaggio**.  
Alice **fornisce** allo user agent l'**indirizzo di destinazione**, cioè l'indirizzo di Bob
2. Lo user agent di Alice **spedisce il messaggio** al suo server di posta e il messaggio **viene accodato in attesa di invio**
3. Il **lato client** dell'SMTP sul server di posta di Alice **vede il messaggio** e **apre una connessione** al server SMTP sul server di posta di Bob
4. L'**SMTP client** **invia il messaggio nella connessione**
5. Sull'**host server** di posta di Bob, il **lato server** dell'SMTP **riceve il messaggio** e lo **colloca nella casella di posta di Bob**
6. Bob, quando vuole, chiede al suo user agent di leggere il messaggio

### 20.3 Schema di principio



La tecnica adottata dai server di posta si chiama **spooling**.

L'utente invia un messaggio, il sistema ne pone una copia in memoria spool – o **area di accomodamento della posta**, insieme all'id mittente, id destinatario, id macchina di destinazione e tempo di deposito.

Il sistema **avvia il trasferimento alla macchina remota**. Il sistema (client) stabilisce una connessione TCP con la macchina destinazione.

Se la connessione viene aperta, **inizia il trasferimento del messaggio**

Se il trasferimento va a buon fine il **client cancella la copia locale del film**

Se il trasferimento non va a buon fine, il **processo di trasferimento scandisce periodicamente l'area di spooling e tenta il trasferimento dei messaggi non consegnati**. Oltre un certo intervallo di tempo (definito dall'amministratore del server), se il messaggio non è stato consegnato **viene inviata una notifica all'utente mittente**.

### 20.4 Indirizzo

Un ricevente è identificato da un indirizzo email del tipo:

`local-part @ domain-name`

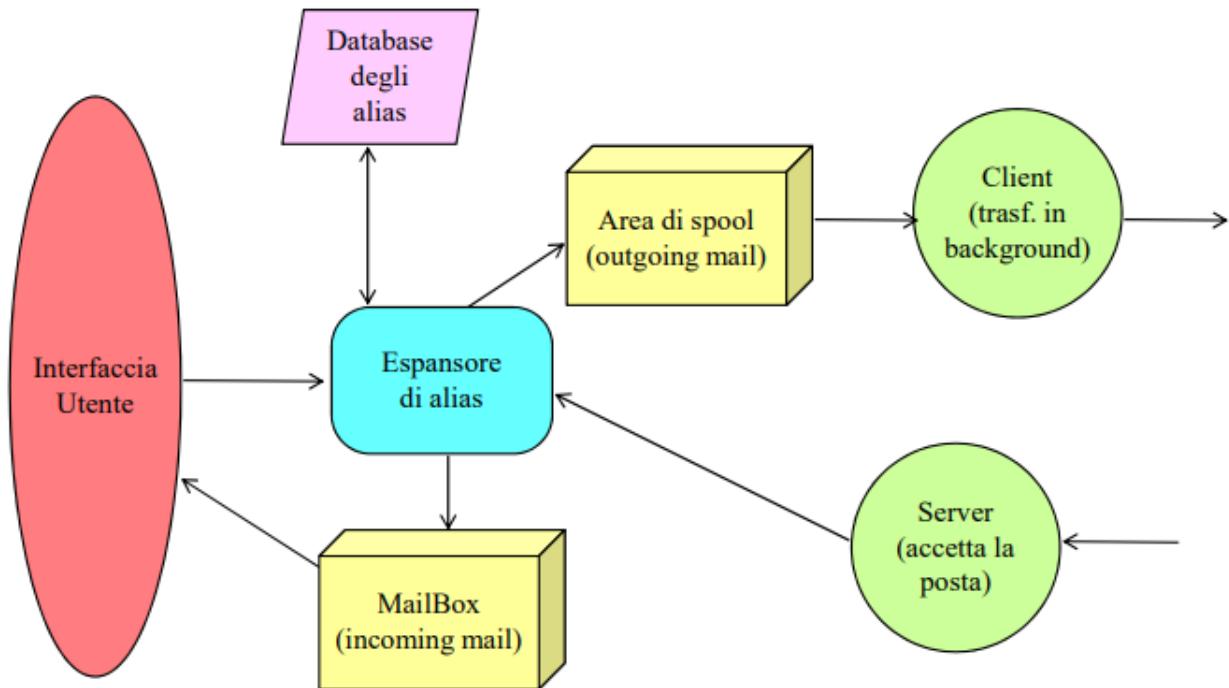
**domain-name**: specifica un **mail server**. Determina il nome di dominio della destinazione verso la quale la mail deve essere consegnata.

**Nome di dominio del server di posta**

**local-part**: specifica una **casella di posta** sul mail server. Spesso identica al login o al nome completo dell'utente.

**Indirizzo della casella di posta del destinatario**

## 20.5 Gestione Alias



L'alias è una **casella postale virtuale** che serve a **ridistribuire i messaggi** verso uno o più indirizzi di posta elettronica personali.

**Molti – Uno:** il sistema di alias permette ad un singolo utente di avere identificatori di mail multipli, assegnando un set di identificatori ad una singola persona.

Un utente → più indirizzi postali

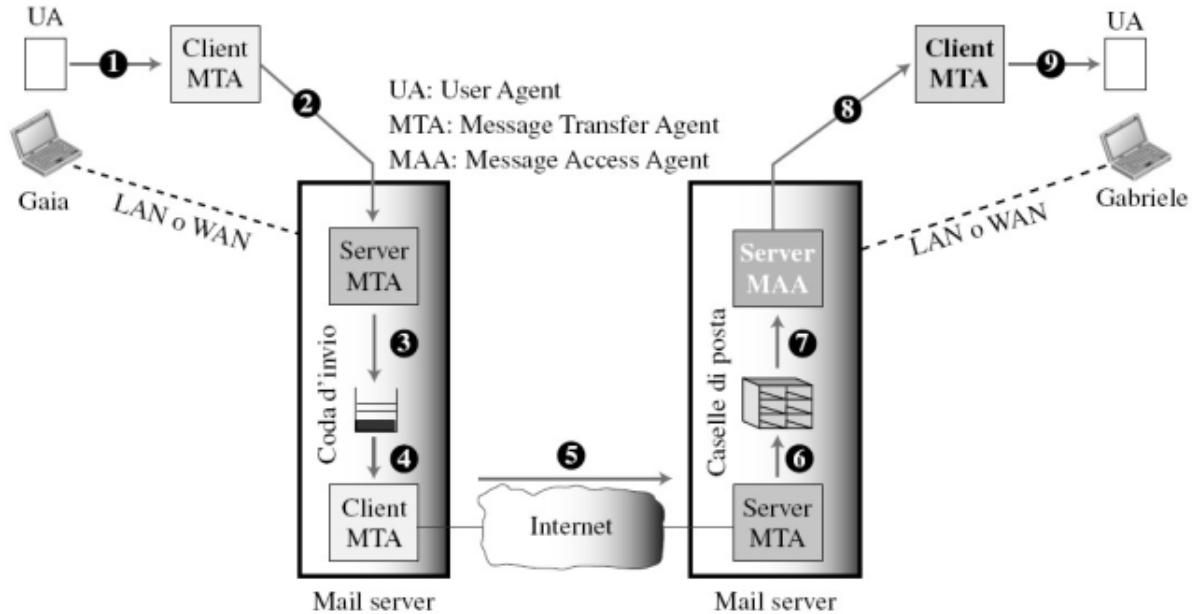
**Uno – Molti:** il sistema di alias permette di associare un gruppo di destinatari ad un singolo identificatore.

Un indirizzo postale → più utenti destinatari (es. mailing list)

**Espansione degli alias postali** Conversione di identificatori di indirizzi postali in uno o più indirizzi postali nuovi.

Se l'alias database specifica che all'indirizzo x deve essere assegnato il nuovo indirizzo y, l'espansione dell'alias riscriverà l'indirizzo di destinazione. Si provvederà poi a determinare se y specifichi un indirizzo locale o remoto e lo posizionerà nell'area di spool relativa.

## 20.6 Modello di riferimento



## 20.7 Simple Mail Transfer Protocol

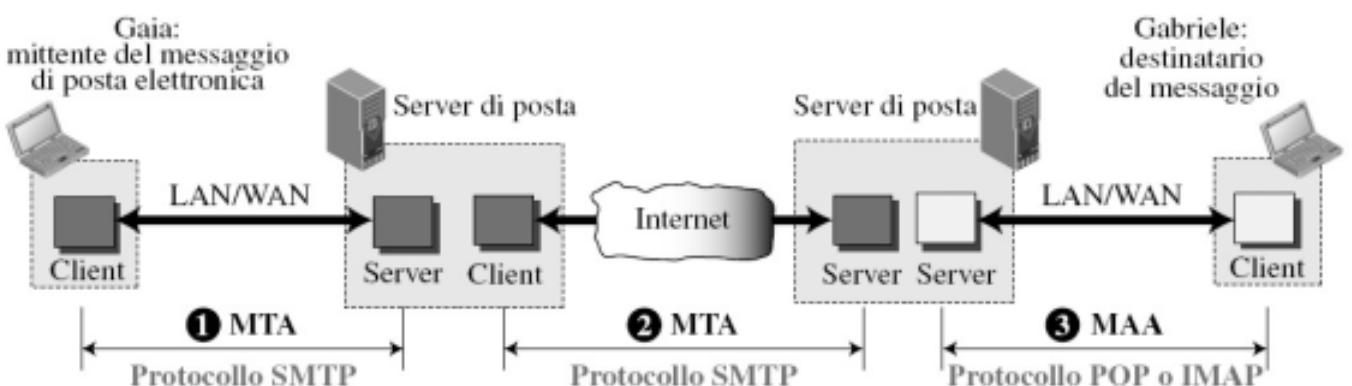
RFC 821, RFC 2821, RFC 5321

L'obiettivo dell'SMTP è trasferire le mail in maniera affidabile ed efficiente. L'SMTP è indipendente dal particolare sottosistema di trasmissione e richiede solo un canale che trasmetta dati ordinati in maniera affidabile.

Nonostante (nelle RFC) si parli specificamente del TCP, altri mezzi di trasporto sono possibili.

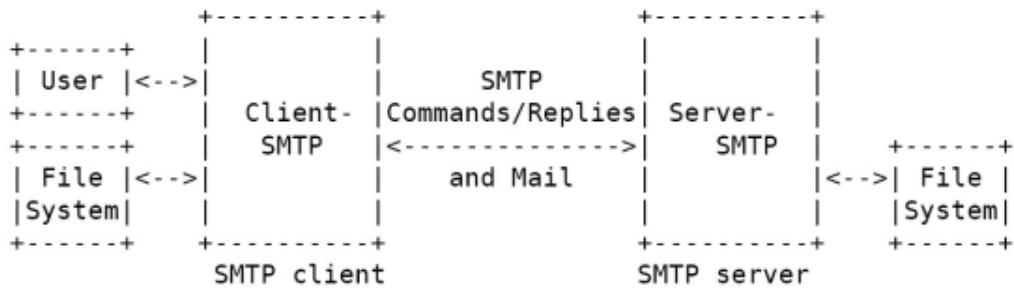
## 20.8 SMTP Mail Relaying

Una funzionalità importante dell'SMTP è la capacità di **trasportare mail attraverso reti multiple**. In questo modo, un messaggio di posta può passare attraverso molti dispositivi intermedi sul suo percorso dal mittente al destinatario.



## 20.9 Modello SMTP

Quando un client SMTP ha un messaggio da trasmettere, stabilisce una **canale di trasmissione bidirezionale** verso un server SMTP. La **responsabilità** del client SMTP è di **trasferire messaggi di posta a uno o più server SMTP**, o comunicarne il fallimento.



Un client SMTP determina l'indirizzo di un host appropriato che esegue un server SMTP **risolvendo un nome di dominio di destinazione** ottenendo o un **mail server intermedio** o l'**host finale** (tramite query DNS).

**Trasferimento** Il trasferimento di un messaggio può avvenire **in una singola connessione** tra mittente e destinatario o tramite **una serie di salti tra sistemi intermedi**.

In entrambi i casi, una volta che il server ha risposto positivamente alla fine della trasmissione dei dati della mail, avviene una **cessione di responsabilità del messaggio**: il **protocollo richiede che un server debba accettare la responsabilità di o consegnare il messaggio o segnalare il fallimento** della consegna.

### 20.9.1 Fallimenti nella consegna

Possibili problemi

Connessione con il mail server del mittente

Server **inesistente o irraggiungibile**

Connessione con il mail server del destinatario

Server **inesistente o irraggiungibile**

Inserimento nella casella di posta del destinatario

**User sconosciuto, casella piena**

In tutti i casi **il mittente riceve una notifica!**

L'unico caso in cui il destinatario non riceve il messaggio e il mittente non viene avvisato è se **qualcuno rimuove il messaggio** (intrusi, filtri antispam...)

## 20.10 Protocollo

Nella pratica viene usato il **protocollo TCP** sulla porta 25 per **consegnare in modo affidabile** i messaggi dal client al server. Questo avviene in tre fasi:

Handshaking

Trasferimento del messaggio

Chiusura della connessione

L'**interazione** avviene tramite **comando/risposta**, con comandi in testo ASCII e risposte con codici di stato e descrizione facoltativa.

I messaggi, header e body, sono in caratteri ASCII a 7bit.

**Protocollo semplice, di comandi e risposte** I comandi e (a meno di alterazioni dovute ad un'estensione del servizio) i dati dei messaggi SMTP sono trasmessi da mittente a destinario attraverso il canale di trasmissione in "right".

Una **risposta SMTP** è un **riconoscimento** (positivo o negativo) mandato in "righe" da destinatario a mittente attraverso il canale di trasmissione in risposta ad un comando.

La forma generale di risposta è un **codice numerico**, solitamente seguito da una stringa di testo.

```
S: MAIL FROM: <Smith@Alpha.ARPA>
R: 250 OK

S: RCPT TO:<Jones@Beta.ARPA>
R: 250 OK

S: RCPT TO:<Green@Beta.ARPA>
R: 550 No such user here

S: RCPT TO:<Brown@Beta.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: <CRLF>.<CRLF>
R: 250 OK
```

**Handshaking** Il client stabilisce la connessione e attende che il server invii "220 READY FOR MAIL", cioè la disponibilità a ricevere la posta.

Il client risponde con il comando HELO

Il server risponde identificandosi

A questo punto il client può trasmettere i messaggi

```
S: 220 Beta.GOV Simple Mail Transfer Service ready
C: HELO Alfa.EDU
S: 250 Beta.GOV
```

## 20.11 Comandi SMTP

Alcuni comandi:

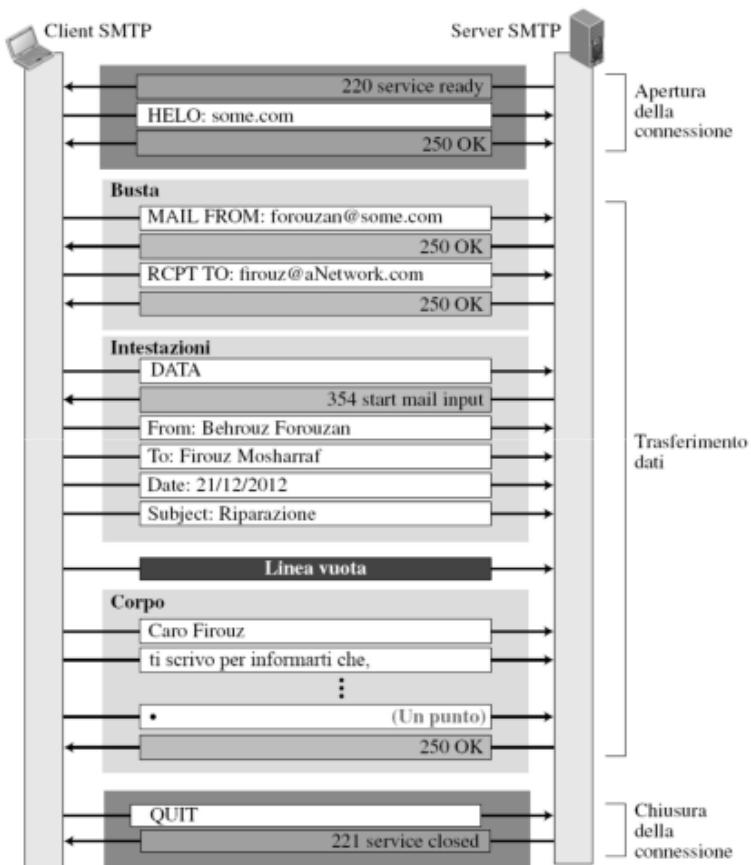
```
HELO <client identifier>
MAIL FROM:<reverse-path> [SP <mail-parameters>
<CRLF>
RCPT TO:<forward-path> [SP <rcpt-parameters>]
<CRLF>
DATA
QUIT
```

**Nota bene** <CRLF>.<CRLF> per terminare la fine di un messaggio

## 20.12 Esempio

```
S: MAIL FROM:<Smith@Alpha.ARPA>
R: 250 OK
S: RCPT TO:<Jones@Beta.ARPA>
R: 250 OK
S: RCPT TO:<Green@Beta.ARPA>
R: 550 No such user here
S: RCPT TO:<Brown@Beta.ARPA>
R: 250 OK
S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: <CRLF>.<CRLF>
R: 250 OK
```

## Esempio d'interazione



## 20.13 Formato messaggi mail

L' SMTP è il protocollo per lo scambio dei messaggi  
L'RFC 2822 è lo standard formato di testo:

**Linee d'intestazione** ad esempio

To:  
From:  
Subject:  
*Diversi dai comandi SMTP!*

*Linea vuota*

**Body**

Il "messaggio", solitamente in caratteri ASCII 7bit

Esempi di header:

A.3.1, minimo richiesto

```
Date: 26 Aug 76 1429 EDT
From: Jones@Registry.Org
Bcc:
-oppure-
Date: 26 Aug 76 1429 EDT
From: Jones@Registry.Org
To: Smith@Registry.Org
```

Il campo Bcc può essere buoto, il campo **To** deve avere almeno un indirizzo

A.3.2, con qualche campo addizionale

```
Date: 26 Aug 76 1430 EDT
From: George Jones<Group@Host>
Sender: Secy@SHOST
To: "Al Neuman"@Mad-Host, Sam.Irving@Other-Host
Message-ID: <some.string@SHOST>
```

A.3.3, il più complesso possibile

```
Date      : 27 Aug 76 0932 PDT
From      : Ken Davis <KDavis@This-Host.This-net>
Subject   : Re: The Syntax in the RFC
Sender    : KSecy@Other-Host
Reply-To  : Sam.Irving@Reg.Organization
To        : George Jones <Group@Some-Reg.An-Org>, Al.Neuman@MAD.Publisher
cc        : Important folk:
          Tom Softwood <Balsa@Tree.Root>,
          "Sam Irving"@Other-Host;;
Standard Distribution:
  /main/davis/people/standard@Other-Host,
  "<Jones>standard.dist.3"@Tops-20-Host;;
Comment   : Sam is away on business. He asked me to handle his mail for him. He'll be able to provide
           a more accurate explanation when he returns next week.
In-Reply-To: <some.string@DBM.Group>, George's message
X-Special-action: This is a sample of user-defined field-names. There could also be a field-name
                  "Special-action", but its name might later be preempted
Message-ID: <4231.629.XYzi-What@Other-Host>
```

## 20.14 Estensioni multimediali

L'RFC 822 permette di inviare **solo messaggi di testo** in ASCII. ricevere messaggi

Problema: permettere agli utenti di Internet di inviare/ricevere messaggi

In lingue con accenti (come l'italiano), in alfabeti non latini (come il russo o l'ebraico) o in lingue senza alfabeto (come il cinese o il giapponese)

Contenuti audio o video

### 20.14.1 MIME

Idea di base Continuare ad usare il formato del messaggio specificato in RFC 822 ma aggiungendo una struttura al message body e definendo regole di encoding per il trasferimento di testo non-ASCII.

