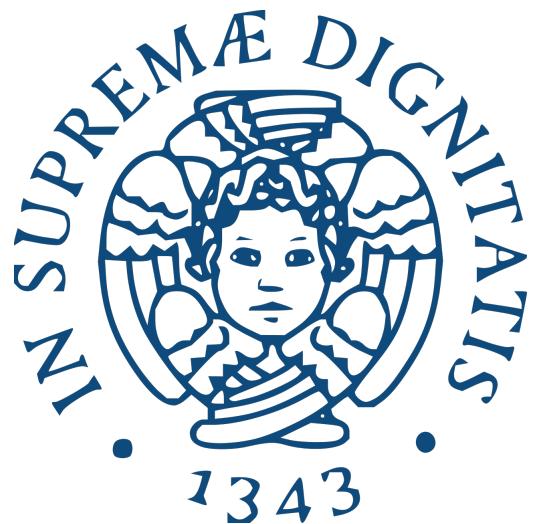


# Reti e Laboratorio III

Giuseppe Di Palma

November 14, 2023



Appunti basati sulle lezioni e dispense delle professoresse Federica  
Paganelli<sup>1</sup> e Laura Ricci<sup>2</sup>

---

<sup>1</sup><http://pages.di.unipi.it/paganelli/>

<sup>2</sup><https://pages.di.unipi.it/ricci/>

## Contenuti

<b>0 Introduzione</b>	<b>3</b>
<b>1 Commutazione e Ritardi</b>	<b>3</b>
1.1 Commutazione di Circuito . . . . .	4
1.2 Commutazione di Pacchetto . . . . .	5
1.3 Ritardi . . . . .	6
<b>2 Modelli Stratificati</b>	<b>7</b>
2.1 Perché e come stratificare . . . . .	7
2.2 Open System Interconnection . . . . .	7
2.3 Gerarchia degli strati ISO/OSI . . . . .	9
2.4 Stack TCP/IP . . . . .	12
<b>3 Lo strato Applicativo</b>	<b>14</b>
3.1 Protocolli TCP e UDP . . . . .	15
3.2 URI, URL E URN . . . . .	15
3.3 HTTP . . . . .	16
3.3.1 Formato messaggi HTTP . . . . .	18
3.3.2 Metodi . . . . .	20
3.3.3 Caching e Cookie . . . . .	22
3.4 TELNET . . . . .	23
3.5 Email . . . . .	24
3.5.1 SMTP . . . . .	26
3.5.2 Estensioni multimediali . . . . .	27
3.5.3 Accesso alla posta . . . . .	28
3.6 DNS . . . . .	29
3.6.1 Spazio dei nomi . . . . .	30
3.6.2 Gerarchia dei NameServer . . . . .	31
3.6.3 Tipologia di Query . . . . .	32
3.6.4 Caching e aggiornamento Record . . . . .	33
3.6.5 Messaggi DNS . . . . .	34
3.6.6 DNS Hijacking . . . . .	35
3.7 FTP . . . . .	36
<b>4 Lo Strato di Trasporto</b>	<b>38</b>
4.1 TCP . . . . .	41
4.1.1 Formato Segmento . . . . .	42
4.1.2 Gestione della Connessione . . . . .	44
4.1.3 Affidabilità e Controlli . . . . .	48
4.1.4 Throughput ed Equità . . . . .	59
4.2 UDP . . . . .	60
4.2.1 Vulnerabilità di UDP . . . . .	61

## 0 Introduzione

**Definizione** (Rete). Un'interconnessione di dispositivi in grado di scambiarsi informazioni, quali sistemi terminali (host), router, switch e modem

**Definizione** (Router). Dispositivi che interconnettono reti.

**Definizione** (Switch). Dispositivi che collegano fra loro più host a livello locale

**Tipologie di reti** Esistono varie tipologie di reti

- **LAN**: Local Area Network, sono reti di piccole dimensioni (al più qualche km). Conneggono principalmente host, stampanti e workstation tra loro.
- **WAN**: Wide Area Network, è una rete il cui compito è di interconnettere LAN o singoli host separati da distanze geografiche.
- **MAN**: Metropolitan Area Network, rete di computer che collega i computer all'interno di un'area metropolitana, più grande di una LAN ma più piccola di una WAN.

**Network of networks** Gli host si collegano ad internet tramite Internet Service Provider (ISP) i quali devono a loro volta essere connessi tra loro. La risultante rete di reti è molto complessa.

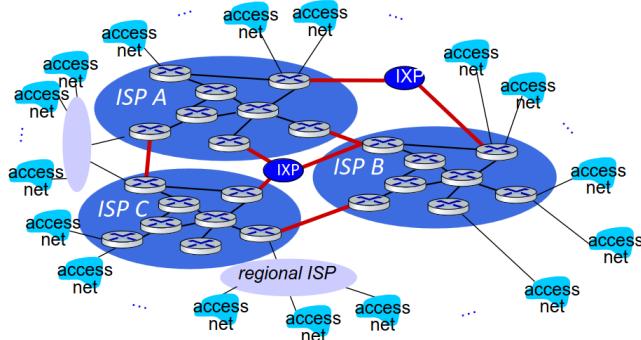


Figura 1: Struttura della rete Internet

## 1 Commutazione e Ritardi

**Definizione** (Commutazione). Modalità con cui viene determinato il percorso sorgente-destinazione e vengono dedicate ad esso le risorse della rete.

Esistono due meccanismi:

- **Commutazione di Circuito**
- **Commutazione di Pacchetto**

**Metriche** Data una comunicazione tra due host, indipendentemente dal tipo di commutazione utilizzata, è possibile utilizzare delle metriche comuni per l'analisi delle prestazioni:

- **Bandwidth:** larghezza dell'intervallo di frequenze utilizzato dal sistema trasmittivo (si misura in Hz).
- **Transmission Rate:** quantità di dati (bits) che possono essere trasmessi per unità di tempo su un certo collegamento (dipende dal bandwidth ma anche dal mezzo trasmittivo, rumore, ecc...).
- **Throughput:** quantità di dati che possono essere trasmessi con successo dalla sorgente alla destinazione in un certo intervallo di tempo al netto di perdite sulla rete (duplicazioni, protocolli, ecc...). In un percorso da una sorgente a una destinazione un pacchetto può passare attraverso numerosi link, perciò il throughput dell'intero percorso è dato dal percorso intermedio con throughput minore.

NB: Throughput < Transmission Rate

## 1.1 Commutazione di Circuito

Nella commutazione di circuito si instaura un cammino dedicato tra i due dispositivi che vogliono comunicare. Il percorso viene stabilito all'inizio della comunicazione (**Setup**) e vengono dedicate risorse alla comunicazione (canale logico o circuito) in modo esclusivo. Le risorse allocate sono garantite per tutta la durata della comunicazione, indipendentemente dall'utilizzo effettivo.

**Canale logico** Per quanto riguarda l'assegnazione di un canale di comunicazione logico esistono due principali metodi:

- **FDM:** Frequency Division Multiplexing, il canale di comunicazione viene suddiviso in bande di frequenze ognuna delle quali viene assegnata in modo esclusivo ad una certa connessione.
- **TDM:** Time Division Multiplexing, il tempo viene suddiviso in slot di tempo. Ogni comunicazione ha uno o più slot periodici assegnati nei quali può trasmettere alla velocità massima del canale.

**Vantaggi:**

- Performance garantite.
- Tecnologie di switching efficienti.

**Svantaggi:**

- Necessaria una fase di instaurazione della comunicazione.
- le risorse rimangono inattive se non utilizzate (non c'è condivisione).

## 1.2 Commutazione di Pacchetto

Nella commutazione di pacchetto il flusso di dati punto-punto viene suddiviso in pacchetti. Ogni pacchetto è instradato singolarmente e indipendentemente dagli altri pacchetti della stessa comunicazione (possono seguire lo stesso percorso o percorsi diversi in base alle necessità).

I commutatori (es. router) devono ricevere integralmente i pacchetti prima di poterne continuare l'instradamento. Quando ci sono più pacchetti in ingresso ripetto a quanto il canale può supportare, questi vengono messi in coda in dei buffer (se il buffer è pieno i pacchetti vengono persi).

**Vantaggi:**

- Risorse trasmissive usate solo se necessario.
- Fase di setup e segnalazione della connessione non richiesta.

**Svantaggi:**

- Tecnologie di inoltro poco efficienti (calcolo del percorso indipendente per ogni pacchetto).
- Ritardi variabili nel percorso end-to-end (jitter).
- protocolli necessari per un trasferimento dati affidabile, controllo della congestione.

### 1.3 Ritardi

**Definizione** (Latenza). Tempo richiesto affinché un messaggio arrivi a destinazione dal momento in cui il primo bit parte dalla sorgente.

Il valore della latenza in una rete a commutazione di pacchetto è determinato da 4 tipologie di ritardo:

- **Ritardo di elaborazione**, dovuto ai controlli di errore sui bit e dal calcolo del percorso di uscita del pacchetto.
- **Ritardo di accodamento**, dovuta all'attesa dei pacchetti nel buffer la quale dipende dal tipo e dall'intensità del traffico.
- **Ritardo di trasmissione**: tempo impiegato per trasmettere un pacchetto sul canale.

Dipende da due valori

- $R$ , rate di trasmissione sul canale (in bps).
- $L$ , lunghezza del pacchetto (in bit).

Si calcola come  $L/R$ .

- **Ritardo di propagazione**, tempo impiegato da 1 bit per essere propagato da un nodo all'altro

Dipende anch'esso da due valori

- $d$ , lunghezza del collegamento fisico (in metri).
- $s$ , velocità di propagazione del mezzo.

Si calcola come  $d/s$ .

La latenza è quindi la somma di tutti questi 4 ritardi. Nella pratica però i ritardi di elaborazione e accodamento vengono trascurati e la latenza end-to-end è data dalla somma dei ritardi di propagazione e trasmissione di tutti i collegamenti intermedi.

**Prodotto Rate-Ritardo** Con prodotto **rate-ritardo** indichiamo numero massimo di bit che un link può contenere ad un certo istante. Rappresentiamo il link come un tubo la cui sezione rappresenta il rate e la lunghezza il ritardo. Il volume di questo tubo fornisce il rapporto rate-ritardo

## 2 Modelli Stratificati

### 2.1 Perché e come stratificare

Nei sistemi di comunicazione non si utilizza un unico protocollo infatti, stratificare il problema permette di:

- scomporre il problema in sottoproblemi più semplici da trattare; il singolo strato è più semplice del sistema nel suo complesso.
- semplificare la progettazione, implementazione e manutenzione del software.
- rendere i livelli indipendenti: è possibile modificare l'implementazione di uno strato senza dover cambiare gli altri, a patto che l'interfaccia non cambi.

La separazione dei livelli non può ovviamente essere fatta in modo superficiale. Avere troppi livelli infatti rende il sistema poco scalabile, d'altro canto averne pochi rende i vari livelli complessi da realizzare.

Il processo di stratificazione avviene seguendo due principi:

- **Separation of Concern**: separazione degli interessi e delle responsabilità, fare ciò che compete, delegando ad altri tutto ciò che è delegabile.
- **Information Hiding**: nascondere tutte le informazioni che non sono indispensabili per il committente per definire compiutamente l'operazione.

Per garantire questi due principi inoltre la comunicazione tra livelli adiacenti deve essere ridotta al minimo indispensabile e ogni strato deve svolgere una sola e ben definita funzione.

### 2.2 Open System Interconnection

**Sistemi chiusi** Negli anni '60 nascono le prime reti di calcolatori come ARPANET<sup>3</sup>, SNA (IBM)<sup>4</sup> e DNA (Digital) che utilizzano architetture di rete a strati. Ognuna di queste reti però utilizzava protocolli proprietari incapaci di operare in ambienti condivisi perché non in grado di interpretare i segnali provenienti dall'esterno.

---

<sup>3</sup><https://it.wikipedia.org/wiki/ARPANET>

<sup>4</sup>[https://it.wikipedia.org/wiki/Systems\\_Network\\_Architecture](https://it.wikipedia.org/wiki/Systems_Network_Architecture)

**Sistemi aperti** L'obiettivo principale di questa tipologia di sistemi è quello di realizzare una rete di calcolatori in cui qualsiasi terminale comunica con un qualunque fornitore di servizi mediante una rete. Per poter rendere possibile questo obiettivo è necessario stabilire delle regole comuni, degli **standard**.

**Definizione.** Un protocollo è detto **Aperto** se:

- i dettagli sono disponibili pubblicamente.
- i cambiamenti sono gestiti da un'organizzazione la cui partecipazione è aperta al pubblico.

**Definizione.** Un sistema che implementa protocolli aperti è un **Sistema Aperto** (Open System)

L'International Organization for Standards (ISO) ha specificato uno standard per l'interconnessione di sistemi aperti denominato Reference Model Open System Interconnection (**OSI RM**).

**Definizione** (Strato). È un modulo interamente definito attraverso i servizi, protocolli e le interfacce che lo caratterizzano.

**Definizione** (Servizio). Servizi che uno strato fornisce ad uno strato sovrastante attraverso primitive di servizio.

**Definizione** (Interfaccia). Insieme di regole che governano il formato e il significato delle unità di dati (es. messaggi, segmenti o pacchetti) che vengono scambiati tra due strati adiacenti della stessa entità.

**Definizione** (Protocollo). Insieme di regole che definiscono il formato e l'ordine dei messaggi inviati e ricevuti tra entità omologhe della rete e le azioni che vengono fatte per la trasmissione e ricezione dei messaggi; in modo:

- **Efficace:** Un sistema che riesce a raggiungere lo scopo prefissato con la maggior frequenza possibile.
- **Efficiente:** Un sistema che riesce a raggiungere lo scopo prefissato con il minor sforzo possibile.

**Cosa deve fare un protocollo** In un protocollo vanno specificati:

- La sintassi del messaggio: che campi contiene e in quale formato.
- La semantica del messaggio: come interpretare i vari campi e il messaggio stesso.
- Le azioni da intraprendere dopo la ricezione di un messaggio.

## 2.3 Gerarchia degli strati ISO/OSI

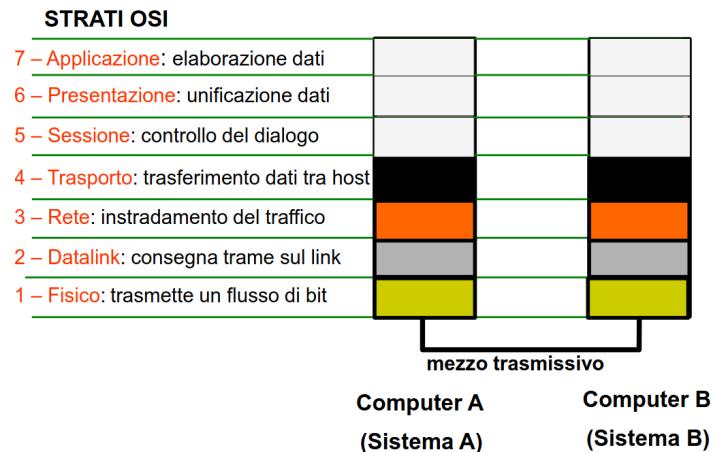


Figura 2: Strati definiti nel modello ISO/OSI

- **Livello Fisico:**

- Comprende tutte le funzioni (procedure meccaniche ed elettroniche) che permettono una connessione a livello fisico.
- Si occupa della trasmissione dei bit attraverso il mezzo trasmissivo e delle caratteristiche di cavi e connettori.

- **Livello di Collegamento:**

- Definisce le regole per inviare e ricevere informazioni tra due sistemi in comunicazione.
- Si occupa di formare i dati da inviare attraverso il livello fisico, incapsulando i dati in un pacchetto provvisto di header (intestazione) e tail (coda), chiamato frame.

- **Livello di Rete:**

- Deve far giungere i “pacchetti” a destinazione.
- Si occupa dell’istradamento (“routing”) dei pacchetti, cioè di determinare la sequenza di collegamenti punto-punto necessari per trasmettere un pacchetto da un nodo generico della rete a un altro.

- **Livello di Trasporto:**

- Questo strato fornisce un servizio di trasferimento dati end-to-end.
- Si occupa di instaurare, mantenere e terminare una connessione.
- Può offrire funzionalità per frammentare e riassemblare i dati, rilevare e correggere gli errori, controllare il flusso dei dati.

- **Livello di Sessione:** assembla il dialogo tra nodi in unità logiche.

- **Livello di Presentazione:** adatta la sintassi dei dati di ciascuna applicazione alla sintassi richiesta dalla sessione.

- **Livello di Applicazione:** protocolli a supporto di applicazioni distribuite.

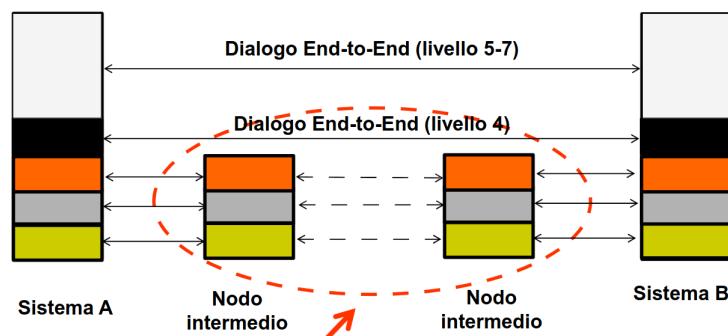


Figura 3: Collegamento tra End-System

**Modalità di Servizio** Esistono due principali metodologie di funzionamento per la comunicazione:

- **Connection-Oriented:**

- Associazione logica tra due o più sistemi al fine di trasferire dati.

- Gestione della connessione:
  - \* Instaurazione della connessione.
  - \* Trasferimento dati.
  - \* Chiusura della connessione

- **Connection-Less:** I dati vengono trasferiti senza l'instaurazione di una connessione.

**Flusso dell'informazione** Dal punto di vista delle reti, le informazioni hanno tutte origine dal livello Applicativo. L'informazione discende i vari livelli fino alla trasmissione sul canale fisico. Ogni livello aggiunge all'informazione del livello superiore una propria sezione informativa (o più) denominata header, che contiene informazioni riguardanti esclusivamente quel livello.

Per i dati ricevuti si segue il cammino inverso.

**Incapsulamento** Il processo di incapsulamento, ovvero quello nel quale ogni strato aggiunge dell'informazione a quella già presente, è un processo reversibile che garantisce l'estrazione durante la risalita delle informazioni dal livello di rete al livello applicativo (o fino ad un livello inferiore se ci troviamo in un nodo intermedio).

In questo processo abbiamo:

- **Header:** Contiene informazioni relative a quel livello.
- **Payload:** Dati provenienti dal livello superiore.
- **Tail:** Generalmente utilizzato per l'individuazione e la correzione degli errori.

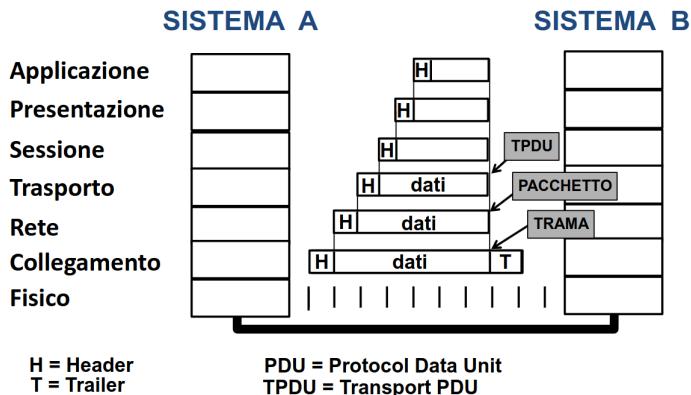


Figura 4: Processo di incapsulamento

## 2.4 Stack TCP/IP

TCP/IP è una famiglia di protocolli attualmente utilizzata in Internet. Si tratta di una gerarchia di protocolli, ciascuno dei quali fornisce funzionalità specifiche.

Definita in origine in termini di quattro livelli software soprastanti a un livello hardware, la pila TCP/IP è oggi intesa come composta di cinque livelli.

- **Applicazione:** supporta le applicazioni di rete, collegamento logico end-to-end; scambio di messaggi tra due processi (ftp, smtp, http).
- **Trasporto:** trasferimento dati end-to-end da un host sorgente all'host destinatario (Tcp, Udp).
- **Rete:** instradamento dei datagrammi dalla sorgente alla destinazione (Ip, ICMP).
- **Link:** trasferimento dati in frame attraverso il collegamento tra elementi di rete vicini (Point-to-Point Protocol, Ethernet, ...).
- **Fisico:** trasferimenti dei bit di un frame sul mezzo trasmittivo.

**Differenze tra ISO/OSI e TCP/IP** Nella pratica in Internet viene utilizzato lo stack protocollare TCP/IP. Questo accade perché ISO/OSI, a differenza di TCP/IP, fornisce una specifica generale, difficile da implementare e poco efficiente. ISO/OSI però viene utilizzato ancora come modello di riferimento.

### 3 Lo strato Applicativo

**Applicazioni** Le applicazioni di rete sono formate da processi distribuiti, su vari host. Ogni host può eseguire uno o più processi contemporaneamente. Questi processi comunicano tra loro mediante lo scambio di messaggi.

I livelli applicazione nei due lati della comunicazione agiscono come se esistesse un collegamento diretto su cui inviare questi messaggi.

I protocolli del livello applicativo definiscono:

- Il tipo dei messaggi (es: di richiesta e di risposta).
- La sintassi dei vari tipi di messaggio (i campi del messaggio).
- La semantica dei campi (significato).
- Le regole per determinare quando e come un processo invia messaggi o risponde ai messaggi.

**Paradigmi del livello applicativo** Il modo in cui gli host devono comportarsi quando utilizzano una certa applicazione di rete dipende dal tipo di paradigma che questa applicazione utilizza:

- **Client-Server:** Un numero limitato di host, detti **Server**, offrono servizi e sono sempre in attesa di richieste; al contrario, i restanti host, detti **Client**, inviano richieste ai server per ricevere servizi.
- **Peer-to-Peer:** Tutti gli host che utilizzano l'applicazione sono allo "pari" tra loro, tutti chiedono e ricevono servizi da tutti gli altri.
- **Misto**

**Definizione (API).** **Application Programming Interface:** insieme di regole che un programmatore deve rispettare per utilizzare delle risorse.

**Interfaccia Socket** L'interfaccia socket (l'API di internet per eccellenza) è quell'interfaccia che si frappone tra il livello di applicazione e il livello di trasporto. Questa interfaccia è messa a disposizione dal sistema operativo che implementa i 4 livelli inferiori dello stack protocollare TCP/IP. La socket è una struttura dati formata da:

- **Indirizzo IP:** identificativo della macchina con cui vogliamo comunicare (32bit per indirizzi ipv4, 128bit per indirizzi ipv6);
- **Porta:** numero a 16 bit che permette di identificare il processo sulla macchina con il quale si vuole comunicare.

### 3.1 Protocolli TCP e UDP

Nel livello di trasporto della pila di protocolli TCP/IP i due protocolli principali sono:

- **TCP** (Transmission Control Protocol): Protocollo connection-oriented, richiede una fase di setup tra client e server, che garantisce un trasporto affidabile dei dati. Fornisce inoltre meccanismi di controllo del flusso (il mittente non inonda il destinatario di dati) e della congestione (il mittente viene "rallentato" in caso di congestione della rete).
- **UDP** (User Datagram Protocol): Protocollo connection-less, non necessita di una fase di setup. Non garantisce trasporto affidabile e non presenta controlli di alcun tipo su flusso e congestione.

NB: entrambi non forniscono garanzie di timing e banda minima.

La scelta del protocollo da utilizzare dipende dalle esigenze dell'applicazione:

- **Throughput**: quanto è importante avere un certo livello di throughput per l'applicazione?
- **Perdita dei dati**: il 100% dei dati trasferiti devono necessariamente arrivare a destinazione?
- **Sensibilità ai ritardi**: l'applicazione è di tipo real-time?

### 3.2 URI, URL E URN

**URI** Uniform Resource Identifier, è una forma generale per identificare una risorsa presente in rete:

- **Uniform**: uniformità della sintassi dell'identificatore, anche con meccanismi diversi di accesso.
- **Resource**: qualsiasi cosa abbia un'identità (documento, servizio, immagine ...).
- **Identifier**: informazioni che permettono di distinguere un oggetto dagli altri.

Esistono due principali tipologie di URI:

- **URL**: (Uniform Resource Locator) sottoinsieme di URI che identifica le risorse attraverso il loro meccanismo di accesso.
- **URN**: (Uniform Resource Name) sottoinsieme di URI che devono rimanere globalmente unici e persistenti anche quando la risorsa cessa di esistere e diventa non disponibile.

**URL** Schema di una URL:

<scheme>://<user>:<password>@<host>:<port>/<path>

- <user> e <password> opzionale, in generale deprecato;
- <scheme> indica il protocollo di accesso alla risorsa;
- <host> nome di dominio di un host o indirizzo ip (in notazione decimali puntata);
- <port> numero di porta del server;
- <path> contiene dati specifici per l'host (o scheme) e identifica la risorsa nel contesto di quello schema e host.

A loro volta le URL possono essere di due tipologie differenti:

- **URL assoluta**: identifica una risorsa indipendentemente dal contesto in cui è usata;
- **URL relativa**: informazioni per identificare una risorsa in relazione ad un'altra URL (è priva dello schema e della authority).

Le URL relative non vengono utilizzate in rete, ma vengono interpretate dal browser in relazione al documento di partenza.

### 3.3 HTTP

**HyperText Transfer Protocol** Protocollo di tipo richiesta/risposta nel quale un client inizia la connessione, inviando al server una **request**, e ricevendo da esso una **response**. HTTP viene detto **stateless** in quanto ogni coppia richiesta/risposta è indipendente da tutte le altre. Per funzionare utilizza il protocollo di trasporto TCP con il quale viene instaurata una connessione tra client e server.

**Definizione** (Connessione). Un circuito logico, al livello di trasporto, stabilito tra due programmi applicativi per comunicare tra loro.

**Definizione** (Connessione non persistente, da RFC 1945). Viene stabilita una connessione TCP separata per recuperare ciascuna URL.

**Definizione** (Connessione persistente, da RFC 2616). Se non diversamente indicato, il client può assumere che il server manterrà una connessione persistente.

- Lo standard specifica un meccanismo con cui client e server possono indicare la chiusura della connessione TCP (Connection header field).
- Dopo la chiusura, il client non deve più inviare richieste su quella connessione.

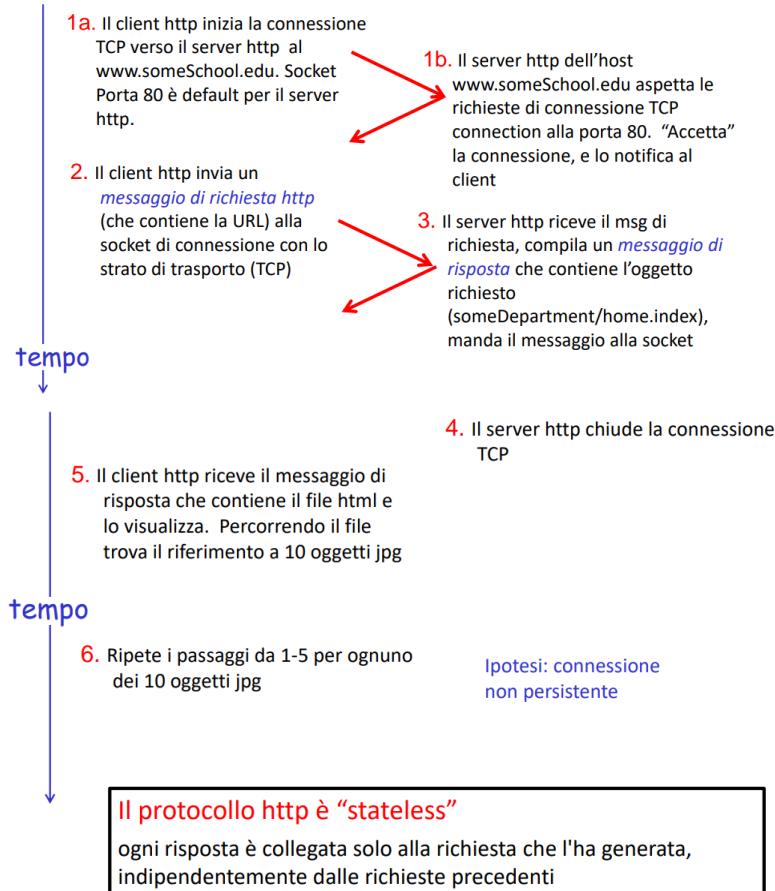


Figura 5: Esempio di interazione con protocollo HTTP.