Questo ha permesso di inviare messaggi MIME usando protocolli e mail server esistenti, rendendo però necessario cambiare gli user agent.

Multipurpose Internet Mail Extension RFC 2045 e RFC 2046. Questo standard di Internet estende il formato delle mail per supportare:

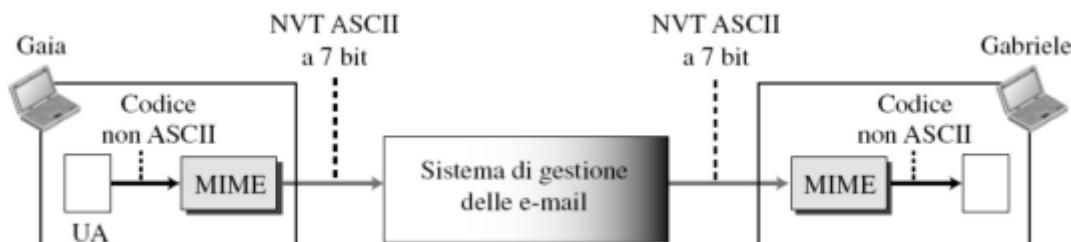
Testo in character set diversi dall'US-ASCII

Allegati non di testo

Corpo del messaggio diviso in più parti

Informazioni nell'header in character set non ASCII

Si realizza con linee di intestazione aggiuntive per dichiarare il tipo di contenuti MIME. La RFC 2045 definisce una serie di metodi per rappresentare dati binari in formato ASCII.



```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0          \\Versione MIME
Content-Transfer-Encoding: base64 \\Metodo usato per la codifica dei dati
Content-Type: image/jpeg    \\Dati multimediali (tipo, sottotipo, parametri...)
base64 encoded data ..... \\Dati codificati
.....base64 encoded data
```

Campi nell'header:

**MIME-Version:** un numero di versione per dichiarare a quale versione aderisce il formato del messaggio

**Content-Type:** usato per specificare il tipo del media e il sottotipo dei dati nel corpo del messaggio, e per specificare completamente la rappresentazione nativa di quei dati

**Content-Transfer-Encoding:** specifica la codifica applicata al corpo del messaggio. La codifica è solitamente applicata ai dati per consentire loro di passare attraverso i meccanismi di trasporto delle mail, che potrebbero avere limitazioni sui dati o sui character set.

La maggior parte dei tipi di media che è utile trasportare via mail sono **nativamente rappresentati con caratteri di 8bit** o dati binari. I **client e i server** SMTP si **aspettano messaggi ASCII** (caratteri a 7 bit), i **dati binari invece usato tutti e 8 bit** del byte (es. immagini, eseguibili, set estesi di caratteri). MIME fornisce **cinque schemi di transfer encoding** tra cui:

**Codifica ASCII** dei dati binari: **codifica base64**

Gruppi di 28bit divisi in 4 unità da 6bit, ciascuna unità inviata come un carattere ASCII

**Quoted-printable encoding**: per messaggi con pochi caratteri non-ASCII, più efficiente

Oggi giorno i mail server possono negoziare l'invio di dati in codifica binario (8 bit), se la negoziazione non ha successo si usano i caratteri ASCII.

## 20.14.2 Tipi MIME

Content-Type: type/subtype; parameters

**Text**

Esempi di subtype: plain, html

**Image**

Esempi di subtype: jpeg, gif

**Audio**

Esempi di subtype: basic (8bit mu-law encoded), 32kadpcm (32 kbps)

**Video**

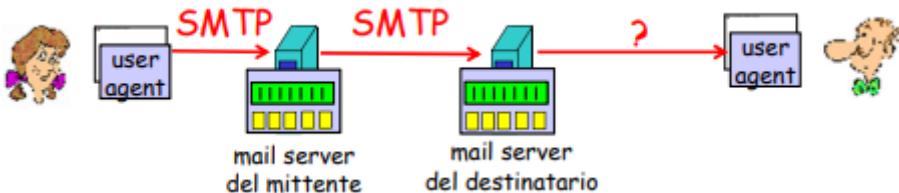
Esempi di subtype: mpeg

**Application**

Altri dati che devono essere processati da un'applicazione prima di essere visualizzabili.

Esempi di subtype: msword, octet-stream (dati arbitrari binari)

## 20.15 Protocolli di accesso alla mail



Per trasferire la posta fino al server del destinatario si utilizza l'SMTP, che è un protocollo di tipo push: la maggior parte dei dati è dal client al server. Per **leggere la posta** invece, è richiesto un protocollo di tipo pull, cioè in cui i dati viaggiano prevalentemente dal server al client.

**Messagge Access Agent** Attualmente si utilizzano due protocolli di accesso alla posta

**POP Post Office Protocol**

Semplice ma con funzionalità limitate. Attualmente è in uso la terza versione, POP3.

**Fase di autorizzazione**

L'accesso avviene con lo **user agent** che **apre una connessione TCP sulla porta 110** e si autentica con i comandi

user: specifica lo username

pass: specifica la password

ed il server risponderà con OK o ERR.

### Fase di transazione

Il client può utilizzare i comandi:

- list: visualizza la lista dei messaggi
- retr: preleva il messaggio per numero
- dele: elimina il messaggio dal server
- quit: chiude la sessione

### Fase di aggiornamento

Dopo aver ricevuto quit il server cancella i messaggi marcati per la rimozione

Il POP3 ha due modalità:

**Delete:** i messaggi vengono automaticamente eliminati dalla mailbox dopo il prelievo

**Keep:** i messaggi vengono conservati dopo il prelievo

Inoltre non mantiene le informazioni di stato tra le sessioni, solo le cancellazioni sono permesse.

**IMAP** Internet Mail Access Protocol

Ha più funzionalità, ed è più complesso, del POP3. Consente tra le altre cose di **manipolare i messaggi memorizzati sul server** con cartelle, e di **estrarre solo alcuni componenti dei messaggi** come solo l'intestazione se si sta usando una connessione lenta.

**HTTP**

Es. Hotmail, Yahoo! Mail, ...

## 21 Livello di Trasporto

### 21.1 Obiettivi

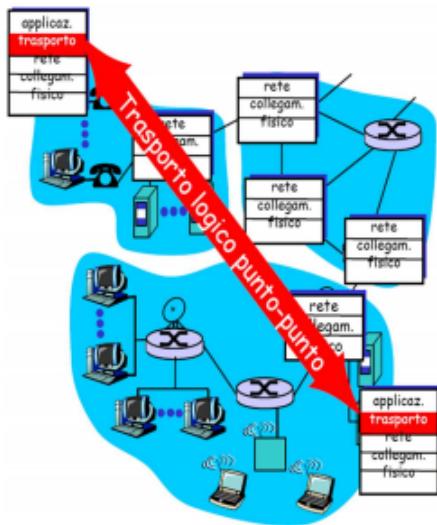
Realizzare una connessione logica fra processi residenti in host diversi.

**Logico** I processi si comportano come se gli host fossero direttamente collegati, non si preoccupano dei dettagli dell'infrastruttura usata fisica per la comunicazione.

#### Offrire servizi allo strato applicativo

Un'applicazione interagisce con i protocolli di trasporto per trasmettere e ricevere dati. L'applicazione sceglie lo stile di trasporto: sequenza di messaggi singoli o una sequenza continua di byte. Il programma applicativo passa i dati nella forma richiesta al livello di trasporto per la consegna.

Utilizza i servizi dello strato di rete.



I servizi di trasporto:

Forniscono la **comunicazione logica** fra processi applicativi di host differenti

I **protocolli** di trasporto vengono **eseguiti nei sistemi terminali**

## 21.2 Caratteristiche

**Servizio privo di connessione** In un servizio privo di connessione il **processo mittente consegna i messaggi al livello di trasporto uno per uno**. Il livello di trasporto **tratta ogni messaggio come entità singola** senza mantenere alcuna relazione fra essi. I segmenti possono **non essere consegnati o non arrivare in ordine**.

**Servizio orientato alla connessione** In un servizio orientato alla connessione client e server **stabiliscono una connessione logica**.

## 21.3 Servizi offerti

**Multiplexing e Demultiplexing** Il termine **multiplexing** si riferisce al caso in cui **un'entità riceve informazioni da più di una sorgente**. Lo strato trasporto provvede **allo smistamento dei pacchetti fra rete e applicazioni**.

**Demultiplexing** invece al caso in cui **un'entità trasmette informazioni a più di un destinatario**. Lo strato trasporto provvede all'**accompagnamento dei flussi dati dai processi verso la rete**, "imbustando" i dati ricevuti dall'alto con un preambolo.

Il livello di trasporto effettua il **multiplexing sul mittente** e il **demultiplexing sul destinatario**.

Si basano sui socket address dei processi e quindi dipendono dal numero di porta su cui i processi sono attivi.

### Controllo degli errori

## 22 UDP

Rispetto al TCP, lo UDP è meno complesso, offre meno servizi, ma è **più indicato in contesti in cui occorre un controllo completo della temporizzazione** cioè in applicazioni time-sensitive come lo streaming.

**Nessuna garanzia** Lo **User Datagram Protocol** è un **protocollo di consegna a massimo sforzo**. I segmenti UDP possono andare perduti o essere consegnati fuori sequenza. Notare però che l'affidabilità può essere aggiunta a livello applicazione.

**Orientato al messaggio** Ogni **datagramma UDP** è **indipendente** dall'altro. I processi devono inviare **messaggi di dimensioni limitate, incapsulabili in un datagramma UDP**.

### 22.1 Proprietà

**Nessuna connessione** quindi non si introduce ritardo.

**Semplice:** non viene gestito lo strato di connessione. Le intestazioni sono corte e **non ha controllo di congestione e di flusso**, quindi si possono sparare dati a raffica.

#### Checksum facoltativo

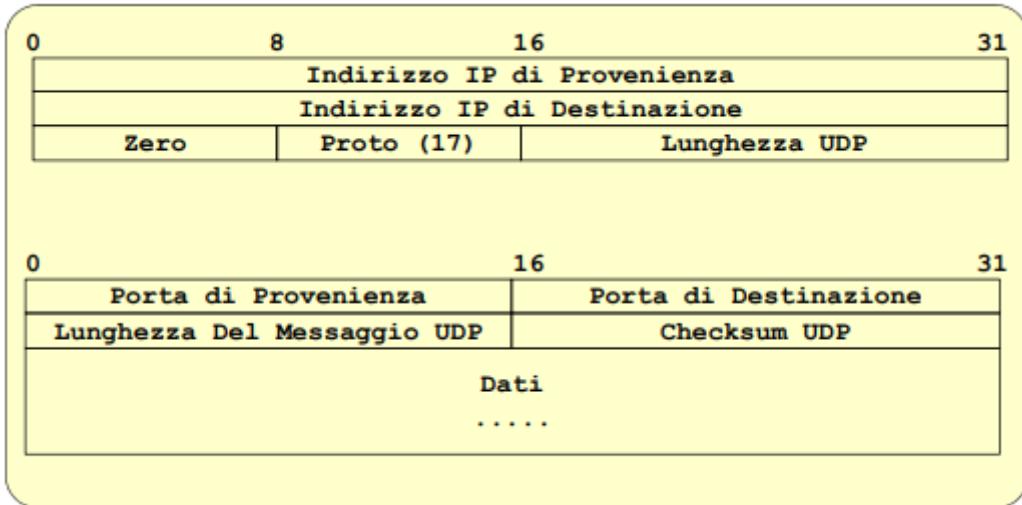
Facile e leggero da gestire, non richiede particolari meccanismi

Concepito solo in funzione del DNS e del TFTP, nel 1980

Utilizzato spesso in **applicazioni multimediali**: tolleranza a piccole perdite e sensibilità alla frequenza

Altri impieghi: DNS, SNMP...

## 22.2 Datagramma UDP



8 Byte di intestazione

La prima parte è uno pseudo-header che *non* viene trasmesso

**Porta:** numeri di porta della comunicazione (per il demultiplexing è usato solo quello di destinazione).

La **porta di provenienza** è un campo facoltativo. Quando è significativo, indica la porta del processo mittente, e può essere considerata come la porta alla quale rispondere in assenza di altre informazioni. Se non è usata, viene riempita col valore 0.

**Lunghezza** del messaggio: lunghezza totale del datagramma UDP, header+dati, è di 65535 Byte.

**Checksum** checksum dell'intero pacchetto, compreso lo pseudo-header ovvero con il controllo dell'indirizzo host). **Opzionale**, se non è usato è posto a 0xFF (complemento a 1 di 0x00)

Controllo dell'errore end-to-end: il pacchetto corrotto è scartato ma l'utente non viene notificato

L'indirizzo di livello trasporto è una coppia composta dall'indirizzo IP e dalla porta del destinatario (con indicazione dell'indirizzo IP e della porta del mittente)

## 22.3 Calcolo del checksum

### Mittente

Tratta il contenuto del segmento **come una sequenza di interi da 16 bit**

**Checksum:** somma (complemento a 1) i contenuti del segmento

Il mittente pone il valore della checksum nel campo checksum del segmento UDP

### Destinatario

**Calcola** il checksum del segmento ricevuto

**Controlla se** il checksum calcolato è **uguale** al valore del campo checksum

No – Errore rilevato

Si – Nessun errore rilevato

## 22.4 TCP vs UDP

### UDP

#### Trasferimento dati non affidabile

Non fornisce: setup della connessione, affidabilità, controllo del flusso, controllo della congestione, timing o garanzia di banda

Richiede **minor overhead**

### TCP

**Connection-oriented:** richiesto **handshake** tra client e server

**Trasporto affidabile** tra i processi

**Controllo di flusso:** il mittente non satura il destinatario

**Controllo della congestione:** limita il mittente a rete sovraccarica

Non fornisce: **timing, banda minima**

**Pro e contro** Nella programmazione di rete si deve ricordare che

Il TCP offre un **servizio di trasporto a stream**, quindi si può leggere da un input di rete quanti byte si desiderano

L'UDP invece offre un **servizio a messaggi**, quindi occorre leggere **tutto** il messaggio in arrivo

L'UDP è adeguato per

Processi che richiedono uno scambio di dati con volume limitato **in caso di scarso interesse al controllo del flusso e degli errori**

Processi con **meccanismi interni di controllo del flusso e degli errori**

Trasmissioni **multicast**

**Applicazioni interattive in tempo reale** che non tollerano ritardi variabili

**Considerazioni** La RFC 768 che definisce l'UDP è del 1980, non è stata cambiata o integrata da altre RFC ed è di tre pagine.

La RFC 793 che definisce il TCP è del 1981, è stata aggiornata o cambiata da diverse altre RFC, ad esempio

2018 - TCP Selective Acknowledgement Options

1146 - TCP Alternate Checksum Options

2581 - TCP Congestion Control

1323 - TCP Extensions for High Performance

1693 - An Extension to TCP : Partial Order Service

1792 - TCP/IPX Connection Mib Specification

e la lunghezza totale è di 85 pagine, con un glossario di 67 voci.

## 23 TCP

### 23.1 Proprietà

**Orientato allo stream** La lunghezza in byte è **indefinita** a priori. Il servizio di consegna a destinazione passa esattamente la medesima sequenza di byte che il mittente ha passato al servizio di consegna nell'origine. Il TCP vede i **dati come un flusso di byte ordinati**, ma **non strutturati**.



**Orientato alla connessione** I processi effettuano un **handshake** prima dello scambio dei dati. L'**handshake** è uno **scambio di informazioni preliminari per preparare il trasferimento dei dati**.

Orientato perché lo stato della connessione risiede sui punti terminali, non sugli elementi intermedi della rete (es. router).

La **connessione** è vista dagli applicativi (USERS) come un **circuito fisico dedicato**. Il **TCP** è quindi capace di fornire servizi di tipo **connection-oriented**, mentre il protocollo **IP** su cui si appoggia è in grado di **fornire servizi connection-less**.

**Connessione full-duplex** Il **flusso dati tra due host può avvenire contemporaneamente nelle due direzioni**. Le due direzioni sono slegate e la connessione è punto-a-punto.

**Trasferimento bufferizzato** Il software del protocollo TCP è libero di **suddividere lo stream in segmenti indipendenti** (pacchetti), in maniera indipendente dal programma applicativo che li ha generati. Per fare questo è necessario disporre di un **buffer dove immagazzinare la sequenza di byte**. Appena i dati sono sufficienti per riempire un **datagramma** ragionevolmente grande, questo viene trasmesso attraverso la rete.

### 23.2 Funzioni del segmento TCP

Funzioni base per il trasferimento di dati:

Capacità di trasferire un **flusso continuo** di byte

Trasferimento bidirezionale (**full duplex**)

**Multiplexing**: consente di assegnare una data **connessione** ad un dato processo.

Permette una comunicazione da processo a processo.

**Controllo della connessione**: meccanismi di inizio e fine trasmissione (controllo della sessione)

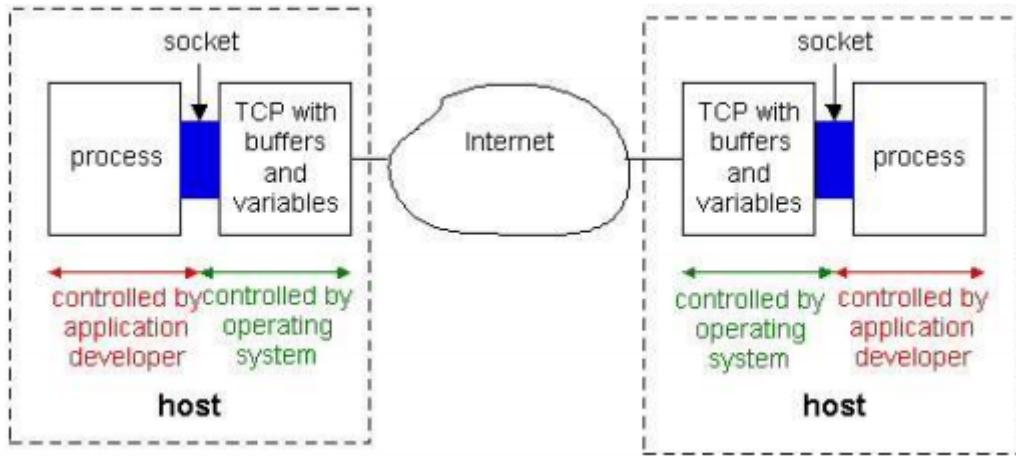
**Trasferimento dati ordinato e affidabile**: si intende la capacità di correggere tutti i tipi di errore, quali:

- dati corrotti
- segmenti persi
- segmenti duplicati
- segmenti fuori sequenza

**Controllo di flusso**: evitare di spedire più dati di quanti il ricevitore sia in grado di trattare

**Controllo di congestione**: ha lo scopo di recuperare situazioni di sovraccarico nella rete

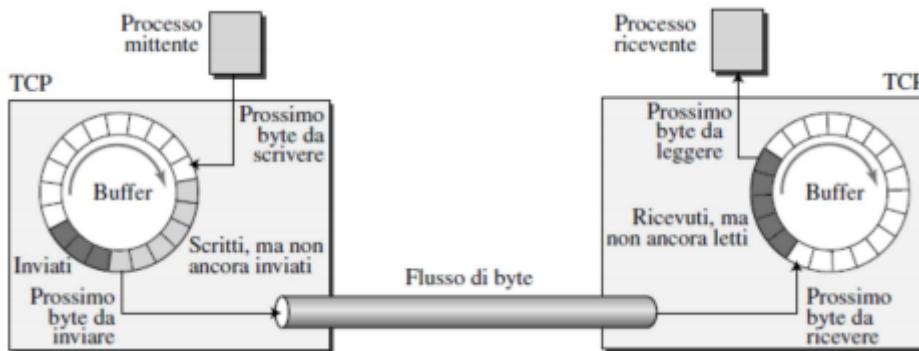
### 23.3 Processi e Socket TPC



I processi in esecuzione su diverse macchine comunicano tra loro mandando messaggi **attraverso i socket**. Un po' come se ogni processo fosse una casa e ogni socket fosse la porta, **un socket è la porta tra il processo dell'applicazione e il TCP**.