L'utilizzo di una connessione persistente non permette di base ad un client di inviare una serie di richieste per ricevere una corrispondente serie di risposte; l'instaurazione di una connessione persistente ha lo scopo principale di migliorare le prestazioni della comunicazione.

**Pipelining** Il **Pipelining** è una tecnica di trasmissione delle richieste che consiste nell'invio da parte del client di molteplici richieste senza aspettare la ricezione di ciascuna risposta. Il server deve inviare le risposte nello stesso ordine in cui sono state ricevute le richieste.  
I server web che rispettano HTTP/1.1 devono supportare il pipelining, ma allo stesso tempo pochi browser web supportano questa tecnica. Questo accade perché se una richiesta necessita tempo per essere processata, le risposte alle richieste successive sono bloccate (**Head**

of Line Blocking) e in questi casi lo scopo principale di migliorare le prestazioni viene a mancare.

### 3.3.1 Formato messaggi HTTP

Come detto i messaggi HTTP possono essere di tipo richiesta o risposta, la struttura del messaggio è però la medesima:

- Start line
- Header
- Body

**HTTP request line** Nella start line denominata Request-Line per i messaggi di richiesta sono presenti tre campi:

- **Method:** operazione che il client richiede al server venga effettuata. I metodi più comuni sono GET, POST, PUT e DELETE;
- **Request-URI:** risorsa sulla quale il client vuole venga eseguita l'operazione;
- **HTTP-Version:** il mittente indica il formato del messaggio e la sua capacità di comprendere ulteriori comunicazioni HTTP.

**HTTP status line** Nella start line denominata Status-Line per i messaggi di risposta sono presenti tre campi:

- **HTTP-Version**
- **Status-Code:** intero a tre cifre che sta ad indicare l'esito della risposta (sono definiti dal protocollo).
- **Reason-Phrase:** descrizione testuale dello status code (pensata per l'utente umano).

**Header** Gli header sono coppie (nome: valore) che specificano alcuni parametri del messaggio trasmesso o ricevuto. Esistono vari tipi di header:

- **General Header:** relativi alla connessione (data, codifica, connessione,...);
- **Entity Header:** relativi all'entità trasmessa (content-type, content-length, data di scadenza,...);
- **RequestHeader:** relativi al messaggio di richiesta;
- **Response Header:** relativi al messaggio di risposta;

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

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
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>
<HEAD><TITLE>Test Page for Red Hat Linux's Apache
Installation</TITLE></HEAD>
<H1 ALIGN="CENTER">It Worked!</H1>
<P>
If you can see this, it means that the installation of the
<AHREF="http://www.apache.org/">Apache</A> software
on this <ahref="http://www.redhat.com/">Red Hat Linux</a>
system was successful. You may now add content to this directory
and replace this page.
</P>
</BODY>
</HTML>
```

**Content Negotiation** Le risorse che un client richiede possono essere disponibili in più rappresentazioni (lingua, formato di dati, dimensione, ecc.). La **Content-Negotiation** è un meccanismo per selezionare la rappresentazione appropriata quando viene servita una richiesta (uso di Request e Entity headers).

### 3.3.2 Metodi

**OPTIONS** Permette di richiedere al server di fornire le opzioni di comunicazione associate ad un URL o al server stesso (le sue capacità, metodi esposti,...).

```
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** Permette di richiedere il trasferimento di una risorsa identificata da una URL o operazioni associate all'URL stessa.

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

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> ...
```

**Conditional GET** Una variante del GET detta GET-Condizionale permette di modificare la risposta del server in base agli header utilizzati (Non modifica il numero totale di messaggi HTTP inviati).

```
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
```

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

**HEAD** Simile al GET, ma il server non trasferisce il message body nella risposta. Utile per controllare lo stato dei documenti (validità, modifiche, cache refresh).

**POST** Permette al client di inviare informazioni al server, le quali vengono inserite nel body.

Lo standard afferma che:

*Il metodo POST è usato per chiedere che il server accetti l'entità (risorsa) nel corpo della richiesta come una nuova subordinata della risorsa identificata dalla Request-URI nella Request-Line.*

Nella pratica la funzione effettivamente eseguita dal metodo POST è determinata dal server e dipendente tipicamente dalla Request-URI.

NB: con le API REST vengono rispettate le specifiche HTTP.

**PUT** Permette al client di chiedere al server di creare o modificare (se già esistente) una risorsa specificata nell'URI.

**DELETE** Permette al client di chiedere al server di cancellare una risorsa identificata dalla Request URI.

NB: I metodi PUT e DELETE sono normalmente non abilitati sui server web pubblici.

**Metodi sicuri** Con **metodi sicuri** si fa riferimento a quei metodi che non hanno "effetti collaterali" (per esempio modificare una risorsa). Questi sono: GET, HEAD, OPTIONS e TRACE.

**Metodi idempotenti** Con **metodi idempotenti** si fa riferimento a quei metodi che presentano la proprietà per la quale  $N > 0$  richieste identiche tra loro hanno lo stesso effetto di una singola richiesta. Questi metodi sono: GET, HEAD, PUT, DELETE, OPTIONS, TRACE.

### 3.3.3 Caching e Cookie

**Web Caching** Tecnica che ha lo scopo principale di soddisfare richieste del client senza contattare i server. Consiste nel memorizzare copie temporanee di risorse Web e servirle al client per ridurre l'uso di risorse e diminuire tempo di risposta. I due principali approcci sono:

- **User Agent Cache:** il browser mantiene una copia delle risorse visitate dall'utente per quando queste verranno nuovamente visitate.
- **Proxy Cache:** server proxy dedicati si intrappongono tra il client e il server. Quando un client esegue una richiesta questa passa per il server proxy e se è già stata memorizzata viene restituita al client; altrimenti il proxy si preoccupa di inoltrare la richiesta al server di competenza e una volta ricevuta la risposta la memorizza e la inoltra al client.

**Cookie** I **Cookie** sono un meccanismo con il quale è possibile memorizzare delle informazioni su un certo client. Come detto HTTP è un protocollo state-less perciò un server non ha modo di tenere traccia dei client (anche perché questi di norma non hanno un ip statico). La prima volta che un client invia un messaggio HTTP ad un server, questo lo "obbliga" a memorizzare un'insieme di informazioni che verranno utilizzate nelle successive comunicazioni.

Gli utilizzi principali dei cookie sono: autenticazione, memorizzazione dati form e in generale creare sessioni su un protocollo state-less.

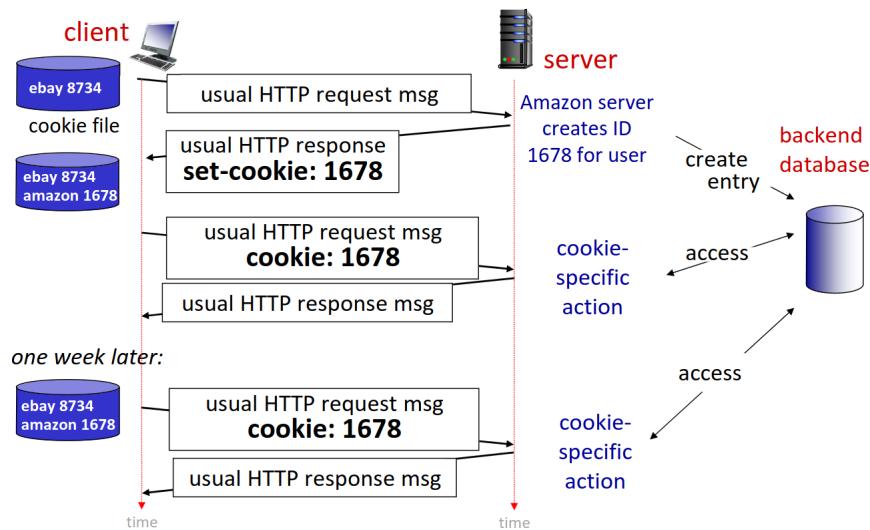


Figura 6: Esempio di interazione con protocollo HTTP e utilizzo di cookie

### 3.4 TELNET

Il protocollo **TERminL NETwork** ha lo scopo principale di permettere l'uso interattivo di macchine remote. Funzionamento:

1. la macchina locale stabilisce una connessione con un server login remoto;
2. tutte le battute dei tasti della macchina locale vengono inviate alla macchina remota, la quale esegue i comandi come se fossero stati battuti sulla macchina stessa.
3. l'output viene inoltrato dalla macchina remota alla macchina locale;

Per poter garantire questo funzionamento il modello di telnet include:

- Un **programma server** che accetta le richieste;
- Un **programma client** che effettua le richieste e interagisce direttamente con l'utente;

La connessione viene realizzata mediante il protocollo TCP sulla porta 23.

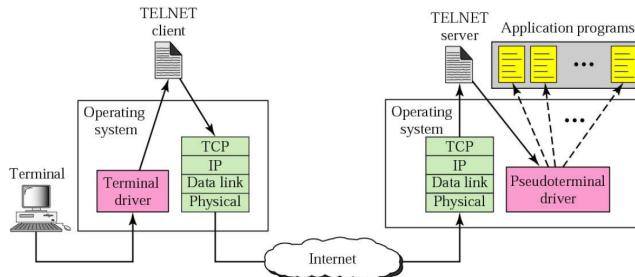


Figura 7: Esempio di interazione con protocollo TELNET

**NVT** Telnet deve poter operare con il numero massimo di sistemi e quindi gestire dettagli di sistemi operativi eterogenei (i quali possono differire per numerosi aspetti). Per risolvere il problema client e server devono eseguire un **Network Virtual Terminal** il quale effettua la codifica dei caratteri del sistema locale nel set di caratteri e comandi specifici della NVT.

Gli NVT si scambiano dati in formato 7-bit US-ASCII. Ogni carattere è inviato come un ottetto con il primo bit settato a zero. I byte con il bit più significativo a 1 vengono usati per le sequenze di comandi

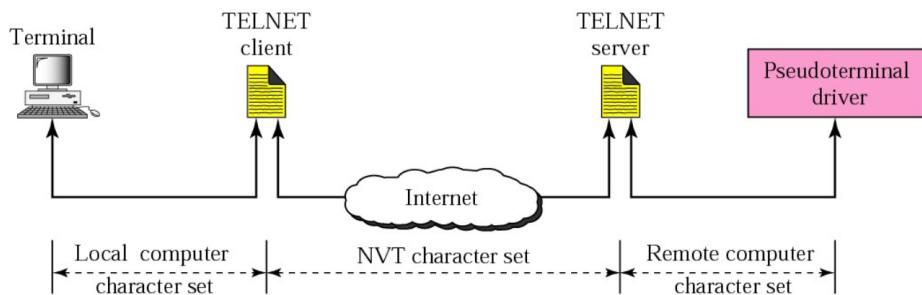


Figura 8: Esempio di interazione con protocollo TELNET e NVT

e vengono preceduti da un ottetto speciale detto IAC (Interpret as Command).

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

**Sicurezza** Telnet non è un protocollo sicuro in quanto tutte le comunicazioni tra macchina locale e remota avvengono in chiaro.

### 3.5 Email

**Posta Elettronica** La posta elettronica è uno dei primi servizi applicativi di internet che consiste nel trasferimento di un messaggio tra uno user mittente ed uno user destinatario. Il servizio di posta elettronica si basa su componenti intermedi per trasferire i messaggi (disaccoppiando mittente e destinatario) per risolvere il problema di disponibilità del destinatario:

- Computer spento;
- Utente impegnato.

I componenti sono:

- **User Agent**, per la composizione, editing, lettura di messaggi di posta (Outlook, Eudora,...);
- **Mail Server**, il quale presenta:
  - **Mail Box**, dove sono archiviati messaggi in ingresso che devono ancora essere letti dall'utente;
  - una coda dei messaggi in uscita che devono essere ancora inviati.
- **Protocollo SMTP**, che permette il dialogo tra i Mail Server;

- **client**, mail server che invia i messaggi;
- **server**, mail server che riceve i messaggi.

**Indirizzo Mail** Un destinatario è identificato da un indirizzo email nella forma:

local-part @ domain-name

- **domain-name**, specifica un mail server. Determina il nome di dominio di una destinazione a cui la mail dovrebbe essere recapitata.
- **local-part**, specifica la cassetta di posta nel mail server. Spesso è identica al nome di login o al nome completo dell'utente.

**Spooling** I server di posta elettronica addottano una tecnica denominata **spooling**:

1. L'utente invia un messaggio e il server ne pone una copia in un'area di memoria denominata **spool** (o anche area di accodamento della posta) insieme a:
  - id mittente;
  - id destinatario;
  - id macchina destinazione;
  - tempo di deposito.
2. Il sistema (**client**) avvia il trasferimento alla macchina remota stabilendo con essa una connessione TCP:
  - se la connessione viene aperta e il trasferimento va a buon fine allora il client cancella la copia locale della mail;
  - altrimenti, il client scandisce periodicamente l'area di spool 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.

### Alias

**Definizione.** Un **alias** è una cassetta postale virtuale che serve a ridistribuire i messaggi verso uno o più indirizzi di posta elettronica personali.

Esistono due tipologie di alias:

- **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;
- **uno-molti**: il sistema permette di associare un gruppo di destinatari ad un singolo identificatore.

### 3.5.1 SMTP

**Simple Mail Transfer Protocol** ha come obiettivo principale il trasferimento affidabile ed efficiente di mail. Le principali caratteristiche sono:

- indipendenza dal sistema di trasmissione usato, SMTP richiede solo il trasferimento di stream di byte ordinato e affidabile.
- la capacità di trasportare mail attraverso più reti. Un messaggio di mail può passare attraverso server intermedi nel percorso da mittente a destinatario finale.

SMTP è detto protocollo di tipo **push**.

**Modello di SMTP** Quando un client SMTP vuole trasferire un messaggio, stabilisce un canale di trasmissione bidirezionale con un server SMTP. La responsabilità di un client è di trasferire la mail a un server SMTP, o comunicare un eventuale insuccesso (scambio formale di responsabilità). Un client SMTP determina l'indirizzo di un host appropriato che ospita un server SMTP risolvendo il nome della destinazione in un indirizzo del mail server destinazione.

Possibili problemi possono essere:

- connessione con mailserver del mittente (server inesistente o irraggiungibile);
- connessione con mailserver destinatario (server inesistente o irraggiungibile);
- inserimento in mailbox destinatario (user unknown, mailbox full).

In tutti questi casi il mittente riceve una notifica.

**Protocollo** Il trasferimento via SMTP avviene in tre fasi:

1. **Handshaking**: Una volta stabilita la connessione TCP il client attende che il server invii **220 READY FOR MAIL**. A questo punto il client invia il comando **HELO** e il server risponde identificandosi;
2. **Trasferimento del messaggio**: Inizia la trasmissione del messaggio, il quale è suddiviso in:
  - **Header**: che contiene informazioni sul mittente, destinatario e oggetto del messaggio;
  - **Body**: il contenuto del messaggio vero e proprio, codificato in 7-bit US-ASCII
3. **Chiusura della connessione**.

La fine di un messaggio in SMTP è identificata dalla stringa **<CRLF>.<CRLF>** (un punto).

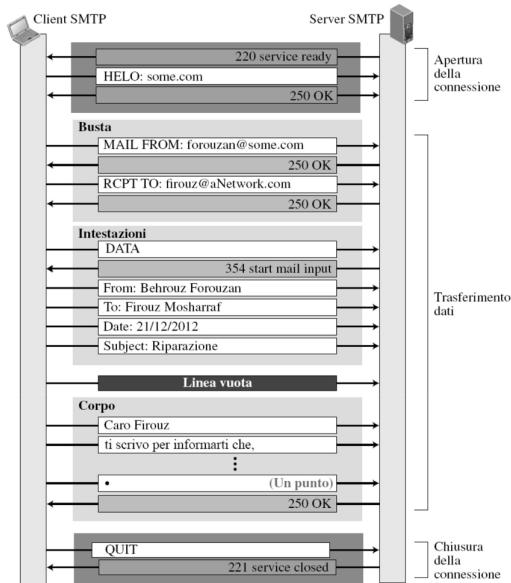


Figura 9: Esempio di interazione con protocollo SMTP

### 3.5.2 Estensioni multimediali

La codifica 7-bit US-ASCII è molto restrittiva, infatti non supporta: accenti, lingue non latine, lingue senza alfabeti e contenuti multimediali come audio o video.

Per risolvere questo problema senza dover in alcun modo apportare modifiche ai Mail Server già esistenti, si è deciso di definire regole di codifica (**encoding**) per il trasferimento di testo non ASCII. (Necessario invece aggiornare o cambiare User Agent)

**MIME Multipurpose Internet Mail Extension** è uno standard di Internet che permette di estendere il formato di e-mail per supportare:

- testo in set di caratteri diversi da US-ASCII;
- allegati in formato non testuale;
- corpo del messaggio con più parti;
- header in set di caratteri non-ASCII.

MIME definisce un insieme di metodi per rappresentare dati binari in formato ASCII.

MIME fornisce vari schemi di transfer encoding, tra cui:

- **Base64 Encoding:** gruppi di 24 bit sono divisi in 4 unità da 6 bit e ciascuna unità viene inviata come un carattere ASCII;
- **Quoted-printable encoding:** per messaggi testuali con pochi caratteri non-ASCII, più efficiente.

L'utilizzo di questo standard per messaggi inviati con SMTP richiede l'utilizzo di ulteriori righe di intestazione per specificare: la versione MIME, il tipo di contenuto, ...

### 3.5.3 Accesso alla posta

Abbiamo visto come SMTP sia un protocollo di tipo push con il quale è possibile inviare un messaggio di posta elettronica fino al Mail server di destinazione. Per recuperare il messaggio però abbiamo bisogno di un protocollo di tipo **pull**, come per esempio:

- **POP3**
- **IMAP**
- **HTTP:** Hotmail , Yahoo! Mail, Gmail, ...

**POP3** Lo User Agent del destinatario apre una connessione TCP sulla porta 110, con il Mail Server e iniziano così le tre fasi per il recupero dei messaggi:

- **Autorizzazione:** Il client esegue l'accesso specificando username e password;
- **Scambio:** Il client con una serie di comandi visualizza i messaggi (**list**), li preleva (**retr**), marca i messaggi da eliminare sul server (**delete**) e chiude la sessione (**quit**);
- **Aggiornamento:** Il server, dopo aver ricevuto il comando quit, cancella i messaggi marcati per la rimozione.

NB: POP3 non mantiene informazioni di stato tra sessioni