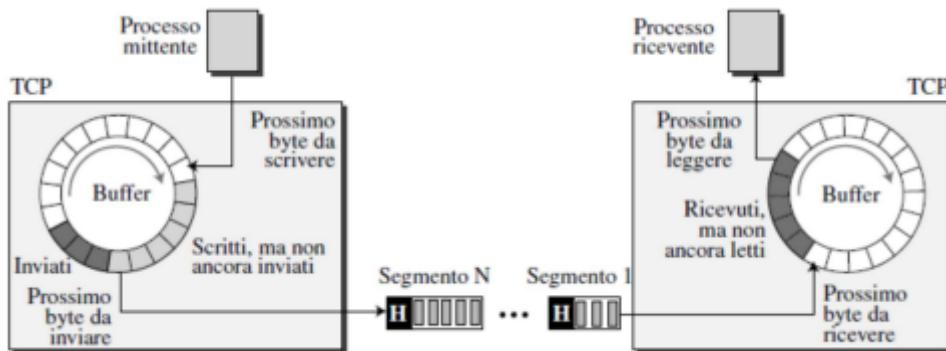
Lo sviluppatore ha il controllo su tutta la parte del **lato applicazione del socket**, ma non può controllare il **lato del livello di trasporto**. Al più, lo sviluppatore può specificare alcuni parametri del TCP, come la dimensione massima del buffer e dei segmenti.<sup>3</sup>

### 23.4 Trasferimento bufferizzato



I processi a livello applicativo **scrivono e leggono byte nel/dal buffer**, e questo può avvenire a velocità diverse.

### 23.5 Segmenti TCP



Il flusso di byte è **partizionato in segmenti**: ogni segmento ha il suo header e viene consegnato al livello IP.

<sup>3</sup><http://www3.gdin.edu.cn/jpkc/dzxnw/jsjkj/chapter2/26.htm>

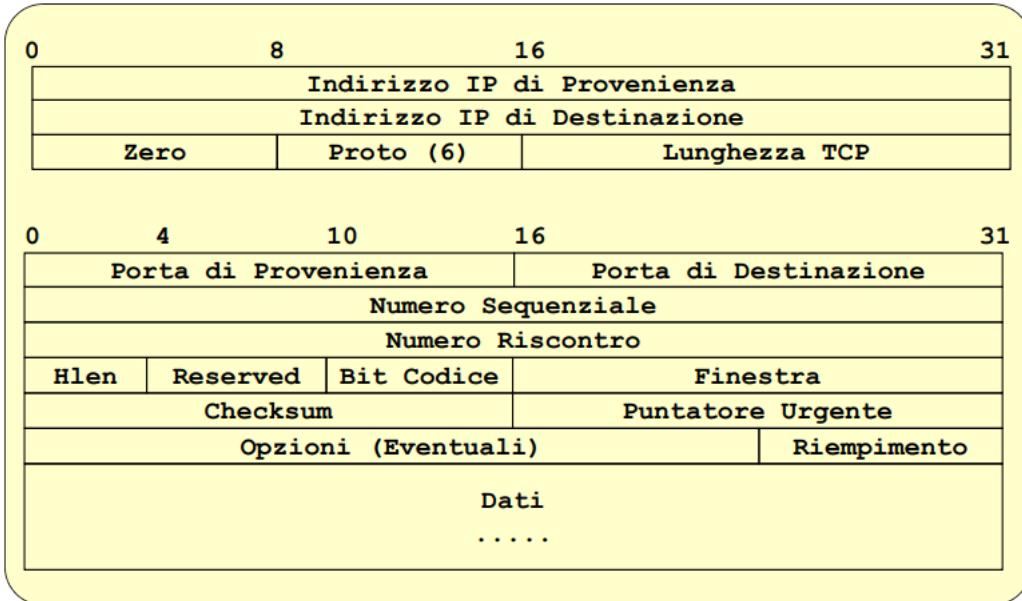
## 23.6 Numeri di sequenza e di riscontro

Il TCP numera i byte anziché i segmenti

**Numero di sequenza** Associato al segmento, è il numero (nel flusso) del primo byte (di dati) del segmento. In genere si parte da un **initial sequence number** generato in modo casuale e diverso da 0.

**Numero di riscontro**  $1 + \text{numero dell'ultimo byte correttamente ricevuto}$ . I riscontri sono interpretati come **cumulativi**: se ricevo  $\text{ACK} = y$  significa che *aspetto il byte y* e quindi ho ricevuto tutti i byte fino a  $y - 1$ .

## 23.7 Segmento TCP



Lo **pseudo header** NON viene trasmesso.

Ha valenza logica, contiene **IP sorgente** e **destinazione**. Queste info **vengono inserite** effettivamente e trasmesse dal livello inferiore.

**Proto:** codice del protocollo.

**Lunghezza TCP:** lunghezza del segmento TCP escluso lo pseudoheader, server per il calcolo del checksum.

Nel segmento TCP bisogna notare la presenza di:

Numero di **sequenza**

Numero di **riscontro**

**Finestra**

Essi permettono il **controllo del flusso**, il meccanismo di **ritrasmissione** ed il **riordino** dei pacchetti in ricezione, necessari per la struttura stream-based del TCP

Inoltre è presente un campo **urgent** che permette la trasmissione dei dati "fuori banda", ovvero a **priorità maggiore degli altri** (la loro gestione però è affidata all'applicazione).

### 23.7.1 Campi del segmento TCP

**Porta** (16 bit): numeri di porta della comunicazione

**Numero di sequenza** (32 bit): è il **numero di sequenza** nello stream **del primo byte di dati di questo segmento**.

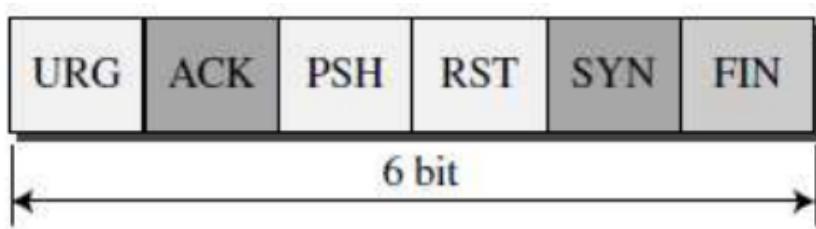
Se il flag SYN è settato, allora il numero di sequenza è **ISN** (Initial Sequence Number) e il primo byte di dati è **ISN + 1**.

**Numero di riscontro** (32 bit): se il flag ACK è settato, allora questo campo **contiene il valore del prossimo numero di sequenza che il mittente del segmento si aspetta di ricevere dall'altro host**. Una volta che la connessione è stabilita è sempre inviato.

**Hlen** (4bit): **lunghezza dell'header** TCP espressa in parole da 4 byte

La lunghezza dell'header può variare da 20 a 60 byte)

**Bit codice**: sono **6 flag** e, da sinistra a destra, servono per:



**URG**: il campo **puntatore urgente** contiene dati significativi da trasferire in via prioritaria

**ACK**: il campo **numero di riscontro** contiene dati significativi

**PSH**: funzione push, cioè **trasferimento immediato** dei dati in un **segmento dal livello trasporto al livello applicativo**

**RST**: reset della connessione

**SYN**: sincronizza il numero di sequenza

**FIN**: non ci sono altri dati utente – chiusura della connessione

**Finestra di ricezione** (16 bit): indica il **numero di byte di dati** a partire da quello indicato nel campo **numero di riscontro che il mittente di questo segmento è in grado di accettare**.

Serve per il controllo del flusso.

**Checksum** (16 bit): checksum dell'intero pacchetto (compreso lo pseudo header).

Serve per il **rilevamento degli errori** in caso di alterazione dei bit del segmento, e si calcola come per l'UDP (ma per il TCP + obbligatorio)

**Opzioni** (facoltativo, lunghezza variabile di massimo 40 byte): **negoziazione di vari parametri**, ad esempio: dimensione massima del segmento (**MSS**), selective acknowledgment supportato e blocchi di dati riscontrati selettivamente.

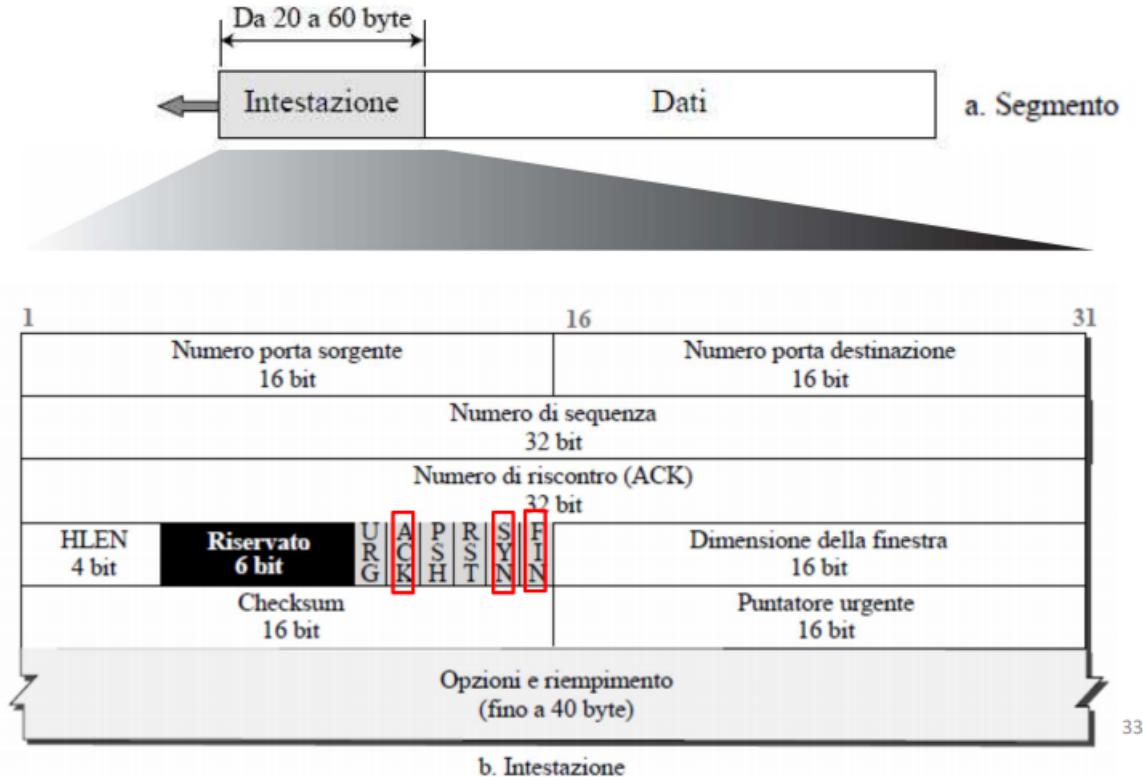
Le opzioni sono sempre multipli di 8 bit e il loro valore è incluso nel checksum.

**Puntatore urgente** (16 bit): questo campo è un **offset positivo a partire dal numero di sequenza del segmento corrente**. Viene **interpretato solo** se il bit URG è uguale a 1.

**Punta al primo byte di dati non urgenti** in attesa nella coda di ricezione. Nel segmento contenente dati urgenti **deve essere presente almeno un byte di dati**.

Ad es. se un segmento contiene 400 byte di dati urgenti e 200 byte di dati non urgenti, il puntatore urgente vale 400.

### 23.7.2 Formato del segmento TCP



## 23.8 Gestione della connessione

### 23.8.1 Three-Way Handshake

**Handshake a tre vie** Dopo l'handshaking a livello di trasporto, non c'è più distinzione tra client e server. Sequenza:

- Il **client invia una richiesta di connessione** ad un server TCP.  
SYN è attivo  
Il segmento non contiene dati  
Si trasmette anche un numero di sequenza iniziale casuali  
Es. SYN = 1, clientISN = 41
- Il **server estrae il segmento**, alloca i buffer e le variabili TCP per la connessione.  
Invia in **risposta un segmento di connessione garantita** chiamato SYNACK  
SYN è attivo  
Il numero di sequenza è il valore iniziale (es. 78)  
ACK è attivo, il server aspetta clientISN + 1 (es. 42)  
Es. SYN = 1, ACK = clientISN + 1, serverISN = 78
- Il **client alloca buffer e variabili** di connessione, poi **manda un riscontro positivo del messaggio del server**.  
SYN è inattivo, questo segmento può già trasportare dati.  
Il prossimo dato data è clientISN + 1 (es. 42) ed il client attende serverISN + 1 (es. 79)  
SYN = 0, riscontro serverISN + 1

#### - Inizia lo scambio di dati

I primi segmenti non hanno carico utile. All'arrivo del primo segmento il server inizializza due buffer e le variabili, necessari per il controllo del flusso e delle congestione.

All'arrivo del riscontro del primo segmento il client alloca due buffer e le variabili, per lo stesso motivo.

Alla ricezione del terzo segmento la connessione è instaurata.

### 23.8.2 Perché a tre vie

Esempio degli scalatori:

*Ti tengo la corda? →*

*← Si, tienimi*

*Ok →*

Le cose possono andare male in tanti modi diversi, alcuni di questi sono **situazioni indistinguibili per A mittente**

*Ti tengo la corda? →*

*Ti tengo la corda? → X*

*X ← Si, tienimi*

Altre invece sono **situazioni indistinguibili per B destinatario**

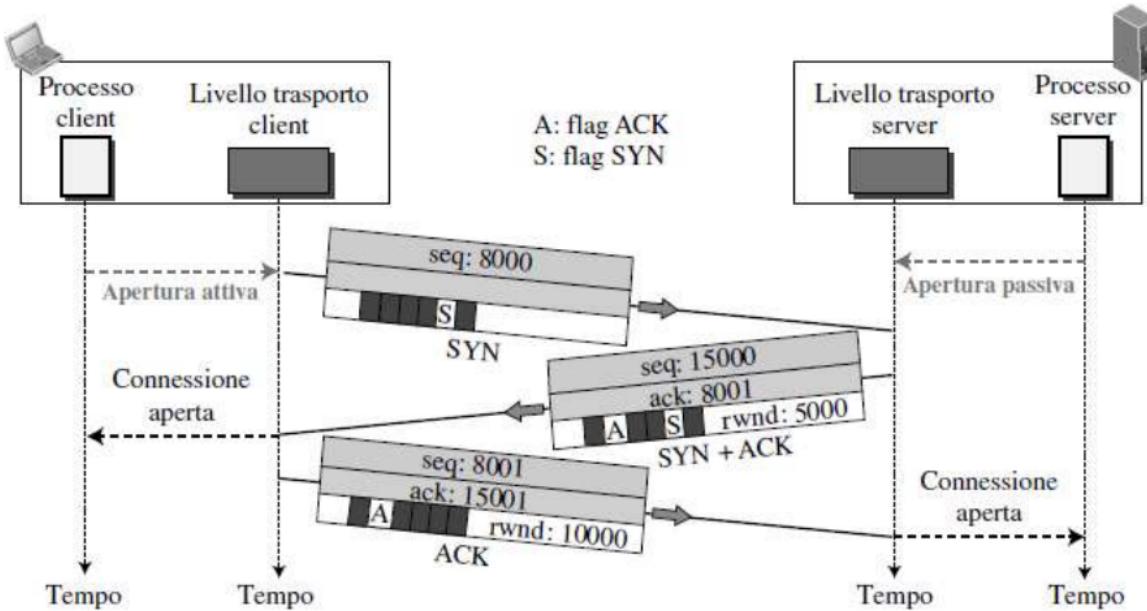
*Ti tengo la corda? →*

*Ti tengo la corda? → X*

*← Si, tienimi*

*Ok → X*

### 23.8.3 Esempio



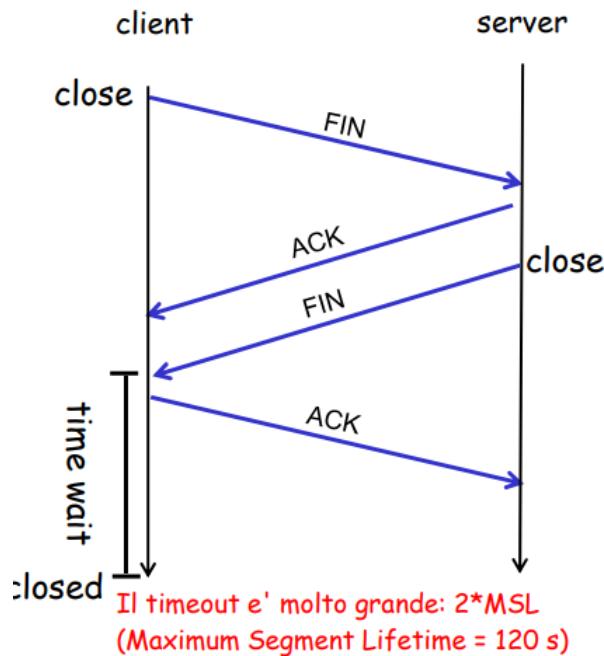
SYN e SYN + ACK non contengono dati utente ma consumano un numero di sequenza

#### 23.8.4 Chiusura della connessione con handshake

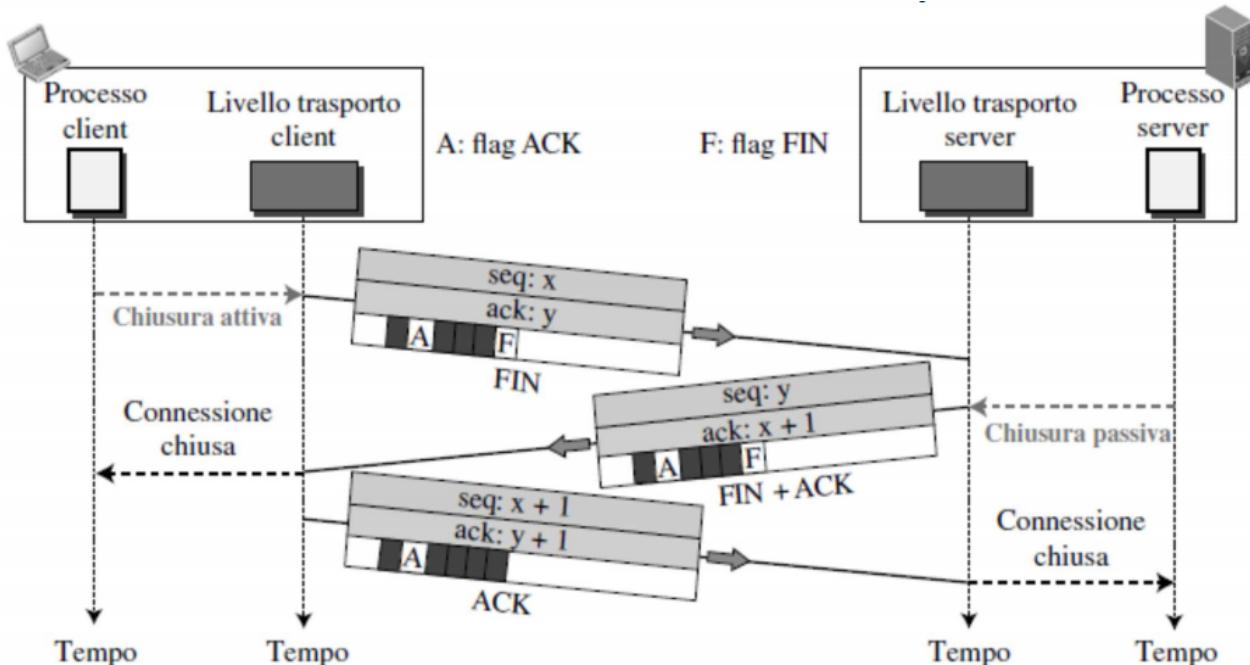
In una connessione TCP normale, dopo l'ACK finale per la chiusura si aspetta un tempo di **timeout molto grande**.