**IMAP** IMAP è un altro protocollo di tipo pull per il recupero di messaggi dai Mail server. Presenta più feature di POP3 (per questo è anche più complesso) come la manipolazione dei messaggi memorizzati sul server e comandi per estrarre solo alcune componenti dei messaggi.

### 3.6 DNS

Agli albori di Internet l'associazione tra nomi logici ed indirizzi IP era statica. Tutte le associazioni erano contenute in un unico file, detto **host file**, e periodicamente gli host prelevavano una versione aggiornata del file, **master host file**, da un server ufficiale. Date le dimensioni attuali di Internet, questo approccio è impraticabile per motivi di scalabilità.

Il **DNS** (Domain Name System) ha l'obiettivo principale di fornire un meccanismo per:

- specificare la sintassi dei nomi e le regole per gestirli;
- consentire la conversione dei nomi in indirizzi e viceversa.

Esso è costituito essenzialmente da:

- uno **schema di assegnazione dei nomi** gerarchico e basato su domini;
- un **database distribuito** contenente i nomi e le corrispondenze con gli indirizzi IP implementato con una gerarchia di name server;
- un **protocollo** per la distribuzione delle informazioni sui nomi tra name server basato su:
  - paradigma client/server, in cui i server comunicano tra loro per **risolvere** nomi (traduzione nome/indirizzo);
  - protocolli di trasporto UDP e TCP.

**Servizi** I servizi offerti dal DNS sono:

- **Host aliasing:** Traduzione dei nomi in nome canonico/indirizzo IP (un host può avere più nomi dei quali, uno è detto canonico mentre gli altri sinonimi);
- **Mail server aliasing:** analogo all'host aliasing ma viene distinto perché potrebbero esistere nomi (sinonimi) identitici tra web server e mail server;
- **Distribuzione di carico:** Ad un nome canonico potrebbero essere associati più indirizzi IP (server replicati) ed in questo caso il DNS restituisce la lista di indirizzi IP variandone l'ordinamento ad ogni risposta.

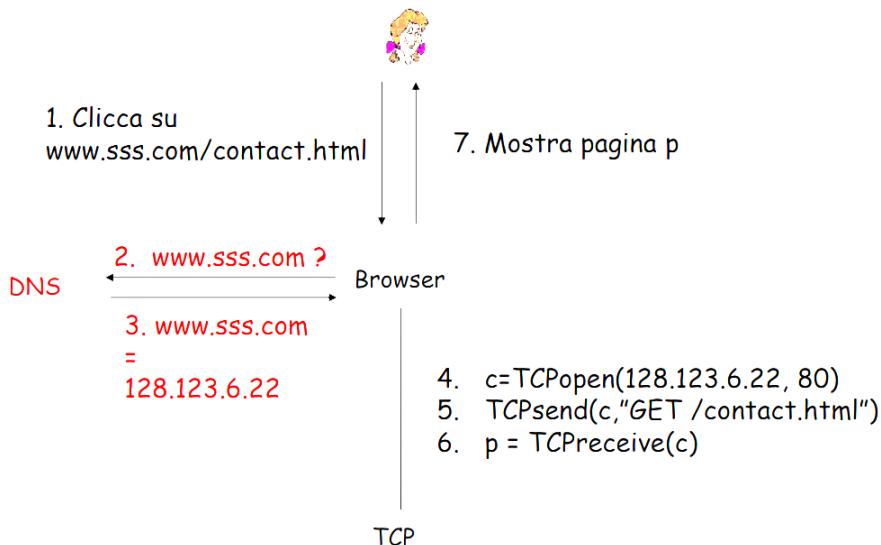


Figura 10: Il DNS si frappone tra client e server (Non interagisce mai direttamente con l'utente).

### 3.6.1 Spazio dei nomi

**Struttura dei nomi** I nomi devono permettere di identificare in modo univoco un host. Questi presentano un struttura gerarchica, infatti un nome è costituito da diverse parti (es. `lab3.di.unipi.it`).

L'assegnazione dei nomi è delegabile, infatti il sistema è in buona parte decentralizzato. La delega avviene a favore dell'autorità per l'assegnazione delle varie parti dello spazio dei nomi.

In questo modo si distribuisce la responsabilità della conversione tra nomi e indirizzi.

**Nomi di dominio** I nomi hanno una struttura ad albero con un numero di livelli variabile ed ogni nodo è identificato da un'etichetta (di massimo 63 caratteri) ed ha un nome di dominio dato da una sequenza di etichette separate da punto. Alla radice è associata un'etichetta vuota.

**Definizione** (Dominio). Con **Dominio** si fa riferimento ad un sottoalbero nello spazio dei nomi di dominio che viene identificato dal nome di dominio del nodo radice del sotto albero.

Un dominio può essere suddiviso in ulteriori domini detti sottodomini. In Internet i nomi gerarchici delle macchine sono assegnati in base alla struttura delle organizzazioni che ottengono l'autorità su porzioni dello

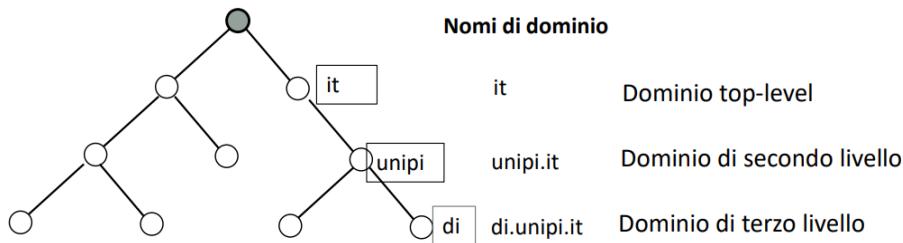


Figura 11: Struttura Domini

spazio dei nomi.

In questo modo è sempre garantita l'univocità dei nomi.

### 3.6.2 Gerarchia dei NameServer

Per ogni dominio non è presente un unico NameServer ma un insieme di essi, per migliorare le prestazioni e per motivi di sicurezza.

**Definizione** (Zona). Una **Zona** è una porzione dello spazio dei nomi di dominio che è gestita da una specifica amministrazione.

NB: Zona e Dominio non necessariamente coincidono.

Ogni server immagazzina le informazioni relative alla propria zona inclusi i riferimenti ai name server dei domini di livello inferiore.

**Gerarchia** La gerarchia è così composta:

- **Server Radice:** responsabile dei record della zona radice, riconosce tutti i domini di massimo livello (TLD - Top Level Domain) e restituisce informazioni su essi;
- **Server Top-Level Domain:** mantiene le informazioni dei nomi di dominio che appartengono al suo TLD e restituisce informazioni sui nameserver di competenza dei sottodomini.
- **Authoritative Name Server:** autorità di una certa zona che memorizza nome e indirizzo IP di un insieme di host. Per ogni zona possono essere presenti server di competenza primari e secondari. I server di competenza primari mantengono il file di zona (file contenente le associazioni nome/indirizzo più aggiornate) mentre i server di competenza secondari ricevono il file di zona e offrono il servizio di risoluzione.

**Local Name Server** Quando un programma (es. browser) deve trasformare un nome in un indirizzo IP chiama un programma in locale detto **resolver**, passando il nome come parametro di ingresso. Se il resolver non ha l'associazione richiesta, interroga un name server di cui conosce l'IP (**Local Name Server**) Il local name server cerca il nome nelle sue tabelle, se trova l'associazione restituisce l'indirizzo al resolver, altrimenti inoltra la query alla gerarchia DNS.

Il Local Name Server non appartiene strettamente alla gerarchia DNS, ogni ISP (Università, società,...) ha il suo (**default**) **name server locale** Il server DNS locale opera da proxy e inoltra la query in una gerarchia di server DNS.

### 3.6.3 Tipologia di Query

Quando un local name server deve instradare una query alla gerarchia DNS questa può essere gestita in due differenti modalità:

- **Query Iterativa:** Una volta inoltrata alla gerarchia DNS, il local name server riceverà la traduzione completa del nome, e la gerarchia si occuperà in modo trasparente di tutte le richieste conseguenti alla prima. Questa tipologia di risoluzione delle richieste è vantaggiosa per il local name server perché non ne aumenta il carico.
- **Query Ricorsiva:** Una volta inoltrata alla gerarchia DNS, il local name server riceverà una risposta contenente i riferimenti per contattare il DNS di competenza per il proseguo della traduzione del nome. Questa tipologia di risoluzione è svantaggiosa per il local name server ma permette di ridurre il carico per la gerarchia.

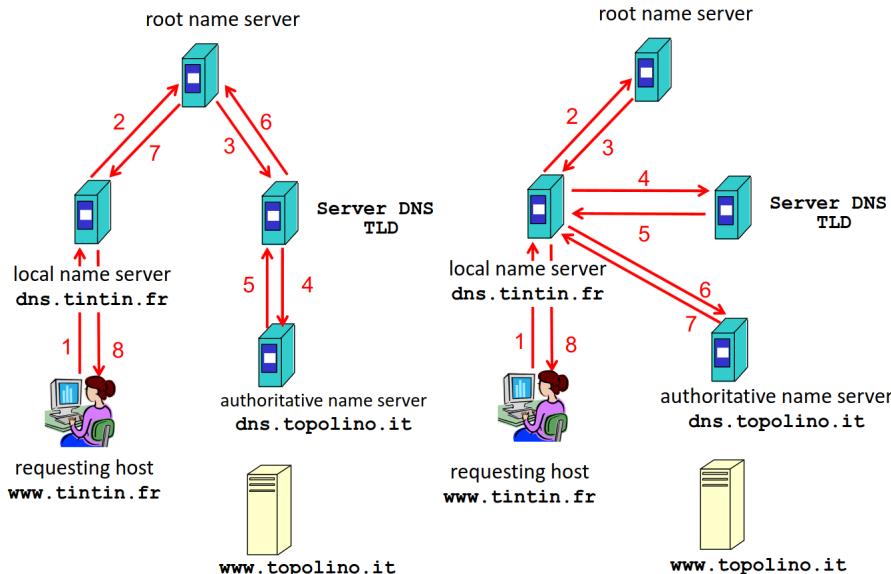


Figura 12: Tipologia di Query DNS

### 3.6.4 Caching e aggiornamento Record

Per ridurre il carico sulla gerarchia DNS e per ridurre i ritardi, i server DNS presentano un cache nella quale vengono memorizzati i **Resource Record** (associazioni), i quali vengono cancellati dopo un certo tempo (Timeout-TTL).

I Resource Record hanno il seguente formato:

(Name,Value,Type,TTL)

- TTL, indica quanto il record dovrà essere rimosso dalla cache.
- Type, specifica come devono essere interpretati i campi Name e Value:
  - Type=A
    - \* Name, Hostname
    - \* Value, Indirizzo IP
  - Type=CNAME
    - \* Name, Hostname (sinonimo)
    - \* Value, Hostname (canonico)
  - Type=NS
    - \* Name, Nome di dominio (es. unipi.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

### 3.6.5 Messaggi DNS

Come detto in precedenza il protocollo DNS utilizza sia il protocollo UDP che il protocollo TCP. Nello specifico il protocollo TCP viene utilizzato unicamente per il trasferimento di file di grandi dimensioni (es. i file di zona) mentre UDP viene utilizzato per il resto delle comunicazioni. Nel protocollo DNS inoltre i messaggi di richiesta/risposta presentano il medesimo formato. I messaggi sono suddivisi in Header e Body. Nell'header sono

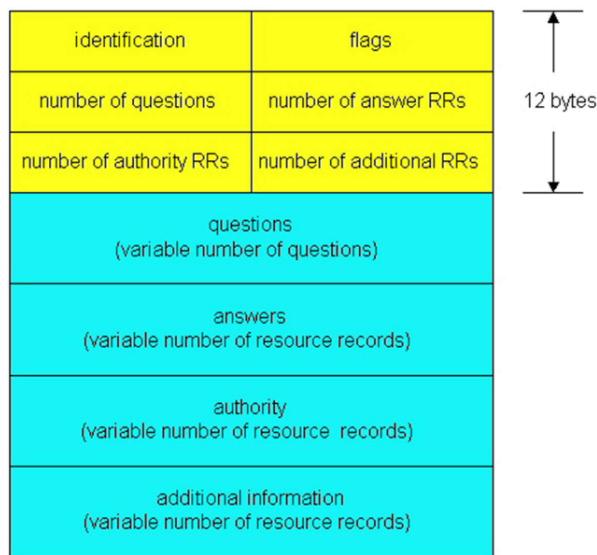


Figura 13: Struttura dei messaggi DNS

presenti:

- **Flags:** che permette di distinguere i messaggi di richiesta e di risposta;
- **Identification:** identificativo a 16 bit che identifica una coppia richiesta/risposta.

Nel body invece sono presenti:

- **Questions:** insieme di campi contenenti il nome richiesto e il tipo di query;
- **Answers:** Resource Record relativi alle richieste effettuate;
- **Authority:** Resource Record relativi ai server di competenza;

NB: Nelle risposte sia il campo Answer che Additional possono anche non contenere alcun RR.

### 3.6.6 DNS Hijacking

Il **DNS Hijacking** (dirottamento) è la pratica di restituire risposte non corrette alle query DNS reindirizzando il client verso siti malevoli. Le principali tipologie sono:

- Local Hijacking
- Router Hijacking
- Rogue Hijacking
- Man-in-the-Middle Attack

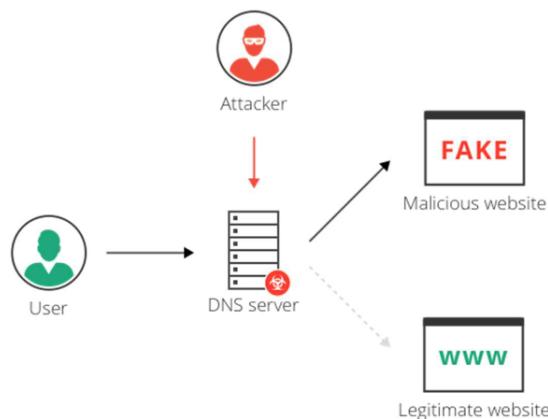


Figura 14: DNS Hijacking

### 3.7 FTP

Il protocollo **File Transer Protocol** è il protocollo standard per il trasferimento file in reti TCP/IP. Offre inoltre altre funzionalità, oltre al semplice trasferimento file, come l'accesso interattivo all'albero delle directory del FileSystem remoto e un accesso con autenticazione.

**Modello FTP** FTP mette a disposizione due differenti tipologie di connessione tra client e server:

- **Control connection:** connessione dedicata allo scambio di comandi e risposte (come Telnet);
- **Data connection:** connessione su cui i dati sono trasferiti con modi e tipi specificati. I dati trasferiti possono essere parte di un file, un file o un set di file.

FTP è un protocollo **statefull**, infatti il server deve tenere traccia dello stato dell'utente (connessione di controllo associata ad un account, directory attuale,...).

**Connessione di Controllo** Il clinet FTP contatta il server FTP sulla porta 21 e, una volta ottenuta l'autorizzazione dal server, utilizzerà questa connessione per scambiare comandi. 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 (Telnet). Questa connessione è di tipo **persistente**.

Quando il client attiva la connessione di controllo con il server, usa un numero di porta assegnato localmente in modo casuale e FTP usa la connessione di controllo per permettere a client e server di coordinare l'uso delle porte assegnate dinamicamente per il trasferimento dati.

**Connessione Dati** Quando il server riceve un comando per trasferire un file (da o verso il client) sulla connessione di controllo, il server apre un'altra connessione dati TCP con il client (**Active Mode**). Il file viene trasferito da o verso il client e la connessione viene immediatamente interrotta.

La Connessione Dati è perciò una connessione **non persistente** e ne viene creata una per ogni interazione di trasmissione (non una per file, si possono infatti inviare anche set di file come detto in precedenza).

Questa non è l'unica modalità possibile, infatti esiste anche la **Passive Mode** in cui è il client ad aprire la connessione verso il server, solo dopo avergli chiesto, tramite la connessione di controllo, di mettersi in ascolto un una determinata porta.

**Modalità di trasmissione** Il client e i server FTP sono spesso molto diversi tra loro (SO, FileSystem...). Prima di effettuare un trasferimento file il client deve definire:

- il tipo di file;
- struttura dati;
- modalità di trasmissione.

al fine di risolvere problemi di eterogeneità tra client e server.

Le principali modalità di trasmissione sono:

- **Stream Mode:** FTP invia i dati a TCP come un flusso continuo di bit;
- **Block Mode:** FTP invia i dati a TCP suddivisi in blocchi. Ogni blocco è preceduto da un header;
- **Compressed Mode:** Il file viene trasmesso compresso.

**Anonymous FTP** Esistono server che supportano connessioni FTP senza autenticazione, in generale permettono un accesso limitato al FileSystem con un sottoinsieme ridotto di funzionalità.

**FTPS** Modalità che può essere usata da client e server FTP per implementare sicurezza e autenticazione usando il protocollo TLS.

### Esempio di FTP

```
ftp
ftp> open ftp.ed.ac.uk
220 FTP Server
Utente (luther.is.ed.ac.uk:(none)):ftp
331 Anonymous login ok, send your complete email address as your
password
Password:
230 Anonymous access granted, restrictions apply
ftp> ls
200 PORT command successful
150 Opening ASCII mode data connection for file list
incoming
pub
INSTRUCTIONS-FOR-USING-THIS-SERVICE
edupload
226 Transfer complete
ftp: 65 bytes received in 0.03secondi (2.60Kbyte/sec)
```

## 4 Lo Strato di Trasporto

Lo strato di trasporto dello stack protocolare TCP/IP presenta caratteristiche quali:

- Realizzazione di **comunicazioni logiche** fra processi residenti su host diversi. Logiche perché i processi si comportano come se gli host fossero direttamente collegati, ingorando i dettagli infrastrutturali.
- Offrire servizi di trasporto (per l'appunto) allo strato applicativo, che possono essere:
  - Sequenze di messaggi singoli;
  - Sequenza continua di byte.

L'applicazione manda i dati al livello di trasporto, nella forma richiesta per la consegna.

- Sfrutta i servizi dello strato di rete che si occupa della comunicazione tra host e il quale consegna i datagrammi all'host destinatario (e non al processo).

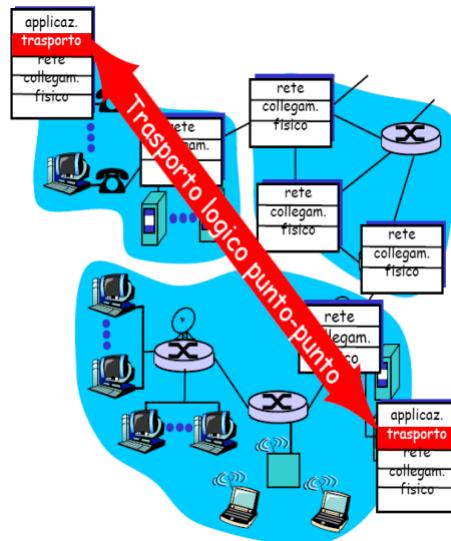


Figura 15: Lo strato di trasporto.

**Tipologie di Connessione** Lo strato di trasporto offre principalmente due tipologie di servizio:

- **Servizio privo di connessione:** In questo caso il processo mittente di occupa di consegnare i **messaggi** al livello di trasporto uno per uno e il livello di trasporto li considera in maniera indipendente, senza creare alcun tipo di relazione tra di essi.  
In questa tipologia di servizio non vi sono garanzie di consegna o di ordinamento.
- **Servizio orientato alla connessione** In questa tipologia di servizio il processo mittente e il processo destinazione vengono collegati mediante una **connessione logica**.

I due principali protocolli messi a disposizione nel livello di trasporto, come detto nel capitolo precedente, sono TCP e UDP che vedremo poi in dettaglio.

**Multiplexing e Demultiplexing** Indipendentemente dal protocollo utilizzato, le azioni che vengono eseguite durante le fasi di invio e ricezione dei segmenti sono le medesime:

- Durante la fase di invio dei dati (**Multiplexing**), il livello di trasporto riceve i messaggi dal livello applicativo, determina gli header per quel segmento, crea il segmento e invia il risultato al livello di rete (IP);
- Durante la fase di ricezione dei dati (**Demultiplexing**), il livello di trasporto riceve i segmenti dal livello di rete, controlla i valori dell'header, estrae il messaggio e lo smista all'applicazione corretta attraverso la socket.

**Concetto di Porta** Ogni datagramma, presenta:

- un indirizzo IP sorgente e IP destinazione;
- un segmento del livello di trasporto, nel cui header sono presenti un numero di porta sorgente e un numero di porta destinazione.

Ogni comunicazione di trasporto è identificata in maniera univoca grazie alle coppie IP/porta degli host. Come visto in precedenza questi sono degli identificativi rispettivamente a 32 bit, per identificare l'host, e 16 bit, per identificare il processo sull'host. Il SO si occupa dell'assegnazione dinamica delle porte ai processi che ne fanno richiesta.

Esistono inoltre delle porte che sono assegnate arbitrariamente da IANA e che non possono essere utilizzate liberamente dagli utenti o dalle applicazioni.

**Demultiplexing con e senza connessione** La fase di Demultiplexing dipende fortemente dal protocollo utilizzato. Nel caso del protocollo UDP le socket vengono identificate univocamente da una coppia IP/Porta, questo sta a significare che due datagrammi con IP e/o porta sorgente differenti ma medesima coppia IP/porta destinazione verranno consegnati alla stessa socket.

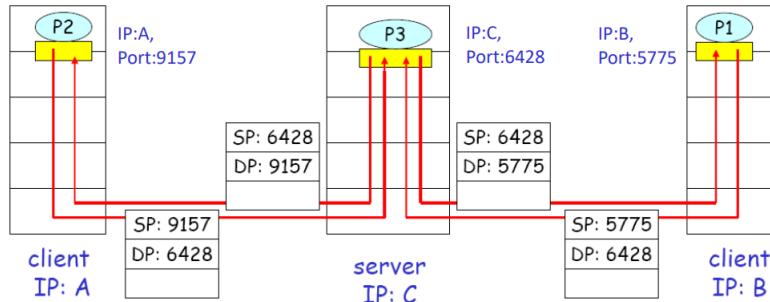


Figura 16: Demultiplexing UDP.

Questo non è vero per le socket del protocollo TCP le quale vengono identificate da una quadrupla data da:

- Indirizzo IP di origine
- Numero di porta di origine
- Indirizzo IP di destinazione
- Numero di porta di destinazione

Una possibile conseguenza di ciò è che un host server può supportare più connessioni distinte contemporaneamente sulla stessa porta.

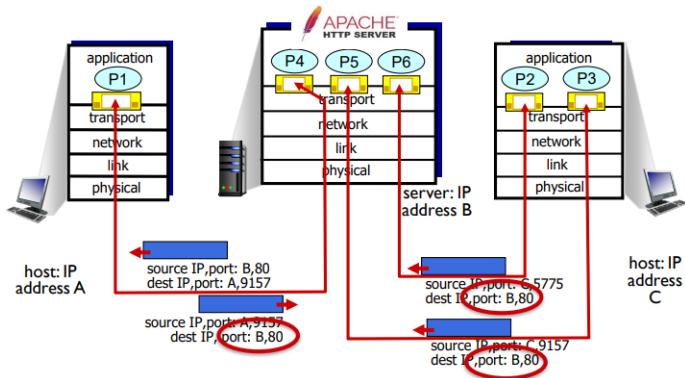


Figura 17: Demultiplexing TCP.

## 4.1 TCP

Il protocollo TCP come detto è uno dei due principali protocolli offerti dal livello di trasporto. Questo presenta una serie di proprietà quali:

- **Orientamento allo stream:** i dati vengono visti come un flusso continuo di byte ordinati, ma non strutturati. La lunghezza di questo fullo è indefinita ma la macchina destinazione riceve esattamente la stessa sequenza di byte che la macchina mittente ha inviato.
- **Orientamento alla connessione:** prima che avvenga lo scambio di dati i processi residenti sulle due macchine effettuano un **handshake**, una procedura di scambio di informazioni preliminari. Viene detto **orientato** alla connessione perché lo stato della connessione risiede sui punti terminali e non sugli elementi intermedi. TCP quindi fornisce agli **USER** l'illusione di avere un circuito dedicato fornendo loro un servizio **Connection Oriented**, basandosi però sul protocollo di rete IP il quale fornisce servizio un **Connection Less**.
- **Connessione full-duplex:** il flusso di dati tra due host può avvenire contemporaneamente nelle due direzioni; queste infatti sono separate tra loro.
- **Trasferimento bufferizzato:** TCP è in grado di suddividere il flusso di byte in segmenti in modo indipendente dal programma applicativo che li ha generati. Appena i dati sono sufficienti per riempire un segmento ragionevolmente grande, questo viene trasmesso attraverso la rete. Ciò viene implementato tramite l'utilizzo di un buffer. La bufferizzazione consente una riduzione del traffico sulla rete "ottimizzando" in qualche modo il numero di segmenti da trasmettere.
- **Ordinamento e affidabilità:** si intende la capacità di correggere errori quali:
  - dati corrotti;
  - segmenti persi;
  - segmenti duplicati;
  - segmenti fuori sequenza.

Questo viene fatto tramite una serie di controlli:

- **Controllo di sessione:** meccanismi di inizio e fine trasmissione;
- **Controllo di flusso:** evitare di spedire più dati di quanti il ricevente sia in grado di trattare;
- **Controllo di congestione:** ha lo scopo di recuperare situazioni di sovraccarico nella rete.

**Trasferimento bufferizzato** I processi al livello applicativo che utilizzano TCP scrivono/leggono dal buffer (spesso a velocità diverse) ed essendo la comunicazione bidirezionale, entrambi i lati avranno un buffer di invio e un buffer di ricezione.

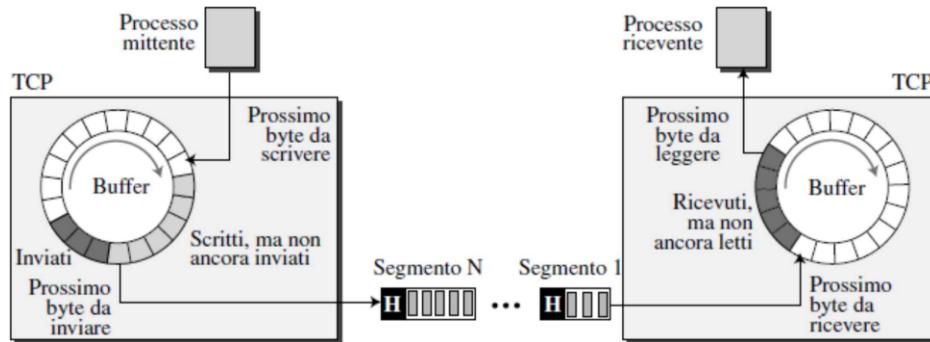


Figura 18: Bufferizzazione TCP.

Il flusso di byte viene partizionato in segmenti, ognuno dei quali ha il suo header, i quali vengono poi consegnati al livello IP.

#### 4.1.1 Formato Segmento

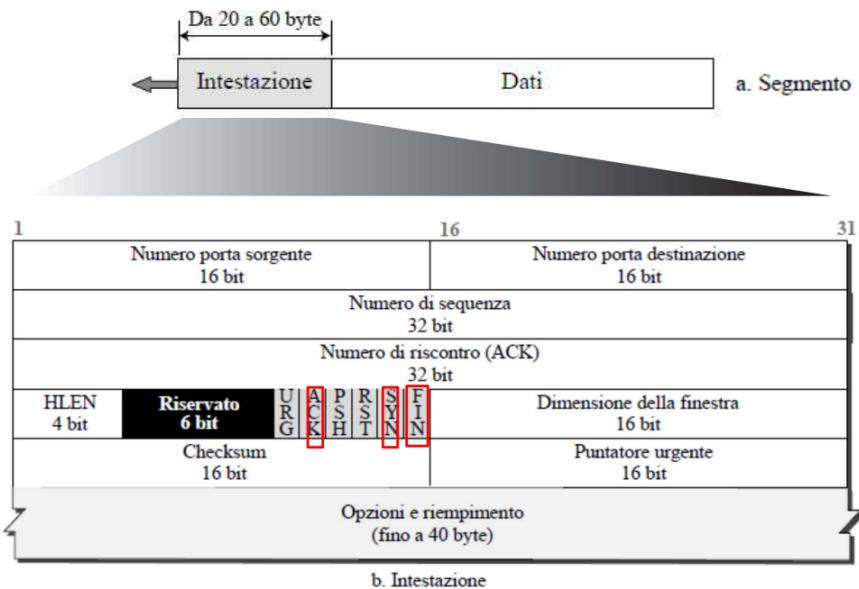


Figura 19: Formato segmento TCP

La semantica dei campi dei campi dell'header TCP è la seguente:

- **Numero di sequenza:** è il numero di sequenza nello stream del primo byte di dati di questo segmento. In TCP infatti vengono numerati i byte e non i segmenti, partendo da un numero casuale  $! = 0$ . Se il flag SYN è settato il numero di sequenza è **ISN** (Initial Sequence Number) e il primo byte di dati è ISN+1.
- **Numero di riscontro:** se il bit ACK è settato, 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.
- **Finestra di ricezione:** 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.

Questi tre campi sono essenziali in quanto permettono di realizzare il **flow control**, il **meccanismo di ritrasmissione** e il **meccanismo di riordino** dei pacchetti in ricezione, necessari per la struttura stream-based del TCP.

- **Hlen:** lunghezza dell'header TCP espressa in parole da 4 byte.
- **Checksum:** checksum dell'intero pacchetto (dati, header TCP e parte dell'header IP) che permette di rilevare errori.
- **Puntatore Urgente:** questo campo è un offset positivo a partire dal **Numero di Sequenza** del segmento corrente. Punta al primo byte di dati non urgenti a partire dal **Numero di Sequenza**, e consente di far passare i dati urgenti in testa alla coda di ricezione. Nel segmento contenente dati urgenti deve essere presente almeno un byte di dati.