$\text{timeout} = 2 * \text{MSL}$

MSL è il Maximum Segment Lifetime = 120s



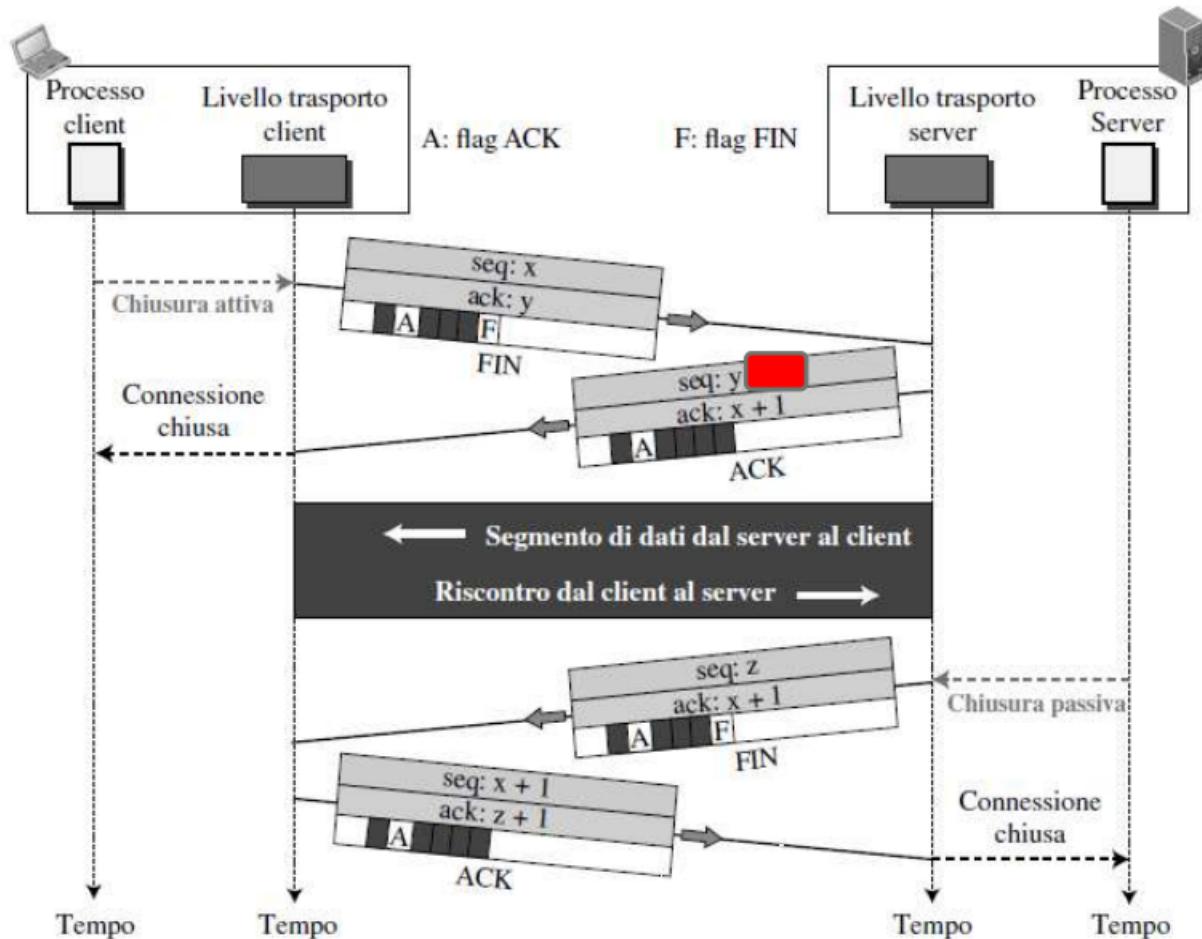
Invece con il three-way handshake, dopo l'ACK finale la connessione è immediatamente chiusa.



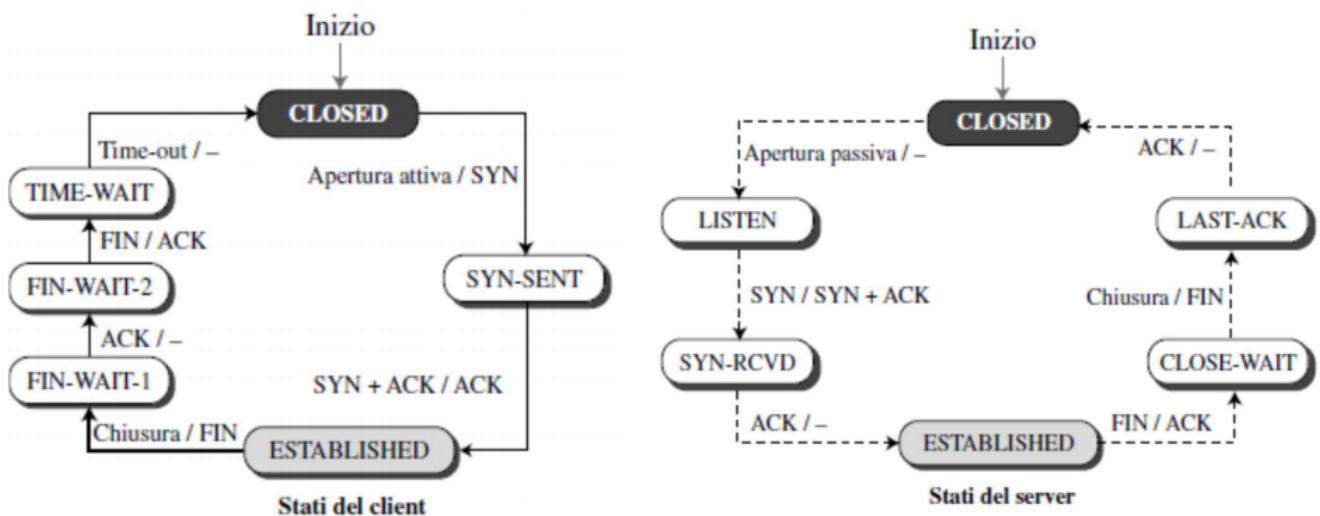
Un FIN che non trasporta dati ma consuma un numero di sequenza. Idem per FIN + ACK.

### 23.9 Half-Close

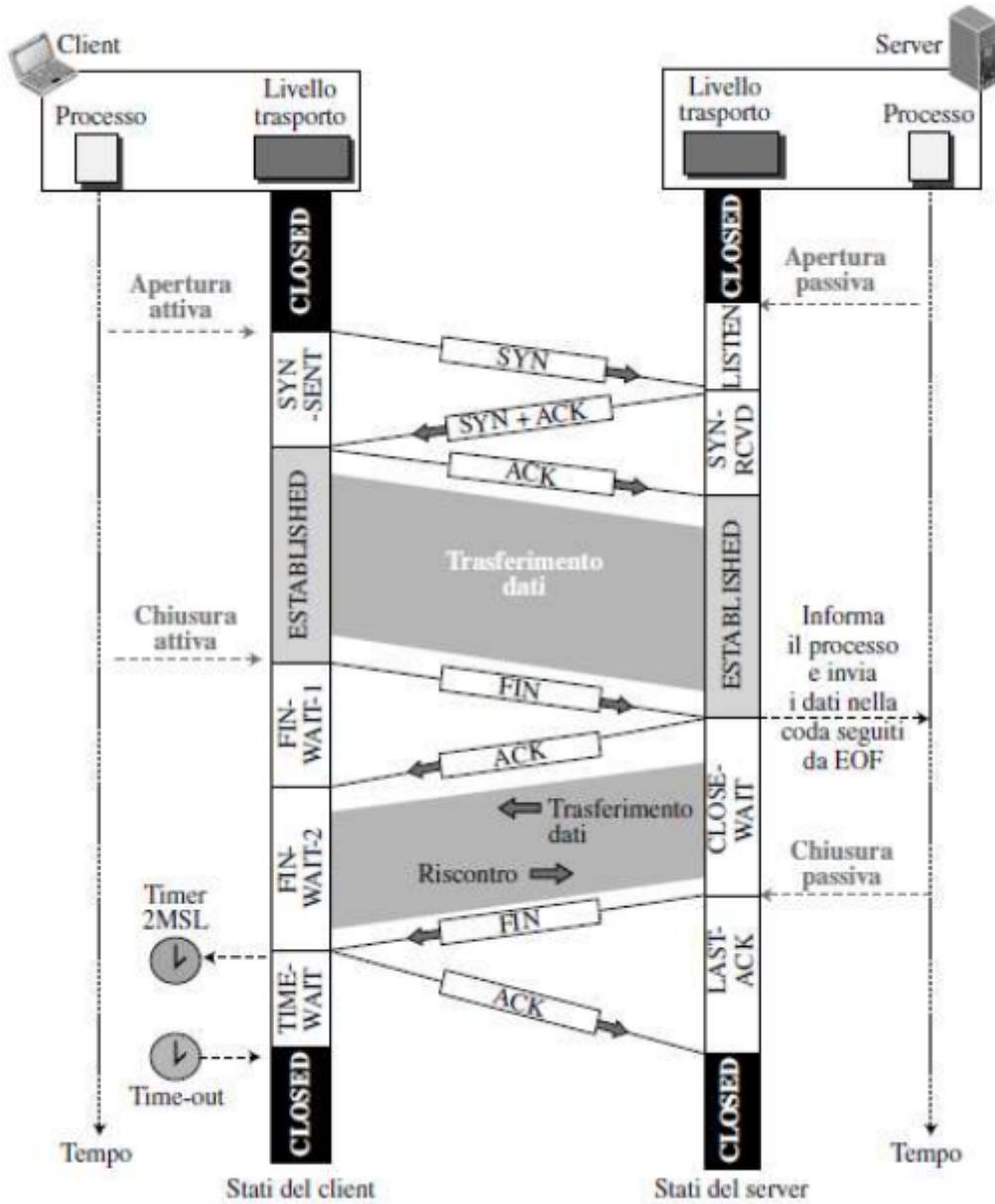
Uno dei due processi può smettere di inviare dati mentre sta ancora ricevendo dati.  
`TCP.close(conn) = "I have no more data to send."`



Di seguito l'ASF corrispondente agli stati dell'**half-close**



### 23.9.1 Scenario Half-Close



### 23.9.2 Stato Time-Wait

**Finale** Time-Wait è lo stato finale in cui il capo di una connessione che esegue la chiusura attiva resta prima di passare alla chiusura definitiva della connessione. Dura due volte la Maximum Segment Lifetime.

**MSL** La Maximum Segment Lifetime è la stima del massimo periodo di tempo che un pacchetto IP può vivere sulla rete.

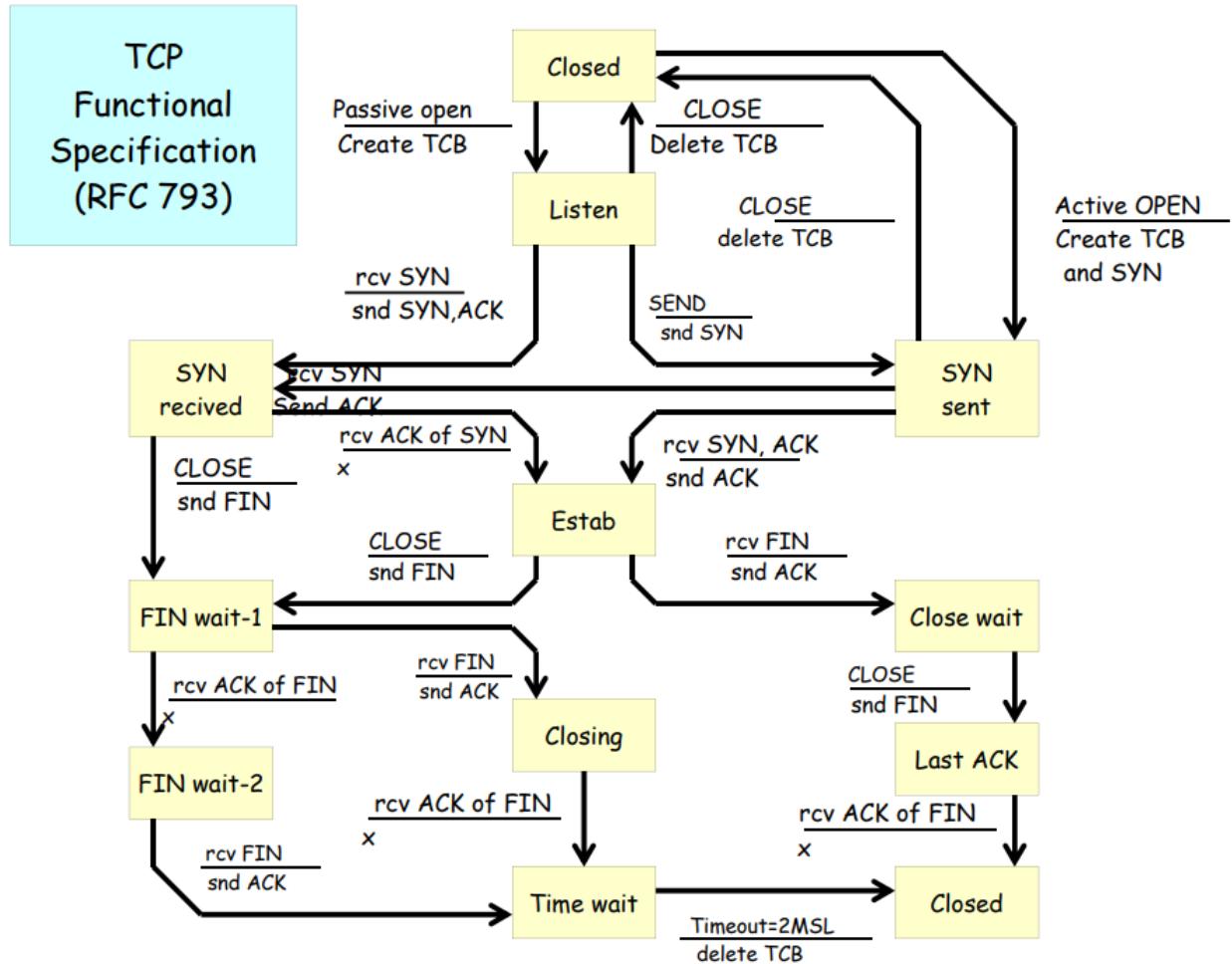
Questo tempo è limitato perché ogni pacchetto IP può essere ritrasmesso dai router un numero massimo di volte detto **hop limit**.

Ogni implementazione del TCP sceglie il proprio valore per la MSL (RFC 1122 indica 2 minuti, Linux usa 30 secondi).

**Perché Time-Wait** Lo stato Time-Wait viene usato dal protocollo per due motivi principali:

- Implementare in maniera affidabile la terminazione della connessione in entrambe le direzioni.  
Se l'ultimo ACK della sequenza viene perso, chi esegue la chiusura passiva manderà un ulteriore FIN, chi esegue la chiusura attiva deve mantenere lo stato della connessione per essere in grado di rinviare l'ACK.
- Consentire l'eliminazione di segmenti duplicati in rete

## 23.10 Stati del TCP



Gli stati del TCP:

**LISTEN:** attesa di una richiesta di connessione remota TCP

**SYN-SENT:** attesa di ricezione di una connection request corrispondente dopo aver mandato la propria connection request

**SYN-RECEIVED:** attesa dell'ACK dopo aver entrambi spedito la connection request

**ESTABLISHED:** connessione aperta, i dati possono essere trasferiti all'utente. Questo è lo stato normale per la fase di trasferimento dati della connessione

**FIN-WAIT-1:** attesa della richiesta di terminazione della connessione dal TCP remoto, oppure un'ACK sulla richiesta di terminazione della connessione precedentemente spedita

**FIN-WAIT-2:** attesa della richiesta di terminazione attiva della connessione dal TCP remoto

**CLOSE-WAIT:** attesa della richiesta di terminazione della connessione dall'utente locale. Si è ricevuta la richiesta di chiusura della connessione e si è già mandato l'ACK, quindi la chiusura passiva è già avvenuta e si attende di avviare la chiusura attiva

**CLOSING:** attesa dell'ACK sulla chiusura della connessione prima di andare in TIME-WAIT

**LAST-ACK:** attesa dell'ACK sulla chiusura attiva della connessione. Chiusura passiva effettuata e chiusura attiva iniziata

**TIME-WAIT:** attesa di abbastanza tempo per assicurarsi che l'host remoto riceva l'ACK per la chiusura della connessione

**CLOSED:** nessuna connessione

## 23.11 Trasferimento Dati Affidabile

Un **segmento TCP** può essere **smarrito** o **corrotto**. Il TCP crea un **servizio di trasferimento dati affidabile** sul servizio inaffidabile dell'IP.

**Checksum** I controlli sono **obbligatori** ed i segmenti corrotti vengono scartati.

### Riscontri

**Numero di sequenza** di un segmento è il **numero del primo byte del segmento nel flusso di byte**. I numeri di sequenza si applicano ai byte, non ai segmenti trasmessi.

**Numero di riscontro:** numero di sequenza successivo che l'host attende.

**Riscontro cumulativo:** si effettua il **riscontro dei byte fino al primo byte mancante** del flusso.

### Timer

#### 23.11.1 Sequenza e riscontro

**Esempio: Telnet su TCP**

A SEQ = 42, ACK = 79, DATA = 'C' → B

Viene inviato il carattere "C" da A verso B, *ti mando il 42 aspetto il 79*

Nella direzione opposta si rimanda quanto ricevuto

A ← SEQ = 79, ACK = 43, DATA = 'C' B

ACK dell'host per ricevuta di "C", restituisco "C", *ti mando il 79, aspetto il 43*

A SEQ = 43, ACK = 80 → B

ACK dell'host sulla ricevuta di "C" di ritorno, *ti mando il 43 (o niente) e aspetto l'80*

**Pipeline** Con il **pipeline** il **mittente può inviare più segmenti senza attendere il riscontro**. Permette di aumentare la produttività

A SEQ = 42 → B

A SEQ = 43 → B

A ← ACK = 43 B

A ← ACK = 44 B

#### 23.11.2 Eventi lato mittente

**TCP riceve i dati dall'applicazione:**

- assegna un numero di sequenza
- avvia il timer (intervallo di scadenza – timeout)

### Timeout

- ritrasmette il segmento non riscontrato
- riavvia il timer

### ACK duplicato

Se il mittente **riceve tre ACK duplicati**, allora significa che il **segmento successivo a quello riscontrato è andato perso**.