- **Bit codice:** sono 6 flag e servono per
  - **URG:** Il campo Puntatore Urgente è significativo e ci sono dati da trasferire in via prioritaria.
  - **ACK:** Il campo Numero di Riscontro contiene dati significativi.
  - **PSH:** Funzione Push (trasferimento immediato dei dati in un segmento dal trasporto al livello applicativo)
  - **RST:** Reset della connessione
  - **SYN:** Sincronizza il Numero di Sequenza
  - **FIN:** Non ci sono altri dati dal mittente, chiusura della connessione

- **Opzioni** (facoltativo, lunghezza variabile, max 40 byte): negoziazione di vari parametri; ad es. dimensione massima segmento (MSS), selective acknowledgement supportato e blocchi di dati riscontrati selettivamente. Le opzioni sono sempre multipli di 8 bit e il loro valore è considerato per il calcolo della checksum.

#### 4.1.2 Gestione della Connessione

**Apertura** L'apertura della connessione avviene tramite un processo denominato **Handshaking a tre vie**.

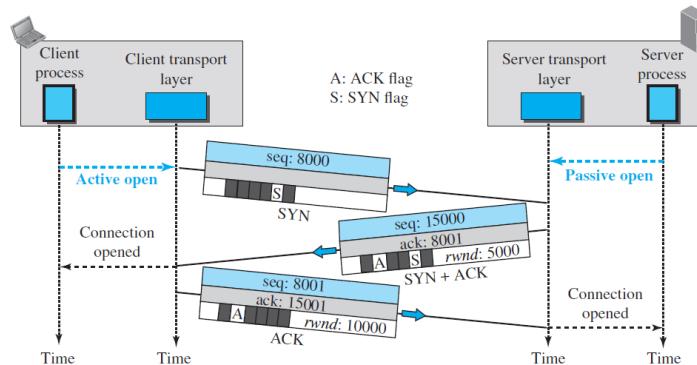


Figura 20: Handshaking a tre vie.

1. Il client invia una richiesta di connessione a un server TCP, tramite un segmento senza dati, con il bit SYN settato e un ISN (Initial Sequence Number).
2. Il server estrae il segmento, alloca i buffer e le variabili TCP per la connessione. Invia in risposta al client un segmento di connessione garantita (denominato **SYNACK**) nel quale appunto sono settati i bit SYN e ACK. Sono inoltre presenti l'ISN e il Numero di Riscontro.
3. Una volta ricevuto il segmento di risposta dal server, il client può a sua volta allocare buffer e variabili di connessione. Invia infine un segmento al server con il bit ACK settato, nel quale sono presenti Numero di Riscontro e Numero di Sequenza. In questo segmento possono già essere trasportati dati.

NB: Dopo l'handshaking a livello di trasporto non c'è più distinzione tra client e server.

**Chiusura** La chiusura della connessione può avvenire indipendentemente nelle due direzioni (in questo caso si parla di **half-close**). Quando

un host vuole chiudere il suo lato della connessione (non vuole inviare altri dati) invia all' altro host un segmento contenente il flag FIN settato e riceverà da questo un segmento con il flag ACK settato. Se entrambi gli host vogliono chiudere la connessione una volta ricevuto un segmento con FIN settato, si può rispondere con ACK e FIN contemporaneamente. È anche possibile lo scambio simultaneo di FIN.

I segmenti contenti i campi FIN settati per la chiusura della connessione non trasportano dati.

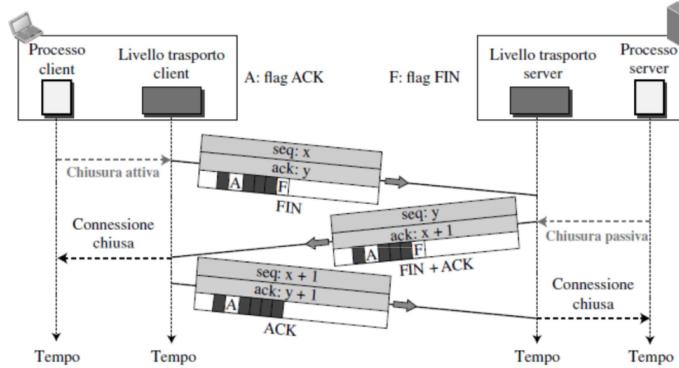


Figura 21: Chiusura connessione TCP.

**Definizione** (Stato TIME-WAIT). **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. La durata di TIME-WAIT è data da 2 volte **MSL** (Maximum Segment Lifetime), cioè la stima del massimo periodo di tempo che un pacchetto IP può vivere sulla rete.

TIME-WAIT viene utilizzato principalmente per:

1. implementare in maniera affidabile la terminazione della connessione in entrambe le direzioni.
2. consentire l'eliminazione dei segmenti duplicati in rete.

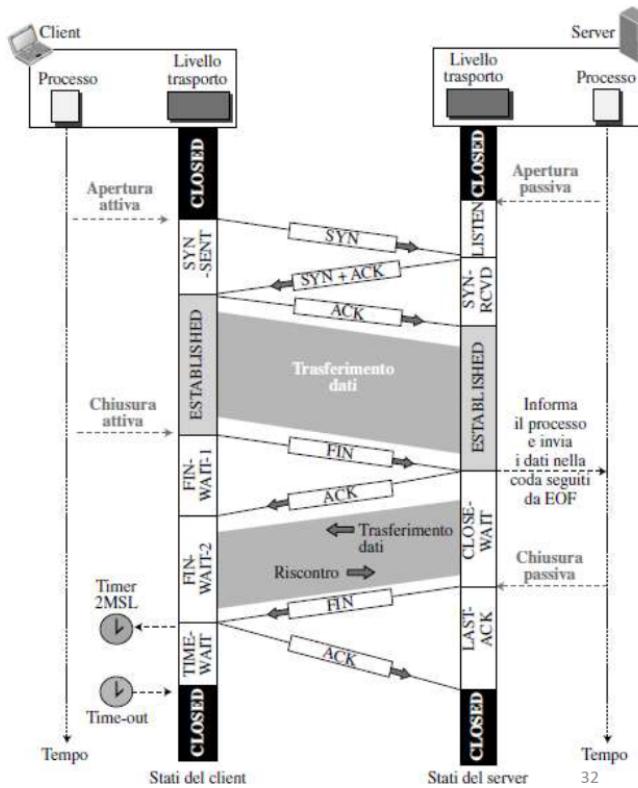


Figura 22: Connessione TCP.

**Stati del TCP** In ogni momento di una connessione tra due host mediante il protocollo TCP ci si trova in uno specifico stato definito dal seguente schema.

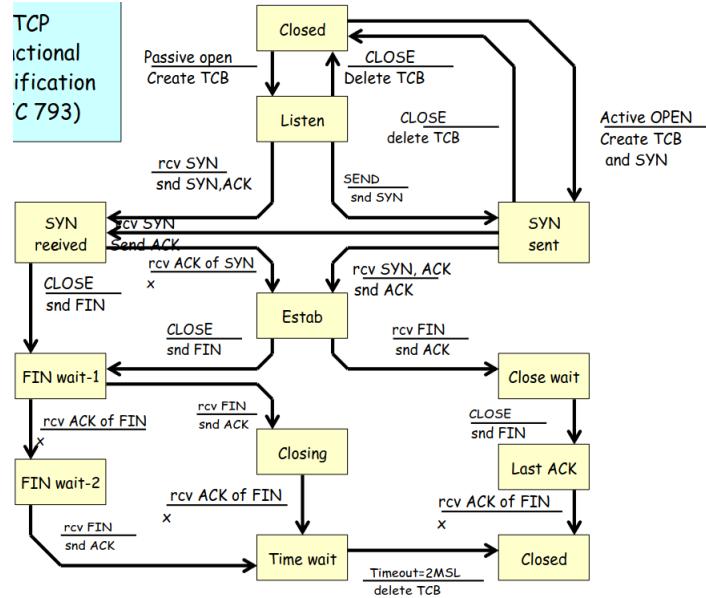


Figura 23: Stati del TCP.

- **LISTEN:** rappresenta l'attesa per una richiesta di connessione da un host TCP remoto.
- **SYN-SENT:** rappresenta l'attesa di una risposta dovuta ad una richiesta di connessione.
- **SYN-RECEIVED:** rappresenta l'attesa per la conferma dovuta alla ricezione e successivo inoltro di una richiesta di connessione.
- **ESTABLISHED:** rappresenta una connessione aperta, i dati ricevuti possono essere trasferiti all'utente. Stato durante la fase di trasferimento dati di una connessione.
- **FIN-WAIT-1:** rappresenta l'attesa di una richiesta di chiusura della connessione da parte dell'host remoto, oppure un attesa di una risposta di accettazione di una richiesta di terminazione della connessione precedentemente inviata all'host remoto.
- **FIN-WAIT-2:** rappresenta l'attesa di una richiesta di chiusura della connessione da parte dell'host remoto.

- **CLOSE-WAIT**: rappresenta l'attesa di una richiesta di chiusura della connessione da parte dell'utente locale.
- **CLOSING**: rappresenta l'attesa dell'accettazione di una richiesta di chiusura della connessione dall'host remoto.
- **LAST-ACK**: rappresenta l'attesa dell'accettazione di una richiesta di chiusura della connessione precedentemente inviata all'host remoto (che include l'accettazione della sua richiesta di chiusura della connessione).
- **TIME-WAIT**: rappresenta l'attesa per una certa durata di tempo per essere sicuri che l'host remoto abbia ricevuto l'accettazione per la sua richiesta di chiusura della connessione.
- **CLOSED**: rappresenta l'assenza di alcun tipo di connessione.

#### 4.1.3 Affidabilità e Controlli

Per permettere un trasferimento dati affidabile TCP, nonostante si basi sul servizio di rete inaffidabile IP, presenta dei meccanismi capaci di corregere gli errori. Come detto per fare ciò utilizza i Numeri di Sequenza e di Riscontro. Utilizza inoltre un **timer di ritrasmissione** detto RTO (che viene avviato se non è già in funzione per un qualche altro segmento)

La ritrasmissione dei segmenti è dovuta:

- **Timeout RTO**: se non abbiamo ricevuto riscontro di uno o più segmenti e che quindi hanno causato il timeout. Il timer viene riavviato
- **ACK duplicato**: se il mittente riceve tre ACK duplicati, il segmento successivo a quello riscontrato è andato perso. In questo caso si utilizza il meccanismo di **Ritrasmissione veloce**, ovvero si ri-inoltrano i segmenti prima del timeout del timer.

I segmenti possono arrivare al destinatario **fuori sequenza**. In questo caso TCP non specifica come questi segmenti devono essere gestiti dal destinatario, dipende dall'implementazione. Nelle versioni più recenti si implementa la **Selective ACK** (SACK)

- i pacchetti ricevuti fuori sequenza vengono memorizzati;
- riscontro di pacchetti fuori sequenza e duplicati inviato in OPTIONS.

Le regole di comunicazione sono le seguenti

1. Tutti i segmenti inviati al destinatario contenenti dati presentano ACK settato.

2. Se il destinatario non ha dati da inviare e riceve segmento in sequenza, ritarda invio ACK di 500ms a meno che non riceva un nuovo segmento.
3. Se il destinatario riceve il segmento atteso e il precedente non è stato riscontrato, allora invia immediatamente ACK.
4. Se il destinatario riceve un segmento fuori sequenza, (5.) presenta un "buco nella sequenza" o ancora (6.) un segmento duplicato, invia immediatamente un segmento ACK (indicando il prossimo numero atteso).

**Esempi** di operatività del TCP:

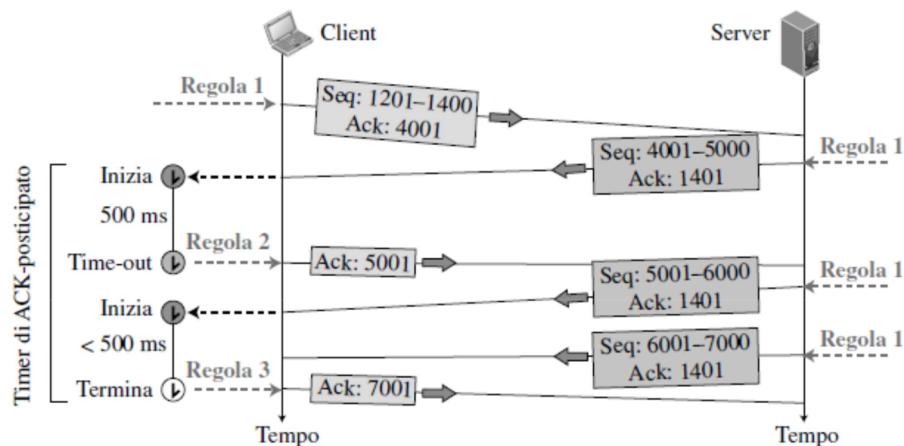


Figura 24: Operatività normale TCP.

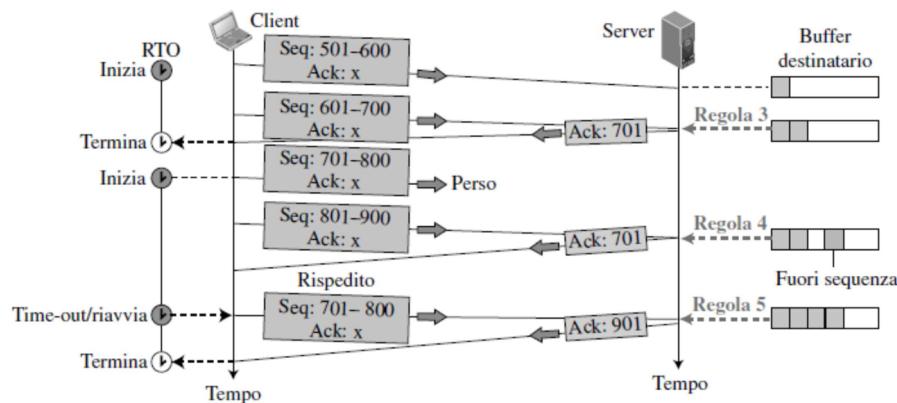


Figura 25: Segmento smarrito TCP

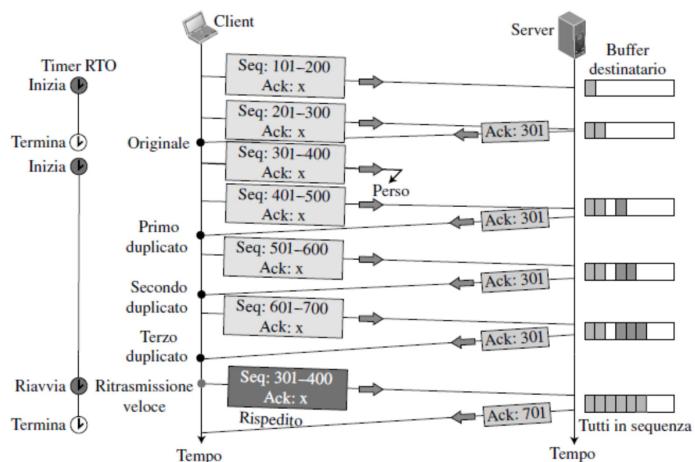


Figura 26: Ritrasmissione veloce TCP.

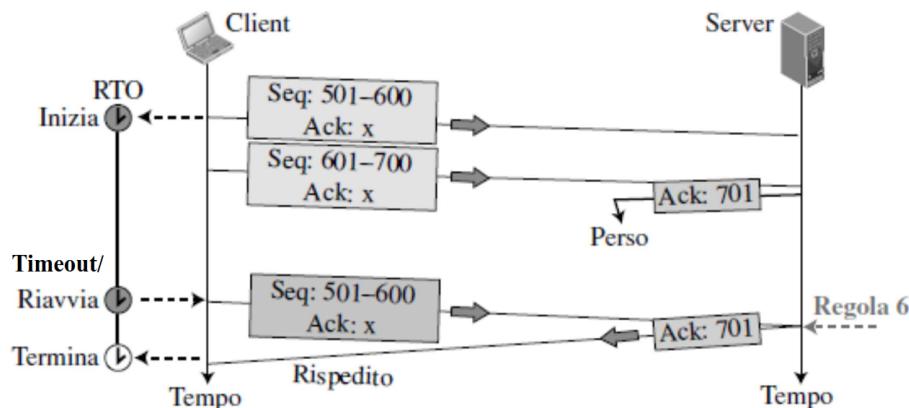


Figura 27: Riscontro perso corretto da ritrasmissione TCP.

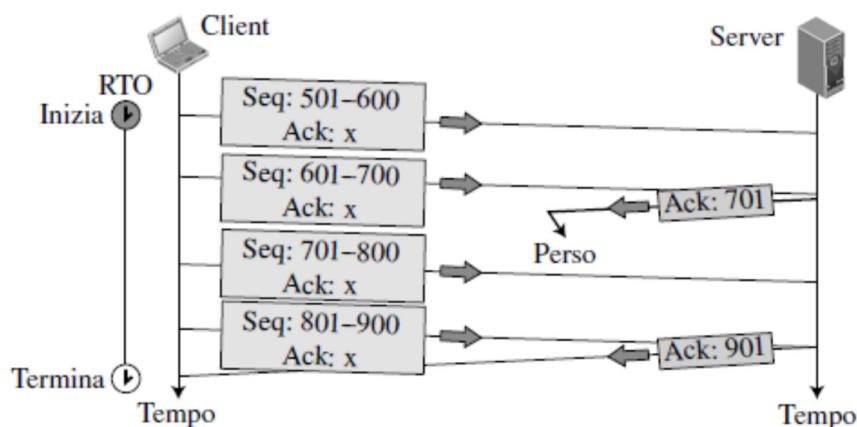


Figura 28: Ricontro smarrito TCP.

**Calcolo del Timeout** Il tempo ti timeout RTO come visto svolge un ruolo fondamentale per il corretto funzionamento del TCP ed è quindi necessario assegnargli un valore corretto. Questo deve essere maggiore di RTT (Round Trip Time), ovvero il tempo trascorso dall'invio di un segmento alla ricezione del suo riscontro. RTT viene calcolato analizzando gli RTT dei segmenti non ritrasmessi ed è dato da

$$\text{Estimated RTT} = (1 - a) * \text{EstimatedRTT} + a * \text{Sample RTT}$$

Sample RTT stimato per un segmento trasmesso in una certa unità di tempo (non per ogni invio) e può variare nel tempo. Il valore di  $a$  viene posto a  $1/8$  in modo da rendere via via meno importanti gli RTT dei pacchetti più vecchi. Oltre al valore RTT stimato è necessario anche una

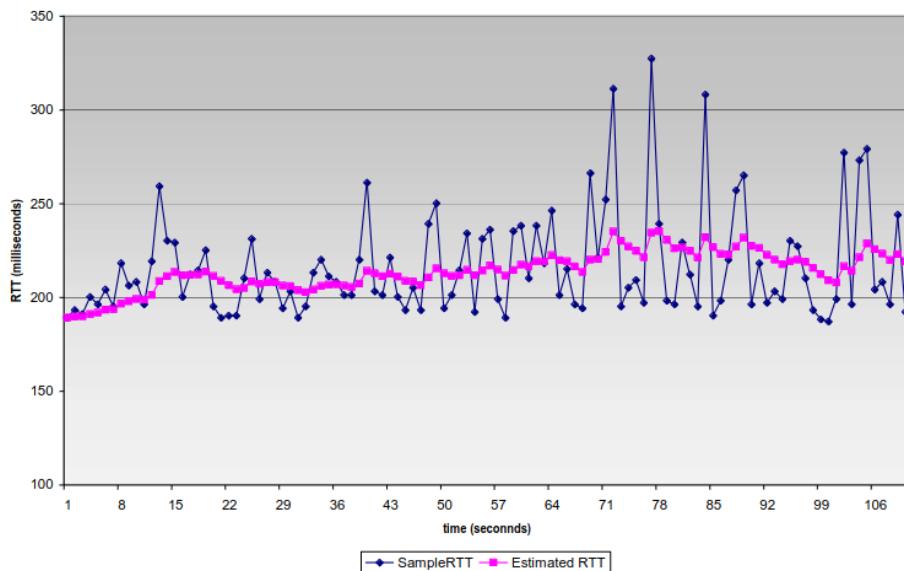


Figura 29: Ricontro smarrito TCP.

stima della variabilità di RTT data dalla seguente formula

$$\text{RTT}_{\text{DEV}} = (1-b) \text{RTT}_{\text{DEV}} + b * |\text{RTT}_{\text{Sample}} - \text{RTT}_{\text{Estimated}}|$$