**Ritrasmissione rapida** (fast retransmission) **prima** dello scadere del timer.

### Ritrasmissione dei segmenti

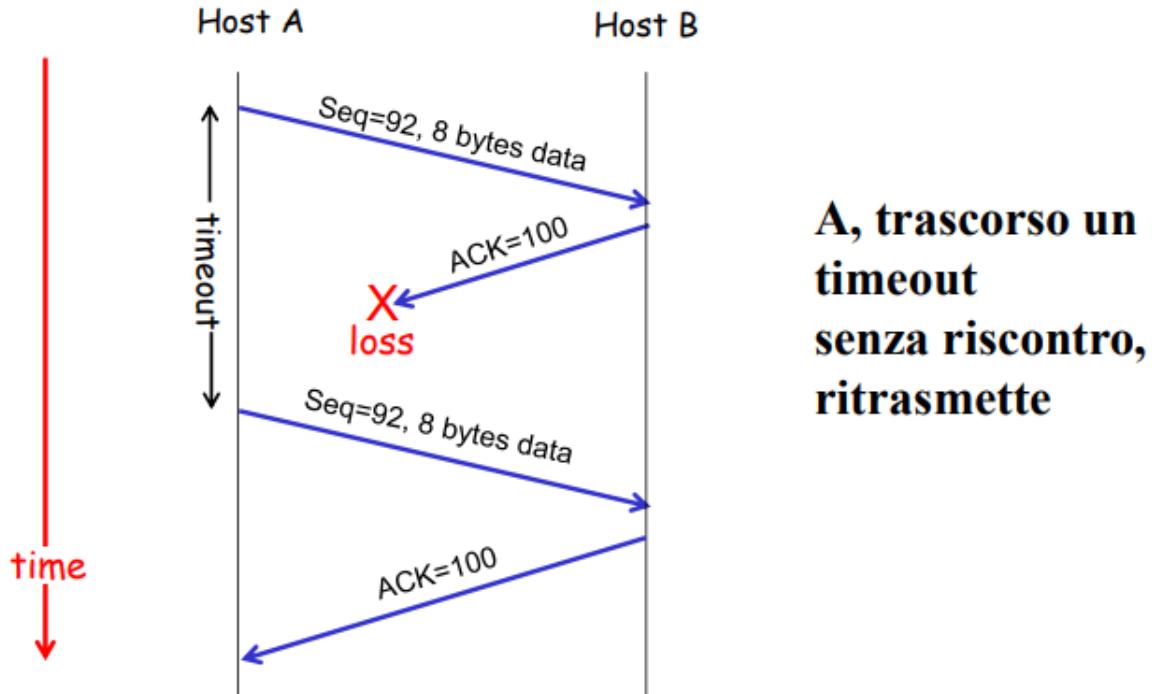
- se timeout
- se il mittente riceve tre ACK duplicati (fast retransmission)

### Segmenti fuori sequenza

I dati **possono arrivare fuori sequenza** ed essere temporaneamente memorizzati dall'entità TCP destinataria. Il TCP non dice come il destinatario **dove gestire i pacchetti fuori sequenza**, dipende dall'implementazione.

Nelle versioni più recenti si implementa **SACK**: i pacchetti fuori sequenza vengono **memorizzati**, il riscontro dei pacchetti fuori sequenza e duplicati è **invia in OPTIONS**.

Ritrasmissione dovuta a riscontro perso



### 23.11.3 Eventi lato destinatario

Tutti i segmenti inviati per trasmettere dati includono l'ACK. Se il destinatario non ha dati da inviare e **riceve un segmento "in ordine"**, allora **ritarda l'invio dell'ACK di 500ms** a meno che non riceva un nuovo segmento.

Se il destinatario **riceve un segmento atteso ma il precedente non è stato riscontrato** allora **invia immediatamente l'ACK**.

Se il destinatario riceve:

**segmento fuori sequenza**

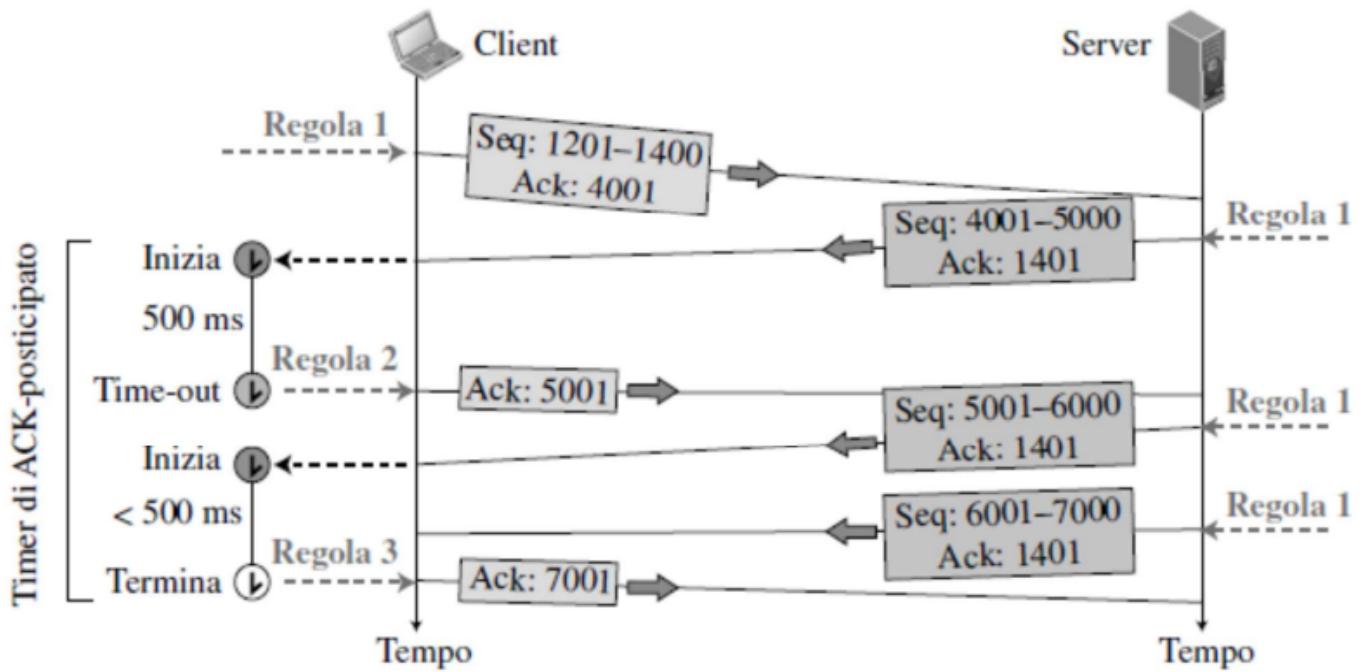
**segmento mancante ("buco" nella sequenza")**

**segmento duplicato**

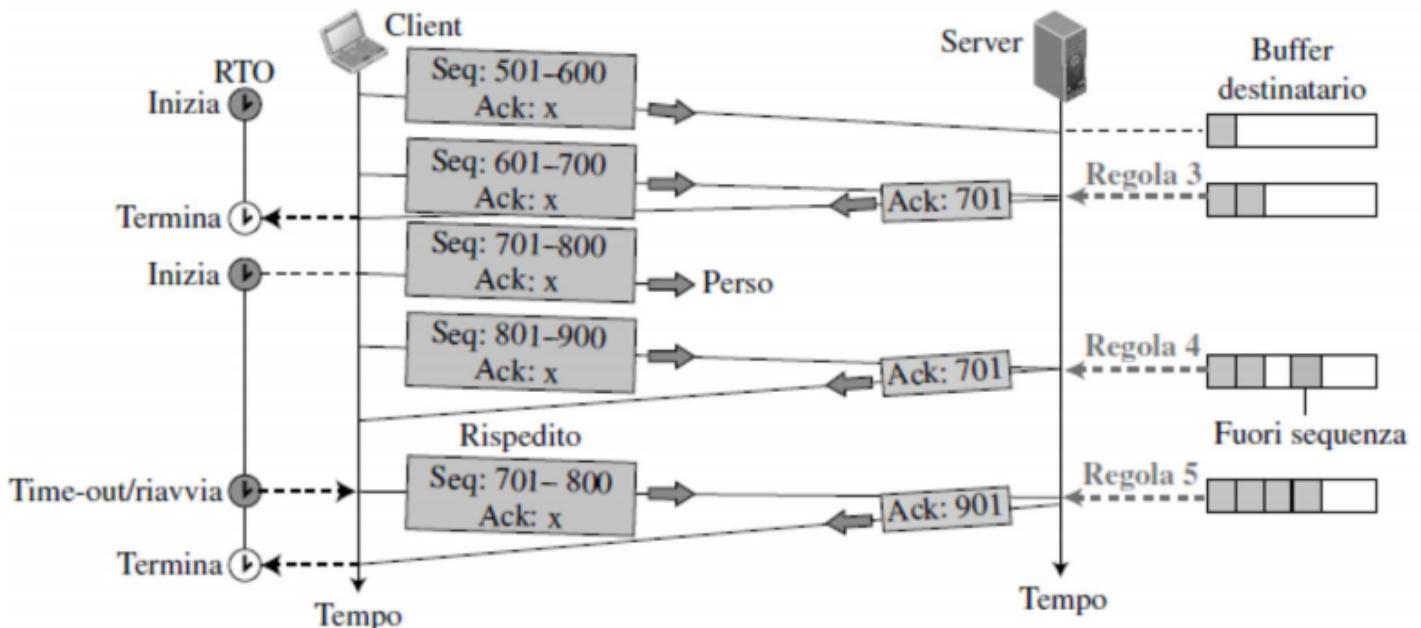
allora **invia immediatamente l'ACK** indicando il prossimo numero atteso.

#### 23.11.4 Esempi

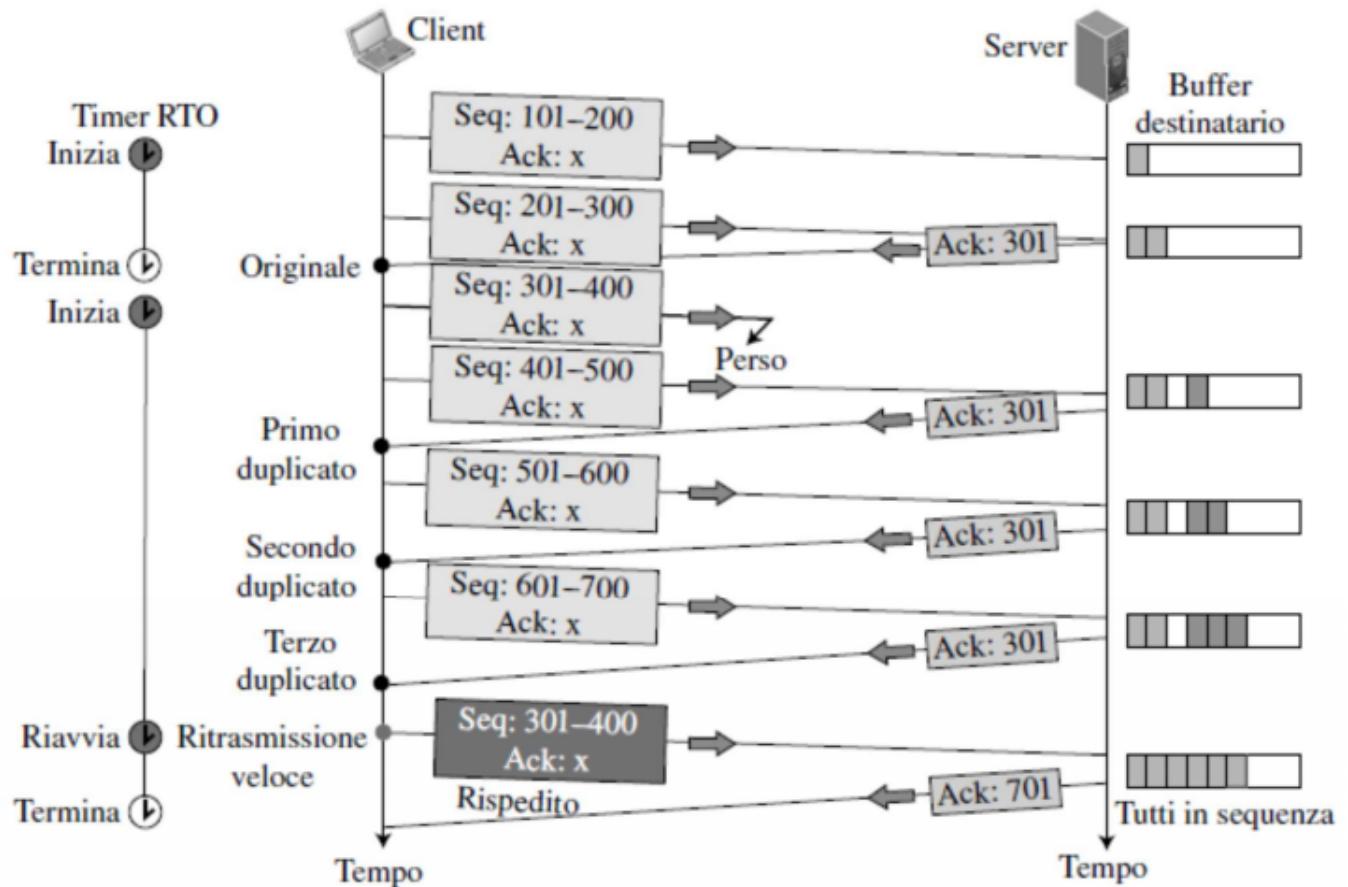
##### Operatività normale



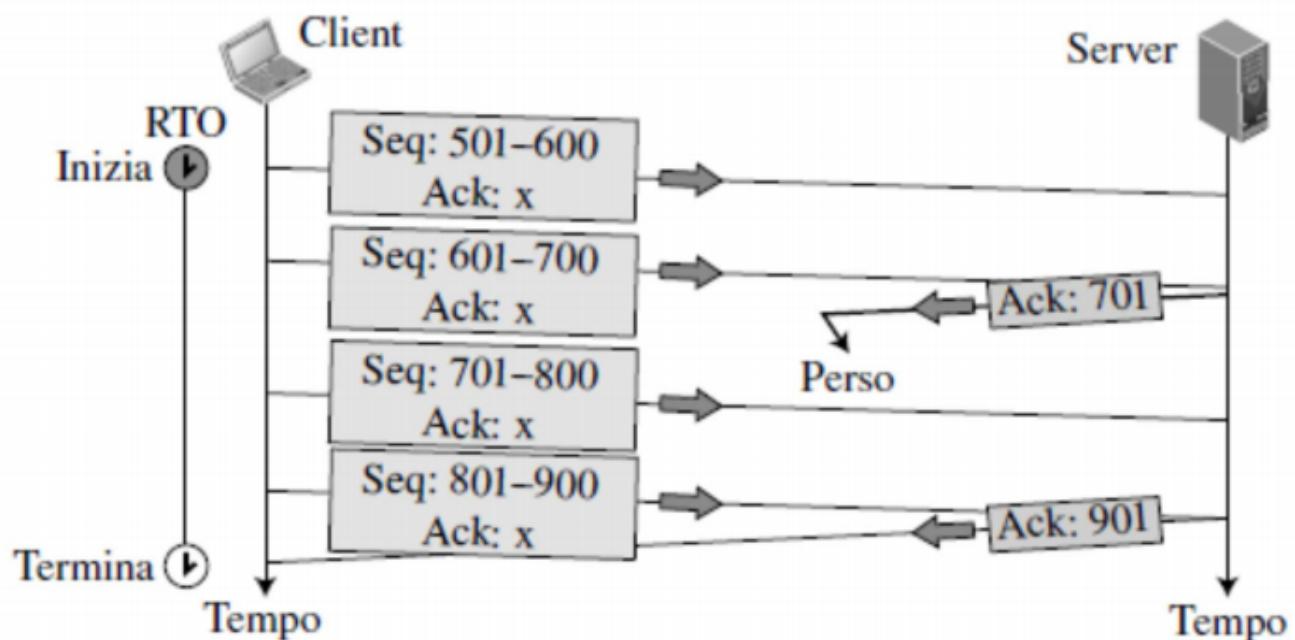
##### Segmento smarrito



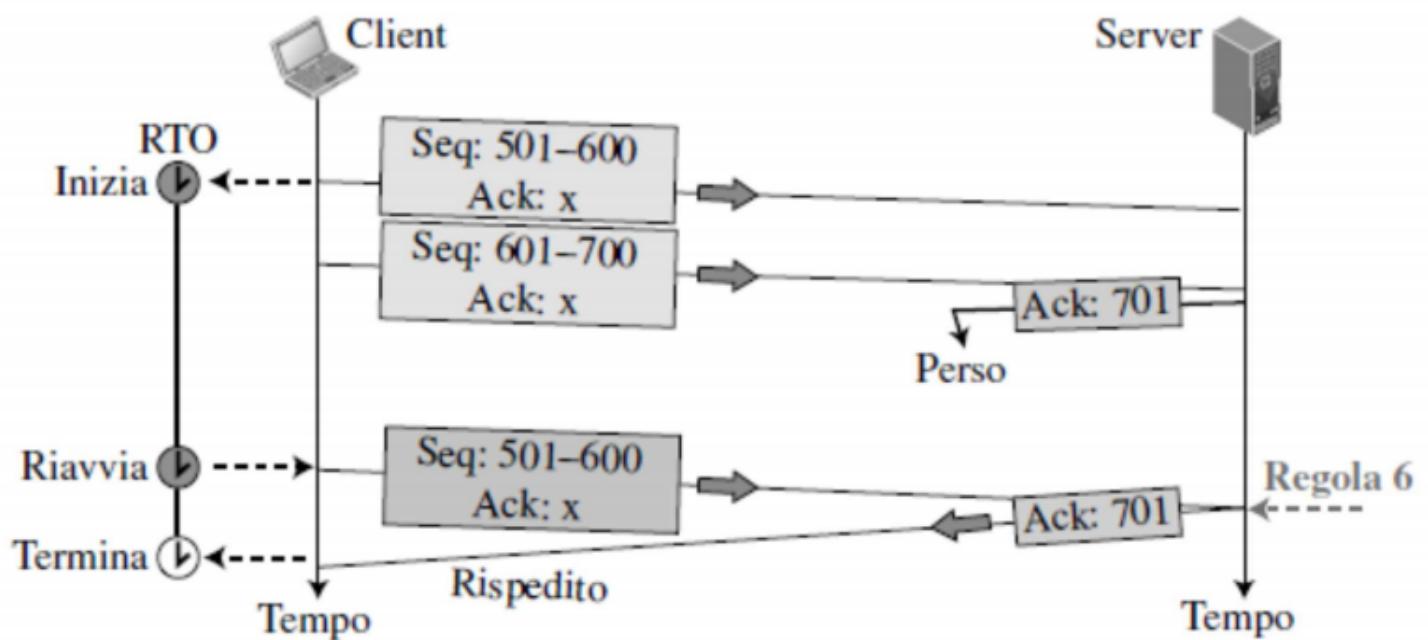
### Ritrasmissione veloce



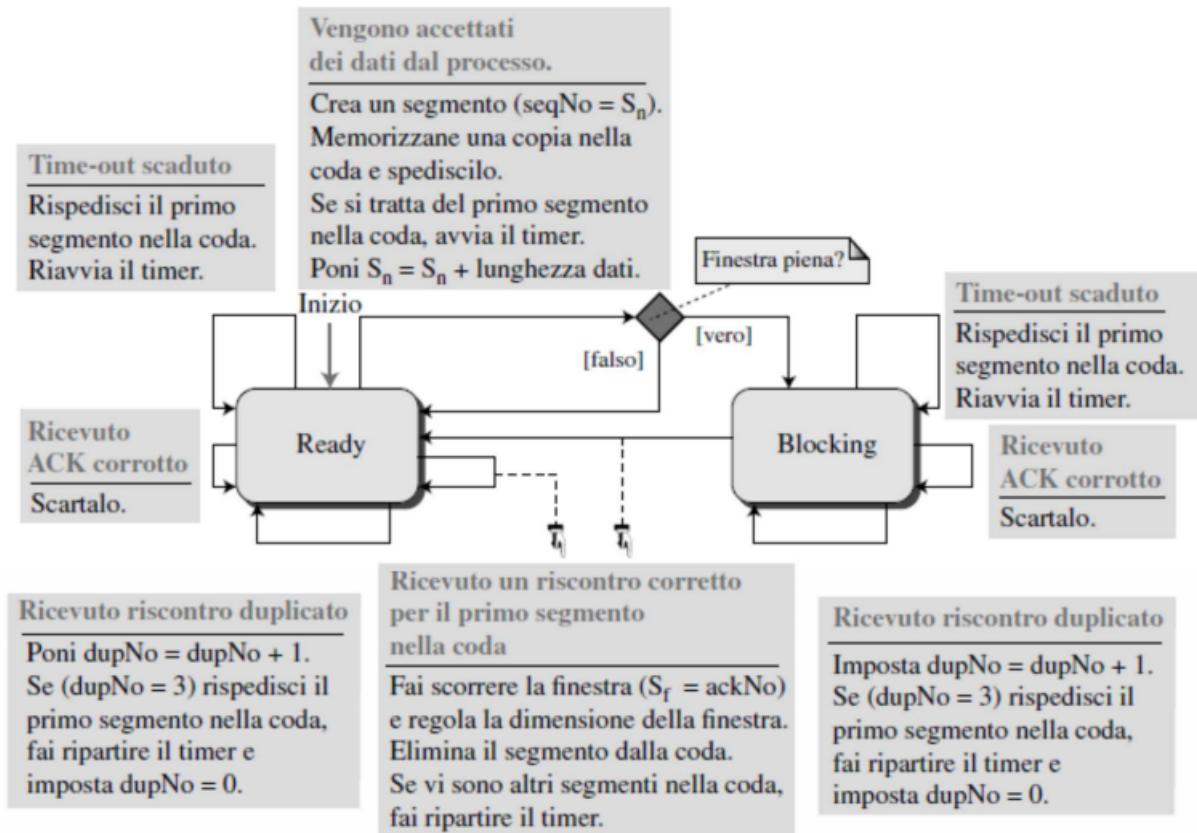
### Riscontro smarrito



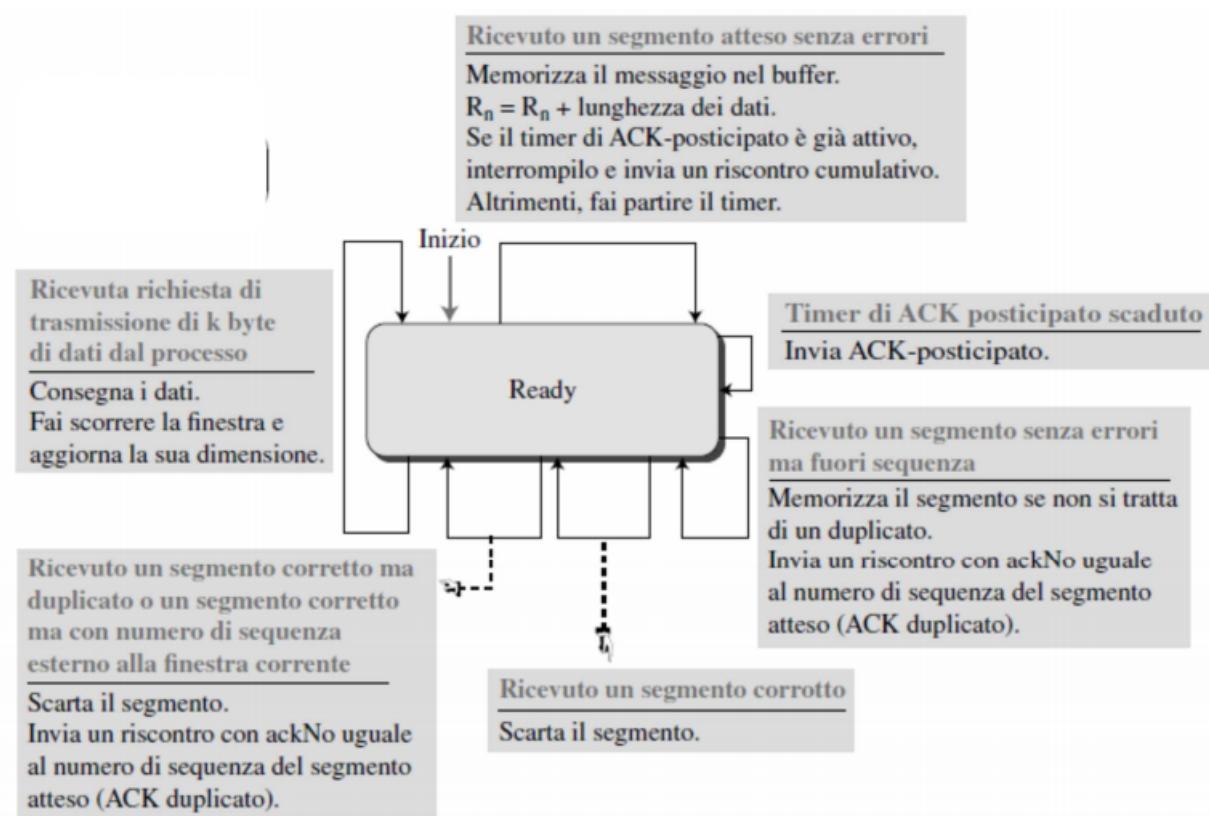
Riscontro smarrito corretto da ritrasmissione



### ASM semplificato per il mittente



### ASM semplificato per il destinatario



## 23.12 Calcolo del timeout

**RTO** Il Retransmission Timeout, o tempo di timeout, è **fondamentale** per il funzionamento del TCP. L'RTO deve essere maggiore dell'RTT (Round Trip Time)

**RTT** Il Round Trip Time **tempo trascorso da quando si invia un segmento a quando se ne riceve il riscontro.** Viene calcolato analizzando gli RTT dei segmenti **non ritrasmessi:** Sample RTT, stimato per un segmento trasmesso, non per ogni invio)

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

Poiché SampleRTT può fluttuare, si considera EstimatedRTT, cioè la **combinazione** dei precedenti valori di EstimatedRTT con il nuovo valore SampleRTT.

RFC 2988 – Il valore  $\alpha$  viene posto a  $\frac{1}{8}$ , in modo da rendere via via meno importanti gli RTT dei pacchetti più vecchi.

$$\text{EstimatedRTT} = 0.875 * \text{EstimatedRTT} + 0.125 * \text{SampleRTT}$$

**Variabilità** Oltre al valore RTT stimato, è necessaria anche una **stima della variabilità** di RTT, data dalla seguente formula che **stima di quanto SampleRTT si discosta da EstimatedRTT:**

$$\text{RTT}_{DEV} = (1 - \beta) \text{RTT}_{DEV} + \beta * |\text{RTT}_{SAMPLE} - \text{RTT}_{ESTIMATED}|$$

RFC2988 – Il valore di  $\beta$  viene posto a  $\frac{1}{4}$

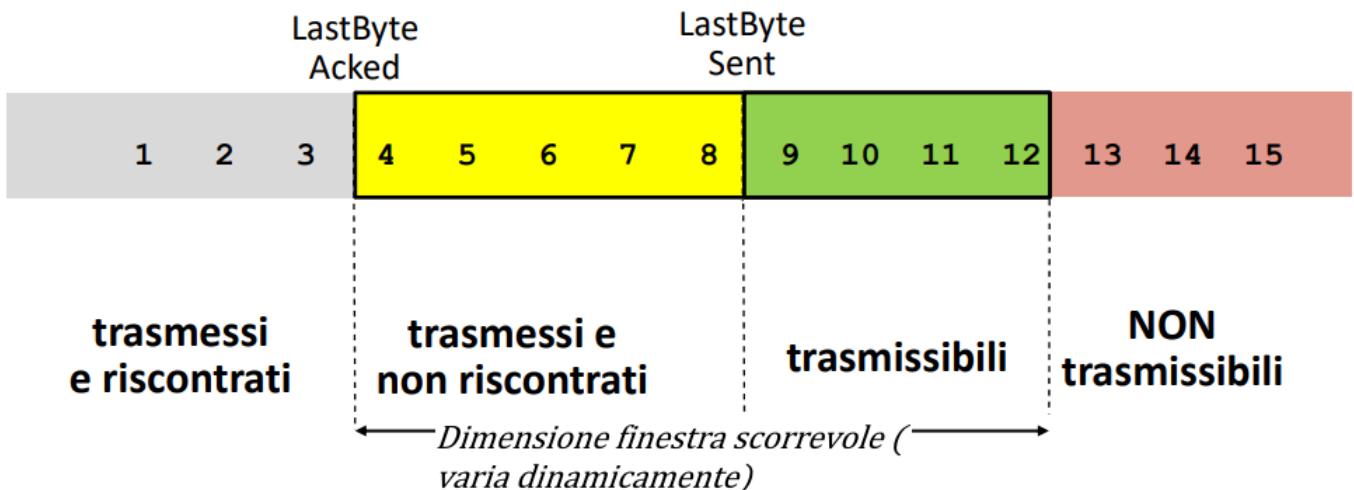
**Calcolo del timeout** Una volta ottenuti questi valori, il timeout viene normalmente calcolato come:

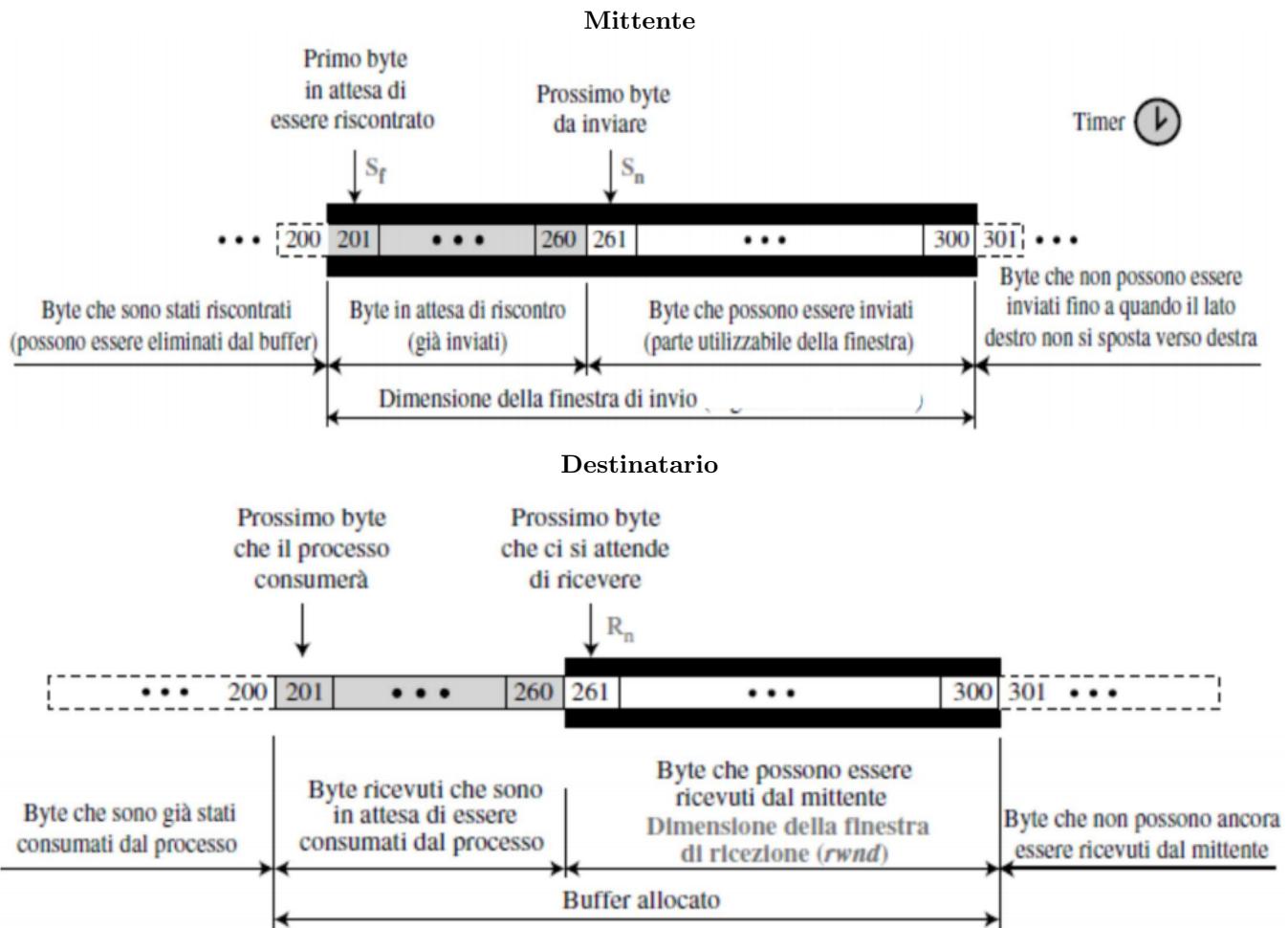
$$\text{RTO} = \text{RTT}_{ESTIMATED} + 4 * \text{RTT}_{DEV}$$

Inoltre in molte implementazioni, dopo un errore (es. ACK non ricevuto) **si raddoppia il timeout**. Questo è un primo meccanismo di controllo della congestione.

## 23.13 Finestra di trasmissione

**Sliding Window** Il controllo del flusso (flow control) del TCP si basa sulla **finestra di trasmissione**. La finestra è sovrapposta alla sequenza da trasmettere, viene **negoziata dinamicamente** e viene fatta avanzare alla ricezione di un ACK.

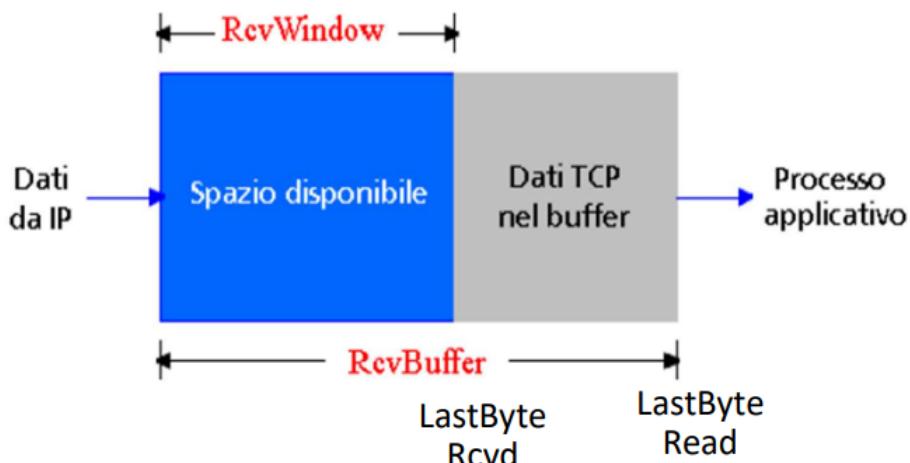




Ogni host impone buffer di invio e ricezione. Il **processo destinatario** legge i dati dal buffer di ricezione (**non necessariamente nell'istante in cui arrivano**)

Con **controllo di flusso** si intende la **capacità del mittente** di evitare la possibilità di saturare il buffer del **ricevitore**, modulando la propria frequenza di invio. Quindi si mette in relazione la **frequenza di invio** del mittente con la **frequenza di lettura** dell'applicazione ricevente.

**Receive Window** Il TCP implementa questa funzionalità con una **variabile** detta **receive window**, mantenuta dal **mittente**. Questa variabile **fornisce un'idea di quanto spazio è ancora a disposizione** nel buffer del **ricevitore**. Tale valore è comunicato nel **campo window** dell'header TCP.



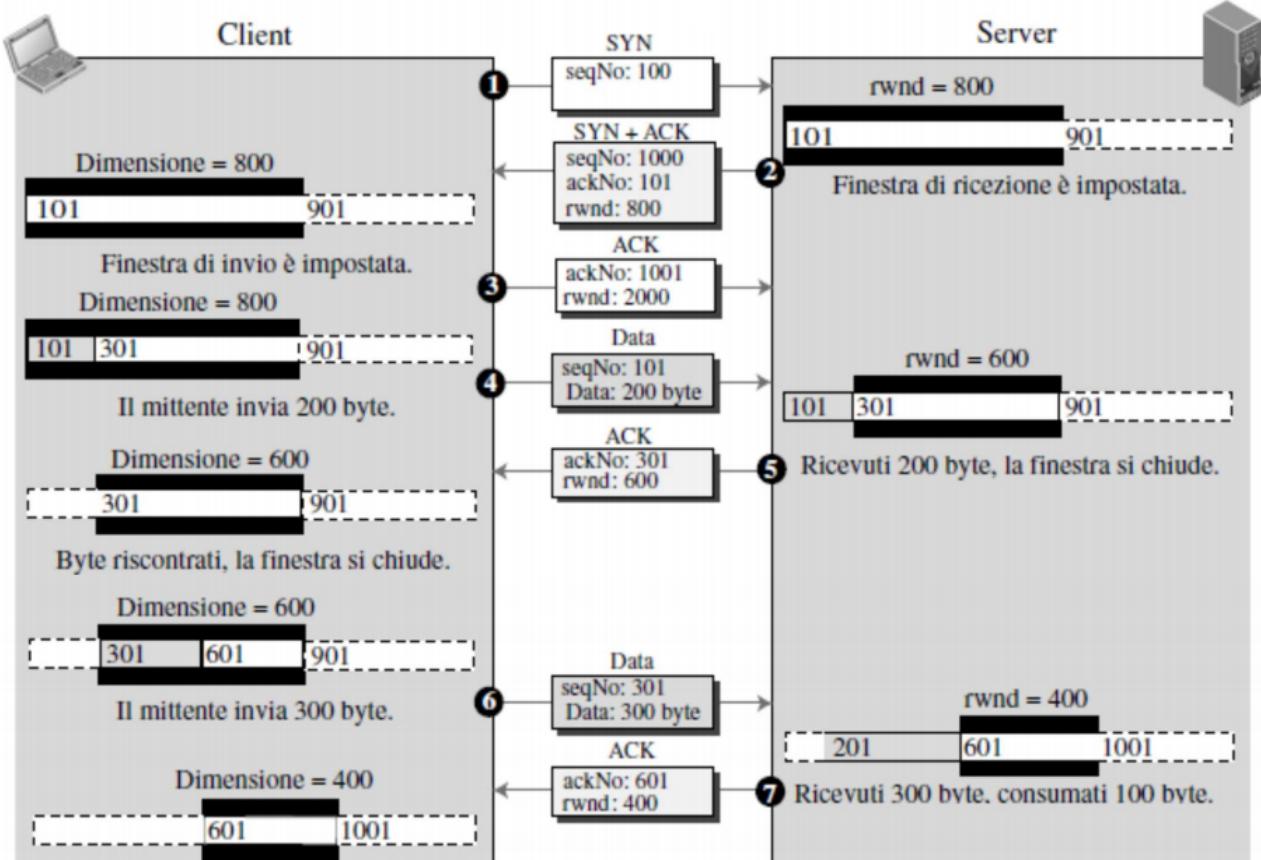
Lo spazio disponibile nel buffer del destinatario è:

$$\text{RcvWindow} = \text{RcvBuffer} - (\text{LastByteReceived} - \text{LastByteRead})$$

RcvWindow è dinamica, il destinatario comunica la dimensione di RcvWindow al mittente. Il mittente si assicura che:

$$\text{LastByteSent} - \text{LastByteAcked} < \text{RcvWindow}$$

Se RcvWindow = 0, il mittente manderà segmenti "sonda" di 1 byte per ricevere l'aggiornamento sulla dimensione di RcvWindow. Di seguito un esempio:



### 23.14 Controllo della congestione

RFC 2581

**Origine** Il fenomeno della congestione ha origine quando una o più delle sorgenti **tentano di richiedere più banda di quella disponibile sul percorso**. Troppe sorgenti che trasmettono troppi dati ad una velocità troppo alta, per cui la rete non può gestirli.

**Il traffico eccessivo può provocare:**

lunghi ritardi, accodamenti nei buffer dei router

perdita di pacchetti, overflow nei buffer dei router

**Controlli** Il protocollo TCP prevede il **controllo della congestione**, imponendo a ciascun mittente di **limitare la frequenza di invio di pacchetti sulla connessione**, in funzione della **congestione percepita**.

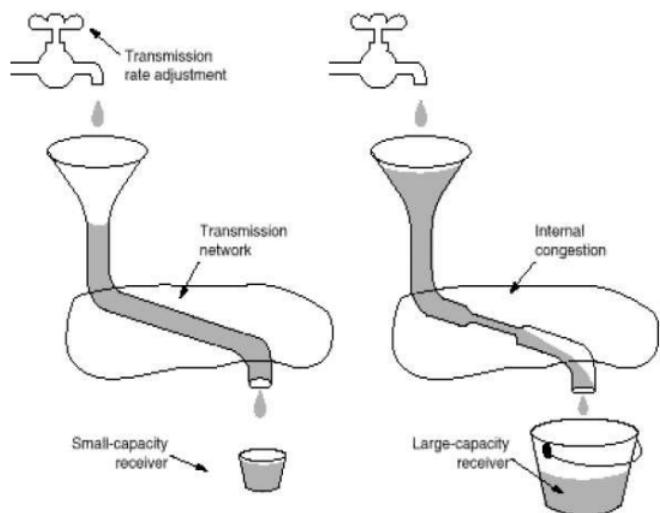
**Capacità del TCP di adattarsi alla velocità di rete:** se il TCP percepisce scarso traffico, aumenta la frequenza di invio, altrimenti la diminuisce.

**Controllo di congestione punto-punto:** nessun supporto esplicito della rete, la **congestione è dedotta dai sistemi terminali**

## Come gestire entrambi i tipi di congestione?

**Receiver window:** dipende dalla dimensione del buffer di ricezione

**Congestion window:** basata su una stima della capacità di rete



I byte trasmessi corrispondono alla **dimensione della finestra più piccola**:

$$\text{dimFinestraInvio} = \min(rWnd, cWnd)$$

<https://www.docenti.unina.it/downloadPub.do?tipoFile=md&id=90675>

### 23.14.1 Algoritmo per il controllo della congestione

**Algoritmo** L'algoritmo che il **mittente TCP** utilizza per **regolare la propria frequenza di invio** in funzione della congestione rilevata, è costituito da **tre passi**:

1. **Partenza, slow start**
2. **AIMD: incremento additivo e decremento moltiplicativo**
3. **Ripresa veloce, fast recovery**
4. **Reazione ai time-out**

cWnd impone un **vincolo** alla frequenza di immissione del traffico sulla rete, in base alla congestione percepita.

### 23.14.2 cWnd

Soltamente è misurata in termini di **Maximum Segment Size, MSS**.

1 MSS è la **quantità massima di dati trasportabili da un segmento**.

Viene **determinato in base alla MTU** (Maximum Transfer Unit), cioè alla lunghezza massima del payload del frame di collegamento inviabile dall'host mittente

MSS scelto in modo tale che **il segmento TCP, incapsulato dentro il pacchetto IP, stia dentro un singolo frame** di collegamento.

RTT (Round Trip Time) è il **tempo impiegato da un segmento per effettuare il percorso di andata e ritorno**.

### 23.14.3 AIMD

#### Additive Increase Multiplicative Decrease

Il TCP del mittente **aumenta proporzionalmente la propria finestra di congestione ad ogni ACK ricevuto**. Di quanto aumenta?

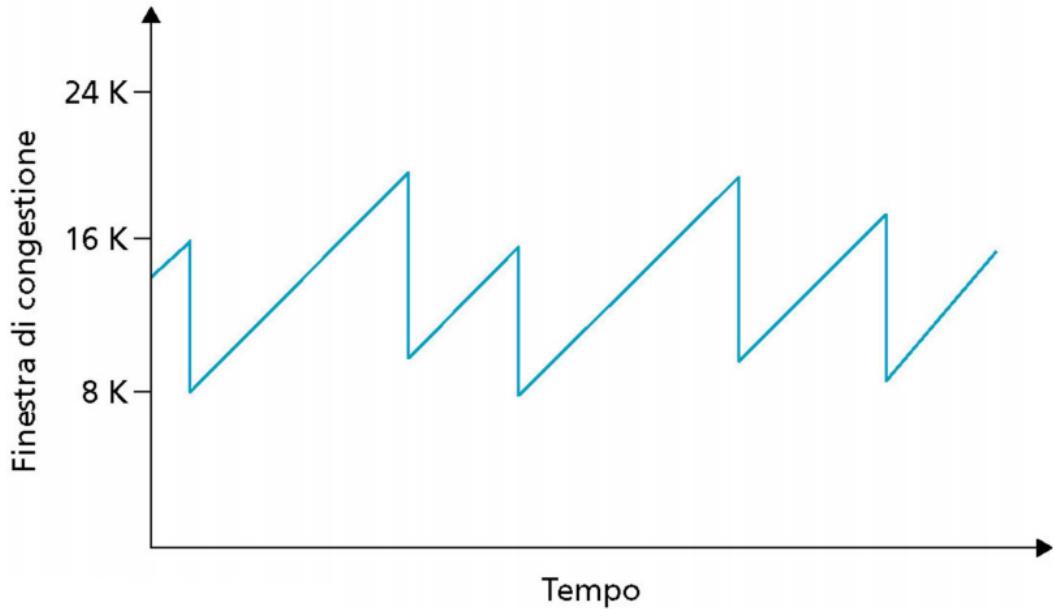
Ad ogni ACK la cWnd viene incrementata in modo che si abbia una **crescita pari ad 1 MSS per ogni RTT (congestion avoidance)**

Incremento di  $1\text{MSS} * (\text{MSS}/\text{cWnd})$

Ad esempio se  $\text{cWnd} = 4\text{MSS}$  allora l'incremento è di  $\text{MSS}/4$

Il TCP del mittente **dimezza la propria finestra di congestione ad ogni evento di perdita**, ad es. timeout o ACK duplicati.

Andamento della dimensione della cWnd nel tempo: andamento a "sawtooth"

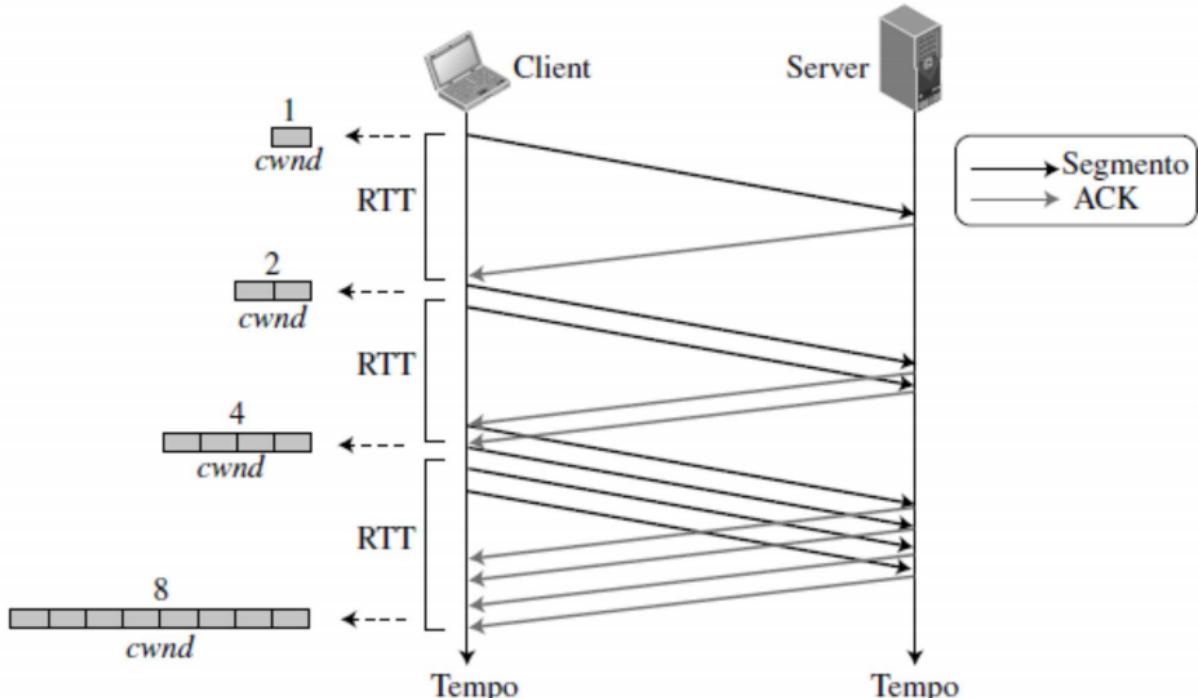


#### 23.14.4 Slow Start

All'inizio, la finestra di congestione cWnd è posta pari a 1 MSS: equivale a dire che la **frequenza di invio è pari a 1 MSS/RTT**.

**Esempio** Se  $MSS = 500\text{Byte}$  e  $RTT = 200\text{ms}$  si ha una frequenza di invio di circa  $20 \text{ Kb/s}$ . Se ho una banda da  $1 \text{ Mb/s}$  impiegherà molto tempo per sfruttarla con un incremento lineare.

cWnd viene incrementata di 1 MSS ad ogni ACK. L'effetto è che cWnd **raddoppia ad ogni RTT** avendo così una **crescita esponenziale** fino ad un errore, poi cWnd dimezza.



### 23.14.5 Politica Reno per il controllo della congestione

**Soglia** Viene definita una variabile "soglia", alla quale è assegnato un valore alto (ad esempio 64 Kb). Questa soglia determina quando termina la slow start ed inizia la congestion avoidance (AIMD).

cWnd < soglia: cWnd aumenta esponenzialmente (*slow start*)

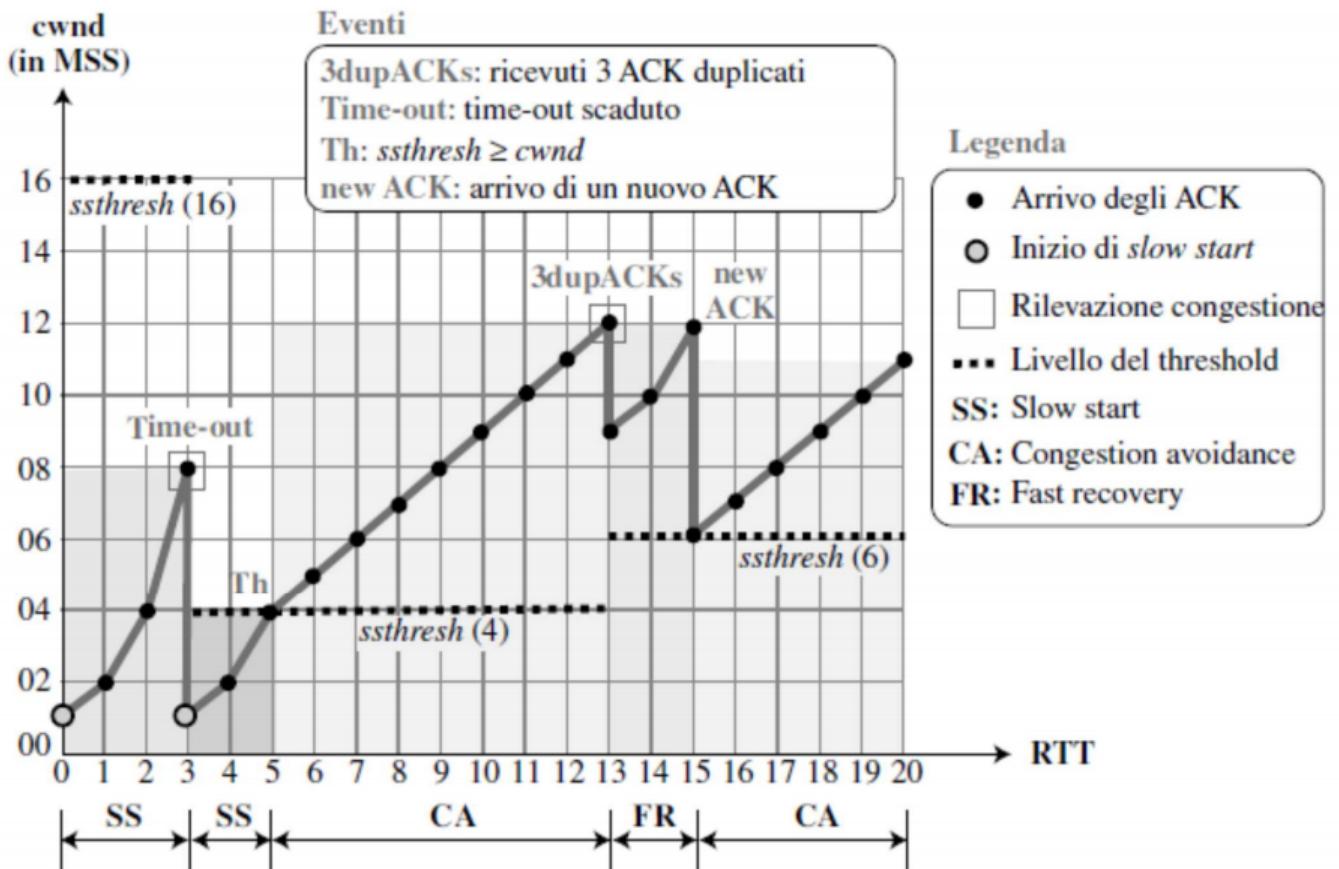
cWnd > soglia: cWnd aumenta linearmente (*AI*)

**Evento di perdita** Se ho 3 ACK duplicati pongo prima la soglia a metà di cWnd e poi  $cWnd = \text{soglia} + 3 \text{ MSS}$  (*fast recovery*)

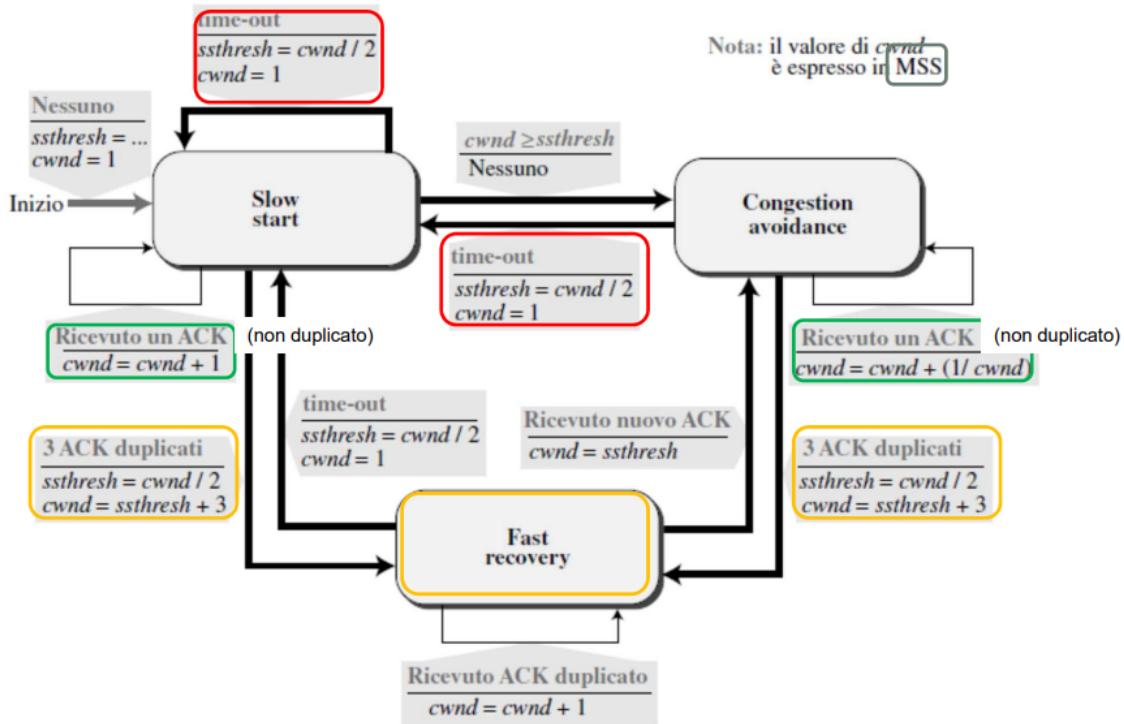
Se ho un ACK perso per timeout pongo la soglia a metà di cWnd e pongo  $cWnd = 1 \text{ MSS}$  (*slow start*)

**Nella fast recovery** Se avviene un timeout si va in **slow start**, se arriva un ACK non duplicato si va in **congestion avoidance**.

#### Esempio di controllo della congestione con TCP Reno

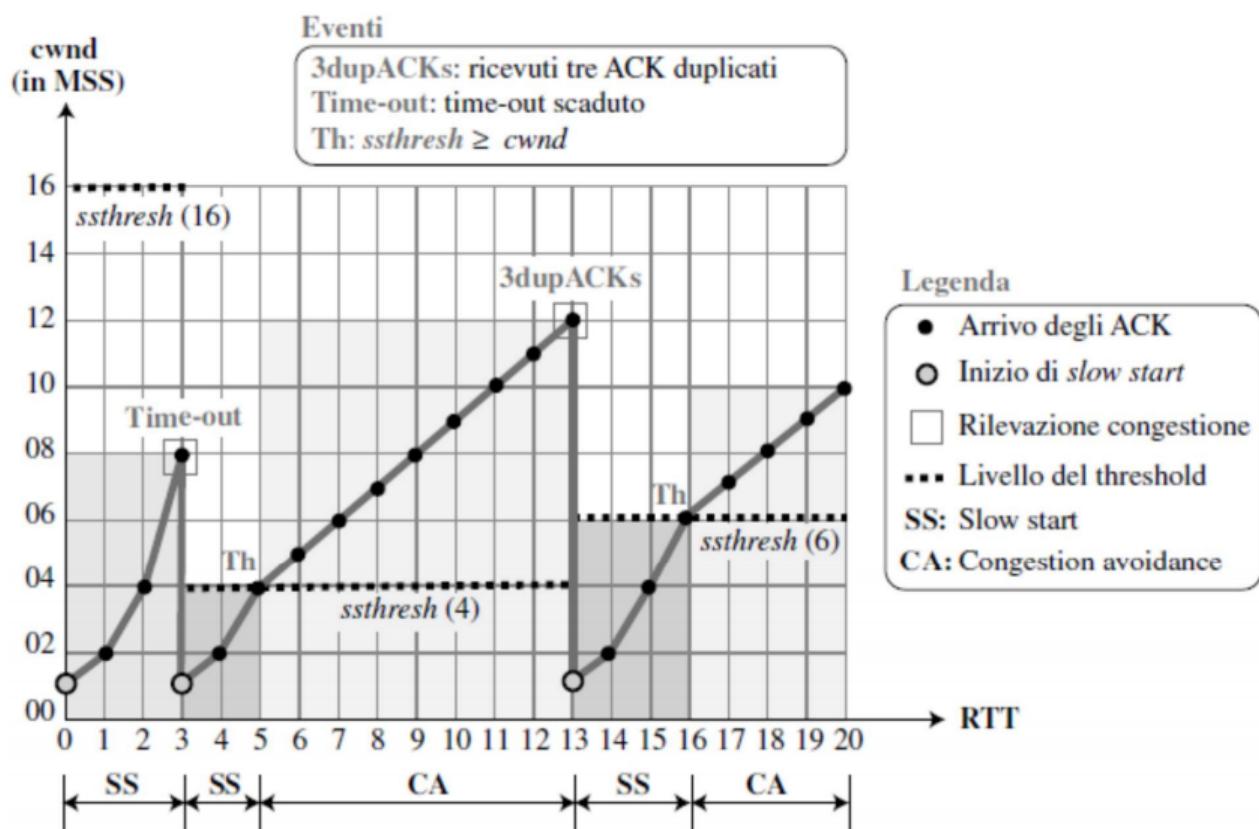


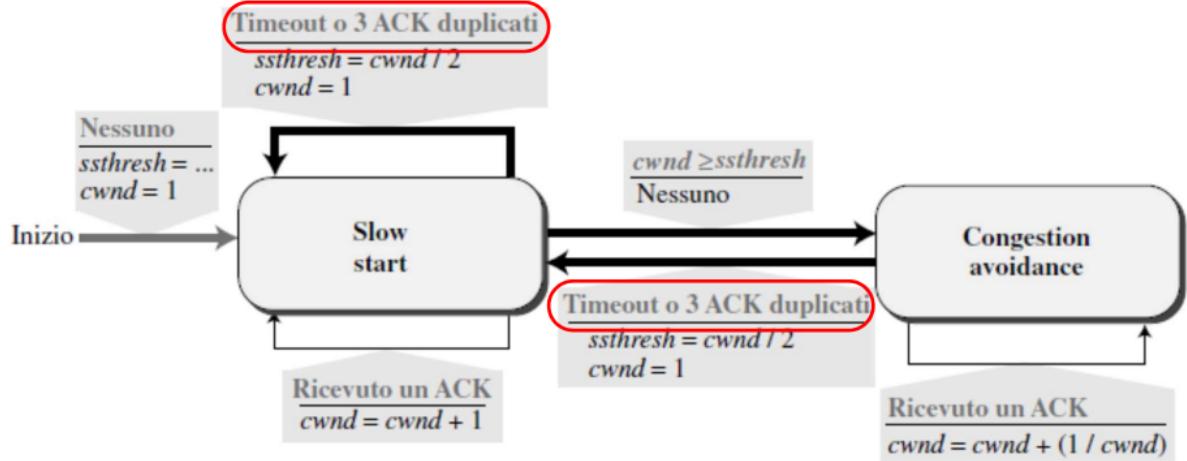
## TCP Reno



### 23.14.6 Politica Tahoe per il controllo della congestione

Versione precedente della politica Reno: timeout e 3 ACK duplicati sono trattati allo stesso modo.





### 23.15 Throughput

Indicando con  $W$  il **valore massimo di byte della finestra** (ovvero quando si verifica l'errore), il **TCP in regime stazionario** offre il seguente throughput medio:

$$\text{throughput} = \frac{0.75 * W}{RTT}$$

Calcolo macroscopico che ignora le fasi di slow start. Con  $W$  la dimensione della finestra alla perdita:

Quando la finestra è  $W$ , il throughput è  $\frac{W}{RTT}$

Appena dopo la perdita, la finestra è ridotta a  $\frac{W}{2}$  e il throughput a  $\frac{W}{2 * RTT}$

Throughput medio:  $0.75 * RTT$

### 23.16 Fairness

**Ipotesi**

**K** connessioni TCP esistono su un **unico link di capacità R bit/s**

Non ci sono altri protocolli esistenti sullo stesso link

**Risultato** Ognuna delle connessioni TCP tende a trasmettere a  $R/K$  bit/s.

### 23.17 Transmission Control Block

**Struttura dati** Poiché ogni connessione è distinta, bisogna **mantenere delle informazioni** su ogni connessione separatamente.

**TCB** Il TCP usa una struttura dati speciale per questo scopo, chiamata **Transmission Control Block** o TCB. Il TCB contiene tutte le informazioni a proposito della connessione, come **i due numeri di socket** che la identificano e **i puntatori ai buffer** dove vengono mantenute le informazioni in arrivo e in partenza.

Il TCB è anche usato per **implementare il meccanismo delle sliding windows**. Mantiene delle variabili che tengono traccia dei **numeri di byte ricevuti e riscontrati**, **byte ricevuti e non riscontrati**, **dimensione attuale della finestra** e così via.

Ovviamente, **ogni dispositivo mantiene il proprio TCB per la connessione**.

## 24 Esercizi

### 24.1 SMTP

**Testo** L'utente mickey@disney.com invia dal suo PC una email a donald@disney.com.  
Indicare la sequenza di comandi SMTP inviati e ricevuti dal PC di mickey@disney.com se:

1. Il mailserver disney.com non è raggiungibile
2. Il mailserver disney.com è raggiungibile

#### Soluzione

1. Il PC del mittente non può stabilire una connessione TCP con il mailserver, di conseguenza nessun messaggio SMTP può essere inviato o ricevuto
2. Il mittente stabilisce una connessione TCP con il mailserver, e avviene lo scambio dei seguenti comandi:

```
R: 220 service ready
I: HELO . .
R: 250 OK
I: MAIL FROM: mickey@disney.com
R: 250 OK
I: RCPT TO: donnald@disney.com
R: 250 OK
I: DATA
I: . .
I: . .
I: .
R: 250 OK
I: QUIT
R: 221 service closed
```

Notare che il corpo del messaggio termina sempre con <CRLF>.<CRLF>.

## 24.2 DNS

**Testo** Un host deve risolvere il nome simbolico `host.engineering.vanderbilt.edu`, il cui indirizzo IP non è noto al suo resolver (i.e. servizio DNS dell'host).

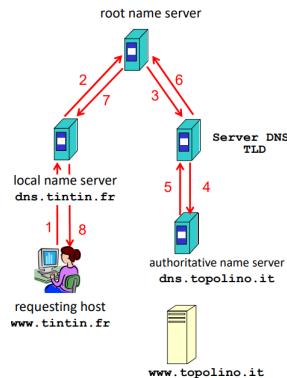
Supponendo che la gerarchia dei name server abbia 4 livelli, indicare – giustificando la risposta - il numero di messaggi DNS che, nel caso peggiore, circoleranno in Internet per risolvere tale nome simbolico, se:

1. Si utilizza ad ogni livello una risoluzione ricorsiva
2. Si utilizza ad ogni livello una risoluzione iterativa

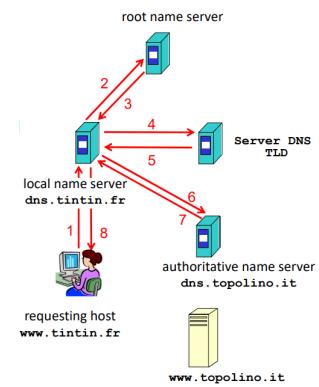
### Soluzione

1. Bisogna percorrere tutto l'albero dei nameserver, dal locale fino ad un root server, e poi da questo all'autoritative name server che, nel caso peggiore, è il nameserver locale di `engineering.vanderbilt.edu`, per poi tornare indietro.

In totale sono  $4 + 4$  messaggi + 2 dal resolver al nameserver locale del client e viceversa.



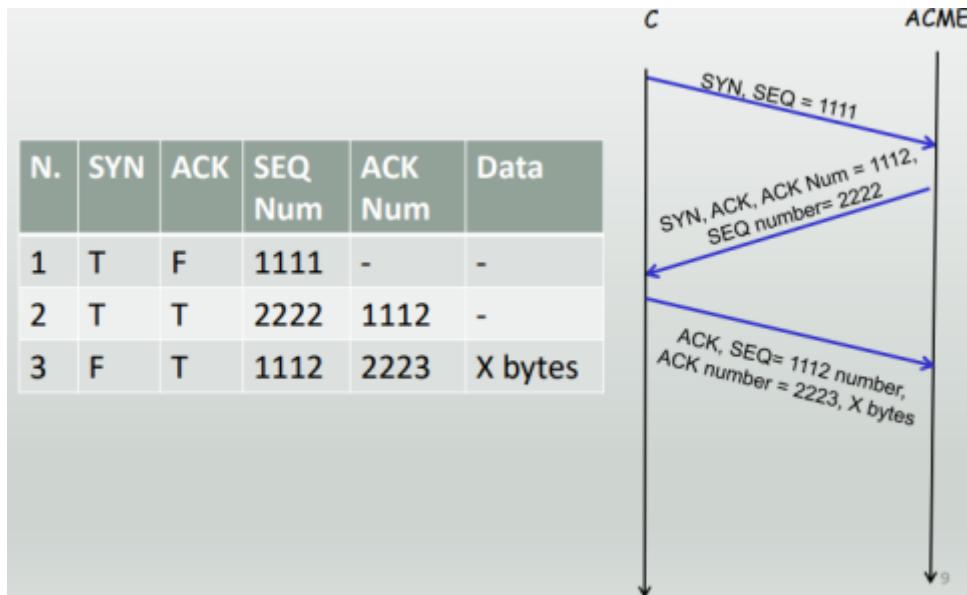
2. Analogamente i messaggi saranno sempre 10, perché i nameserver coinvolti sono 4, nel caso pessimo, più i 2 messaggi tra il nameserver locale del client e il resolver.



### 24.3 TCP

**Testo** Un client C chiede la pagina web <http://www.acme.com/home/products.html> al server B di [www.acme.com](http://www.acme.com) con una GET che è contenuta in un segmento TCP il cui payload (campo Dati) è lungo X byte. Indicare, giustificando la risposta, i valori dei campi sequence number, ACK number, e dei flags ACK e SYN e lunghezza del campo DATA, in ciascuno dei segmenti che C e B si scambiano per aprire la connessione nell'ipotesi che l'ACK finale dell'apertura della connessione sia inviato in piggybacking assieme alla GET. Si supponga che in B ed in C rWnd sia molto grande, che non scada alcun timeout, che non ci siano errori di trasmissione, che nessun segmento vada perduto, e che il numero di sequenza iniziale di C sia 1111 e quello di B sia 2222

#### Soluzione



**Piggybacking** L'invio dei primi dati e l'invio dell'ultimo ACK dell'handshake sono uniti.

**Testo** Un client C ha stabilito una connessione TCP con un server web S per scaricare una pagina web che consiste di tre oggetti. Al tempo t, subito dopo avere inviato la richiesta per il terzo oggetto, l'host di C invia a S un segmento con il flag FIN a true.

Indicare, giustificando la risposta, il tempo minimo necessario al TCP di C per chiudere definitivamente la connessione supponendo che:

1. La dimensione del terzo oggetto sia 1.5 MSS
2. RTT sia costantemente 700ms e che il Maximum Segment Lifetime sia 1100ms
3. Tutti i segmenti vengano ricevuti corretti ed in ordine, e che il valore di cwnd del TCP di S sia 1MSS quando esso riceve il FIN inviato dall'host di C. Trascurare i tempi di preparazione e di trasmissione dei segmenti e assumere che S abbia già a disposizione tutti i dati da inviare.

#### Soluzione $3RTT + 2MSL = 4300\text{ms}$

Il TCP di C riceve al tempo  $t + RTT$  il riscontro S1 del segmento FIN da lui inviato. Se S1 trasporta in piggybacking il primo MSS dei dati del terzo oggetto e il TCP di C invierà un riscontro C2 per tali dati, quindi S potrà inviare l'ultima porzione di dati e attenderne il riscontro, infine il TCP di S invierà un segmento con il flag FIN a true. Il TCP di C invierà un riscontro del FIN e considererà chiusa la connessione dopo avere atteso 2MSL, ovvero al tempo  $t + 3RTT + 2MSL = t + 4300 \text{ msec}$ .

**Testo** Si descriva il meccanismo di controllo di flusso del TCP

**Soluzione possibile** Si intende con controllo di flusso la capacità del mittente di evitare la possibilità di saturare il buffer del ricevitore. Infatti, a livello TCP ogni host imposta un buffer di invio e uno di ricezione. Il processo applicativo destinatario legge i dati dal buffer di ricezione (non necessariamente nell'istante in cui arrivano). Il controllo di flusso ha lo scopo di regolare la frequenza di invio del mittente in base alla frequenza di lettura dell'applicazione ricevente allo scopo di non saturare il buffer del ricevente. TCP implementa questa funzione tramite una variabile detta receive window mantenuta nel mittente: questa variabile fornisce un'idea di quanto spazio è ancora a disposizione nel buffer del ricevitore. Tale valore è comunicato nel campo window dell'header TCP dall'host destinatario.

Il valore di receive window  $RcvWindow = RcvBuffer - (LastByteReceived - LastByteRead)$ .

L'host destinatario comunica la dimensione di  $RcvWindow$  al mittente.

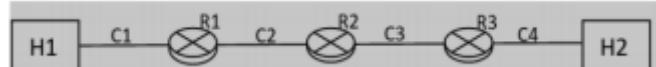
Il mittente si assicura che  $LastByteSent - LastByteAcked < RcvWindow$  pari ovvero alla quantità di dati trasmessi e non ancora riscontrati.

Nelle situazioni in cui il buffer risulta pieno ( $RcvWindow=0$ ), il mittente continua a mandare segmenti sonda di 1 byte per ricevere l'aggiornamento sulla dimensione di  $RcvWindow$ .

**Testo** Due host H1 e H2 comunicano tramite un canale che attraversa tre router R1, R2 e R3 e 4 link di capacità C1, C2, C3 e C4, rispettivamente come mostrato in figura. La comunicazione avviene tramite commutazione di pacchetto con trasmissione di tipo store and forward.

Assumendo che il ritardo di propagazione sia trascurabile, che i ritardi di accodamento nei router R1, R2 e R3 sia rispettivamente a1, a2 e a3, e che il ritardo di elaborazione nei tre router sia uguale a 1 ms, dire quanto tempo è necessario per la trasmissione da H1 a H2 di un pacchetto di dimensione L nel seguente caso:

- $L = 10 \text{ KBytes}$
- $C1 = C2 = C3 = C4 = 2 \text{ Mbps}$
- $a1 = a3 = a4 = 0.01 \text{ s}$



**Soluzione** Il ritardo introdotto da ogni router è dato dalla somma tra: ritardo di trasmissione, ritardo di propagazione, ritardo di accodamento e tempo di elaborazione. In particolare:

- il tempo di trasmissione di H1 è pari a  $L/C1 = 40 \text{ ms}$ .
- il tempo di trasmissione di R1 è pari a  $L/C2 = 40 \text{ ms}$ .
- il tempo di trasmissione di R2 è pari a  $L/C3 = 40 \text{ ms}$
- il tempo di trasmissione di R3 è pari a  $L/C4 = 40 \text{ ms}$

Quindi il ritardo introdotto dai router è:  $40 + 10 + 1 = 51 \text{ ms}$

Il ritardo dovuto alla trasmissione da H1 è pari a  $\frac{L}{R} = 40 \text{ ms}$ .

Quindi il tempo complessivo per la trasmissione del pacchetto da H1 a H2 è

$$51 * 3 + 40 = 193 \text{ ms}$$

**Testo** Tizio manda dal suo account mail tizio@libero.it un messaggio di posta elettronica con testo di 256 caratteri a Caio, di indirizzo caio@occupato.it.

1. Specificare il contenuto dei primi due messaggi TCP inviati dal mailserver di libero.it, ms.libero.it, per ricevere tale messaggio.
2. Specificare, per ciascun segmento, payload, numero di sequenza, ACK number, flags posti ad 1, porta origine e porta destinazione.

### Soluzione

1. Il primo segmento avrà il payload vuoto e il payload del secondo conterrà 220 `service ready`
2. Il primo segmento sarà **SYNACK**, e quindi: come già detto, payload vuoto, flags **SYN** e **ACK** ad 1, numero sequenza Z, ACK number Y, porta mittente 25, porta destinazione effimera. Il secondo segmento conterrà come payload 220 `service ready`, nessun flag ad 1, numero sequenza Z+1, ACK number Y, porta mittente 25, porta destinazione effimera.

**Testo** Dire in quali delle seguenti circostanze il TCP cambia la dimensione della finestra di congestione cWnd, e, nel caso, come viene ricalcolata.

Stato	Evento	Cambia la finestra?	Nuova dimensione della finestra
Slow Start	Ricezione di un ACK duplicato		
Slow Start	Scatta un timeout		
Congestion Avoidance	Scatta un timeout		

### Soluzione

Stato	Evento	Cambia la finestra?	Nuova dimensione della finestra
Slow Start	Ricezione di un ACK duplicato	NO	
Slow Start	Scatta un timeout	SI	CongWin = 1 MSS
Congestion Avoidance	Scatta un timeout	SI	CongWin = 1 MSS

**Testo** Si consideri il seguente scenario TCP in cui, per semplicità, non sono indicati i segmenti inviati o re-inviati dal sender TCP al receiver TCP, che si suppone essere quelli necessari per avere i riscontri descritti di seguito.

- Al tempo t0 il TCP di un host A ha una connessione già stabilita, per la quale ha 4 segmenti full sized in volo (inviai ma non riscontrati) e nessun nuovo dato da spedire, il primo byte dei segmenti in volo è il byte Y, ssthresh = 6.5 MSS, cwnd = 5 MSS. Inoltre, non ha ricevuto nessun riscontro duplicato.
- Tra il tempo t0 e il tempo t1 riceve 7 riscontri: i primi due non duplicati e con ACK number uguale a  $Y + 1$  MSS per il primo, e  $Y + 3$  MSS per il secondo. I seguenti 4 riscontri sono tutti duplicati, ed infine, al tempo t1, riceve un settimo riscontro con ACK number uguale a  $Y + 4$  MSS.

Si supponga che non scatti alcun timeout tra t0 e t1. Indicare, per ciascun riscontro ricevuto, lo stato del TCP e i valori di ssthresh e cWnd, giustificando la risposta.

### Soluzione

n.1	Riscontro	Cong Window	Threshold	Stato
1	$Y+1$ MSS	5+1 MSS	6,5 MSS	SS
2	$Y+3$ MSS	(5+1)+1 MSS*	6,5 MSS	CA
3-4	$Y+3$	7 MSS	6,5 MSS	CA
5	$Y+3$	3,5+3 MSS	3,5 MSS	FR
6	$Y+3$	7,5 MSS	3,5 MSS	FR
7	$Y+4$	3,5 MSS	3,5 MSS	CA

**Testo** Descrivere in modo dettagliato e mediante uno pseudocodice le azioni svolte da un destinatario TCP per realizzare il controllo di flusso. Si assuma che ogni segmento ricevuto dal destinatario contenga anche lo pseudoheader (lunghezza segmento TCP). Inoltre, si supponga che la chiusura della connessione venga fatta dal mittente e che i segmenti di chiusura non contengano dati in piggybacking. Non occorre realizzare la parte iniziale delle azioni svolte (quindi le inizializzazioni delle variabili e l'apertura della connessione). Infine, si supponga che si invii un riscontro appropriato per ogni segmento ricevuto, e che il destinatario non debba inviare dati al mittente. Inoltre, per semplicità, si specifichino solamente i campi dell'header del riscontro relativi al controllo del flusso e al riscontro. Non occorre realizzare le interazioni con il livello applicativo (processo che legge dal buffer), ma solamente quelle con il mittente TCP. Si hanno a disposizione le seguenti procedure:

`receive(segm)`, per ricevere il segmento `segm` dal livello di rete

`OK(segm)`, restituisce true solo se `segm` è corretto

`nuovo(segm.x)`, vettore di booleani che vale true se dal campo `x` di `segm` si deduce che i dati contenuti in `segm` non sono doppioni (specificare `x` nella soluzione)

`insert(segm.y, finestra)`, per inserire `segm.y` nel buffer di ricezione nella posizione corretta

`calcolacknum(segm)`, restituisce il numero di riscontro per il riscontro associato a `segm`

`send(risp)`, per inviare `risp` al mittente

Descrivere il contenuto o la funzionalità delle variabili e delle altre funzioni/procedure eventualmente utilizzate.

## Soluzione

```
finito = false;
while(!finito) {
    receive(segm);
    finito = segm.FIN;
    if (!finito) { //controllare l'inizio della chiusura della connessione
        if (OK(segm)) { //se il segmento non è corrotto
            if (nuovo(segm.seqnum)) { //se il segmento contiene nuovi dati
                insert(segm.dati, finestra); //inserisco i dati nel buffer
                rWnd = rWnd - (segm.lTotTCP - segm.HLEN); //nuovo valore di rWnd
                risposta.ACK = true; //flag settato a true
                risposta.ACKnum = calcolachecksum(segm);
                risposta.rWnd = rWnd;
            }
            send(risposta); //Invia o il nuovo ACK o l'ultimo ACK inviato
        }
    }
}
```

**Testo** Discutere l'affermazione "poichè FTP e HTTP sono protocolli adatti a trasferire file, possono essere usati indifferentemente".

**Soluzione possibile** Un esempio di risposta potrebbe seguire questo schema:

- Descrivere brevemente l'obiettivo di HTTP e FTP
- Entrambi usano TCP, elencare le differenze ai fini del trasferimento file

#### FTP

Connessione controllo, persistente, inizializzata dal client, porta 21. Comandi in formato ASCII a 7 bit

Connessione dati, inizializzata da server, porta 20, non persistente

Stateful

Offre funzionalità aggiuntive rispetto a HTTP per la gestione di file e directory (list, retr, put).

#### HTTP

Unica connessione per dati + comandi

Interazione stateless