stima di quanto SampleRTT si discosta da EstimatedRTT. Il valore  $b$  viene posto a  $1/4$

Una volta definiti questi valori l'RTO viene normalmente calcolato come

$$\text{RTO} = (1-b) \text{RTT}_{\text{Estimated}} + 4 * \text{RTT}_{\text{Dev}}$$

In molte implementazioni, dopo un errore (es. ACK non ricevuto) si radoppia il timeout; si tratta di un primo meccanismo di controllo della congestione.

**Finestra di Trasmissione** I dati inviati dal processo a livello applicativo sono mantenuti nel buffer di invio. La trasmissione dei dati si basa sulla **finestra di trasmissione** (sliding window):

- finestra sovrapposta sulla sequenza da trasmettere;
- negoziata dinamicamente;
- viene fatta avanzare alla ricezione di un ACK.

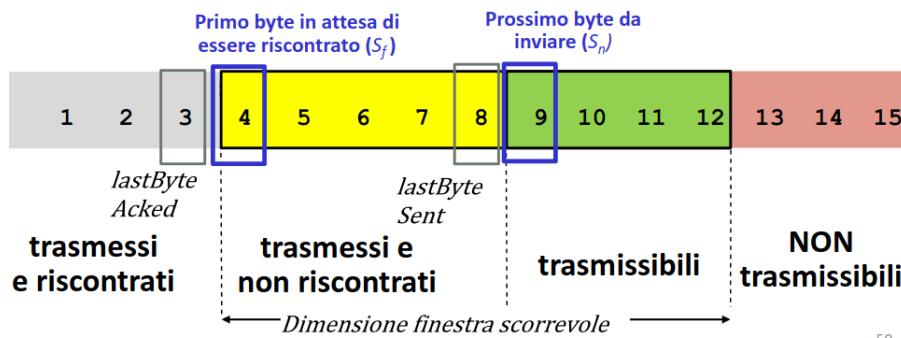


Figura 30: Finestra di trasmissione TCP.

### Controllo di Flusso

**Definizione.** Con **controllo di flusso** si intende la capacità del mittente di evitare la possibilità di saturare il buffer del ricevitore.

Il controllo di flusso mette in relazione la frequenza di invio del mittente con la frequenza di lettura dell'applicazione ricevente. TCP implementa questa caratteristica tramite una variabile detta **finestra di ricezione** (rwnd) mantenuta nel mittente. Questa variabile fornisce un'idea di quanto spazio è ancora a disposizione nel buffer del ricevitore, ed è quindi un valore dinamico.

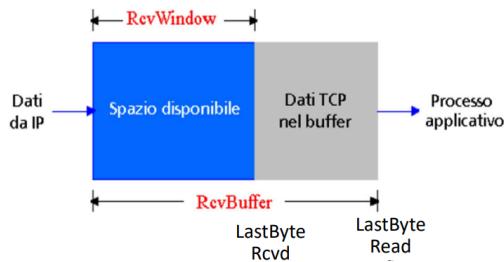


Figura 31: Finestra di ricezione TCP.

Il destinatario comunica il valore della variabile rwnd tramite l'**rwnd field** nell'header TCP. Il mittente sarà così in grado di limitare la quantità di dati **unACKed** ("in-flight"). Inoltre la dimensione del buffer di ricezione è configurabile tramite socket options ed è denominata **RcvBuffer** (dimensione standard 4096 bytes).

Il valore di rwnd viene calcolato come:

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

Il mittente si assicura che

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{Rwnd}$$

cioè che la quantità di dati trasmessi e non ancora riscontrati sia minore della finestra di ricezione. Se rwnd=0, il mittente manda segmenti "sonda" di 1 byte per ricevere aggiornamenti sul valore di rwnd.

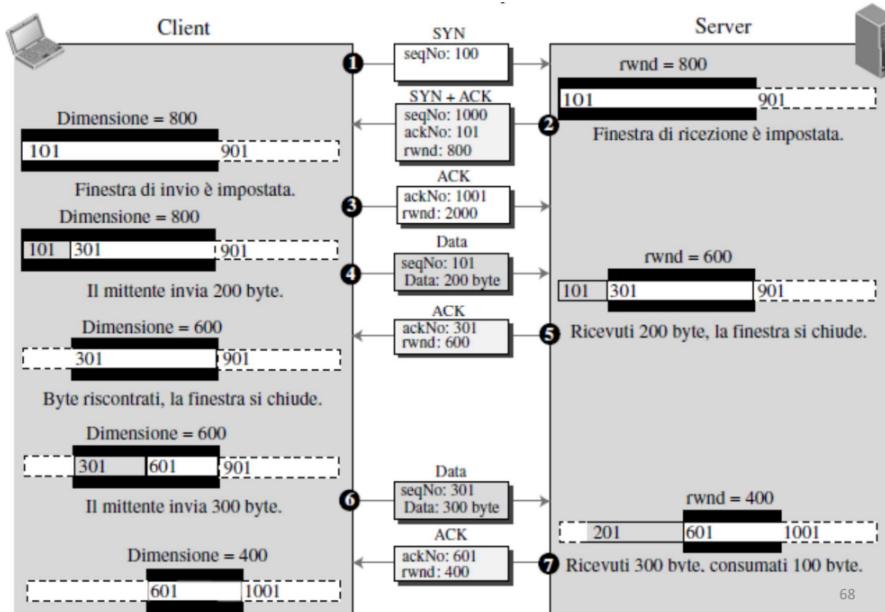


Figura 32: Esempio trasmissione con finestra di ricezione.

**Controllo di Congestione** Il fenomeno della **congestione** è originato dal tentativo di più sorgenti di richiedere più banda di quella disponibile sul percorso, ovvero troppe sorgenti che trasmettono troppi dati a una velocità elevata per cui la rete non è in grado di gestirli.

Il traffico eccessivo nella rete può provocare:

- Lunghi ritardi, dovuti all'accodamento dei pacchetti nei buffer dei router;
- Perdita di pacchetti, dovuta ad overflow dei buffer dei router.

TCP è in grado di implementare meccanismi di congestione senza bisogno di supporto esplicito da parte della rete, imponendo a ciascun mittente di limitare l'invio di pacchetti in rete in funzione della **congestione percepita**.

In generale TCP è in grado di adattarsi alla velocità della rete. Se infatti "percepisce" scarso traffico, aumenta la frequenza di invio altrimenti la diminuisce.

NB: Dovendo gestire sia il controllo di congestione che di flusso la dimensione della finestra di invio in TCP è data valore minimo tra rwnd e cwnd. Quindi la frequenza di invio non può superare  $\min(rwnd, cwnd)/RTT$ . Più in generale cwnd/RTT impone un limite superiore alla frequenza di trasmissione.

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

- Slow Start
- Additive Increase Multiplicative Decrease
- Fast Recovery
- Time-out Reaction

**Congestion Window** La **Congestion Window** viene misurata in **MMS** (Maximum Segment Size), cioè la quantità massima di dati che è possibile trasportare in un unico segmento. MMS viene determinato in base all' **MTU** (Maximum Transmission Unit), lunghezza massima del payload del frame di collegamento inviabile dall'host mittente, in modo tale che il segmento TCP, incapsulato in pacchetto IP, stia dentro un singolo di frame di collegamento (valori tipici: 1460, 536, 512 byte, calcolati togliendo dall'MTU 20 byte header TCP + 20 byte header IP). Come vedremo l'andamento tipico della finestra nel tempo varia seguendo uno schema detto a dente di sega.

**Slow Start** All'inizio la finestra di congestione, è posta pari a 1 MSS (frequenza di invio 1 MSS/RTT). Durante questa fase per raggiungere in fretta la saturazione della banda, la finestra di congestione viene incrementata di 1MSS ad ogni ACK (non duplicato). L'effetto è che la finestra di congestione raddoppia ad ogni RTT, crescita esponenziale.

**AIMD** TCP del mittente aumenta proporzionalmente la propria finestra di congestione ad ogni ACK ricevuto. Ad ogni ACK la finestra di congestione viene incrementata in modo che si abbia un crescita pari ad 1 MSS per ogni RTT ( $cwnd = cwnd+1/cwnd$ ), **Congestion avoidance**.

**TCP RENO** Nell' implementazione denominata **RENO** viene inizialmente inizializzata una variabile detta di **soglia** al quale viene assegnato un valore alto (es. 64KB). La soglia determina quando termina la fase di slow start ed inizia la fase di congestion avoidance.

- Se  $cwnd < \text{soglia}$ , cwnd aumenta esponenzialmente (slow start).
- Se  $cwnd > \text{soglia}$ , cwnd aumenta linearmente (Additive Increase).

La gestione degli eventi di perdita, scadenza del timer o 3 ACK duplicati consecutivi, vengono gestiti in modo diverso in base all'implementazione. In RENO gli eventi di perdita sono così gestiti

- 3 ACK duplicati, viene posta la soglia a  $cwnd/2$  e poi la finestra di trasmissione viene posta a  $\text{soglia}+3\text{MMS}$ . Questa operazione viene detta **Fast Recovery**.
- Scadenza del timer, viene posta la soglia a  $cwnd/2$ , la finestra di trasmissione a 1MMS e ricomincia con una fase di slow start.

Nella fase di fast recovery (si reinviando i segmenti di cui non abbiamo avuto riconcontro)

- se avviene un timeout si va in slow start;
- finché continuano ad arrivare ACK duplicati  $cwnd=cwnd+1$ ;
- se arriva un nuovo ack (non duplicato) si va in congestion avoidance e  $cwnd=\text{soglia}$  (abbiamo ricevuto riscontro di tutti i vecchi segmenti e iniziamo a spedire i nuovi).

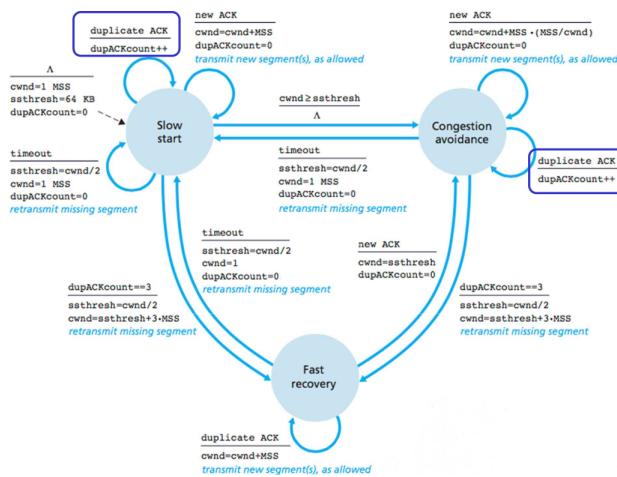
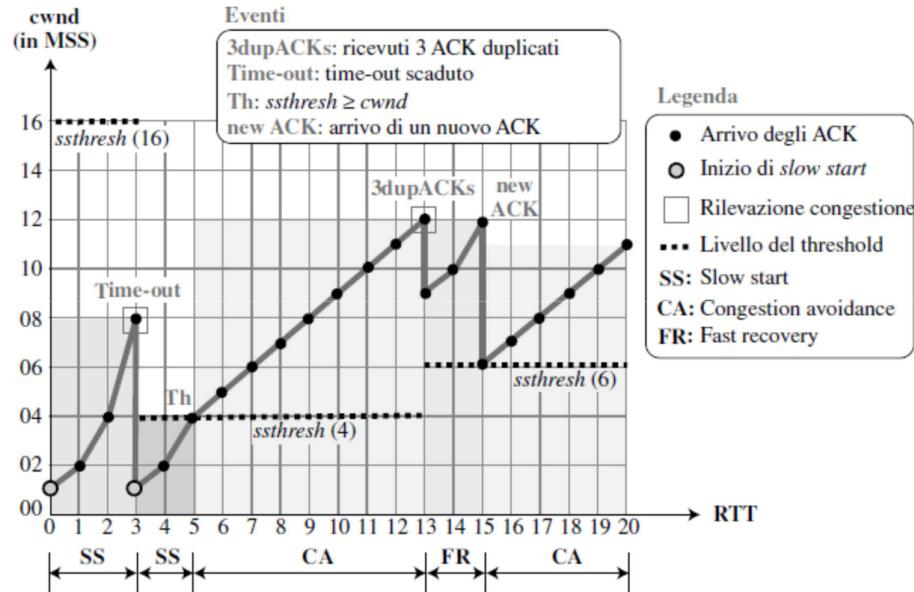


Figura 34: Schmea TCP RENO.

**TCP Tahoe** TCP Tahoe è una versione di TCP precedente a RENO nella quale gli eventi di perdita erano considerati tutti della stessa gravità e gestiti perciò allo stesso modo.

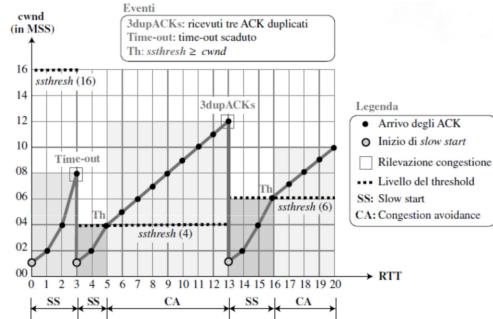


Figura 35: TCP Tahoe.

**TCP CUBIC** Un ulteriore implementazione di TCP è CUBIC (utilizzata per esempio in Linux e in molti web server). Si basa sull'idea che dato  $W_{max}$ , il valore del rate di quanto si è rilevata la congestione, lo stato del collegamento probabilmente non è cambiato. Sia allora  $k$  il punto in cui la cwnd raggiungerà  $W_{max}$ , aumento di  $W$  come una funzione del cubo della distanza tra l'istante di tempo attuale e  $k$ .

- Incrementi maggiori quando siamo lontani da  $K$ ;
- Incrementi più piccoli quando siamo più vicini a  $K$ .

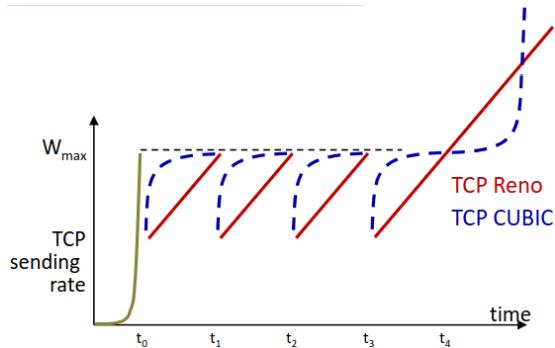


Figura 36: TCP CUBIC.

**Explicit Congestion Notification** Le implementazioni TCP spesso implementano un meccanismo di controllo di congestione **network-assisted**

- due bit in header IP (campo ToS) settati dai router per indicare la congestione;
- Informazione di congestione inviata a destinazione;
- La destinazione imposta il bit ECE (ECN-Echo) nel segmento di riscontro per notificare la congestione al mittente;
- Il mittente setta il bit CWR (Congestion Window Reduced) per indicare che ha ricevuto la notifica di congestione.

#### 4.1.4 Throughput ed Equità

Per quanto visto fin' ora TCP in regime stazionario e dato  $W$  il valore massimo della finestra di trasmissione, offre un throughput dato da

$$(0.75 * W) / RTT$$

Inoltre date le ipotesi:

- $K$  connessioni TCP insistono su un unico link di capacità  $R$  bit/s;
- Le connessioni hanno gli stessi valori di MSS e RTT;
- Non ci sono altri protocolli che insistono sullo stesso link.

Si ha che ognuna delle connessioni TCP tende a trasmettere  $R/K$  bit/s e perciò TCP è equo.

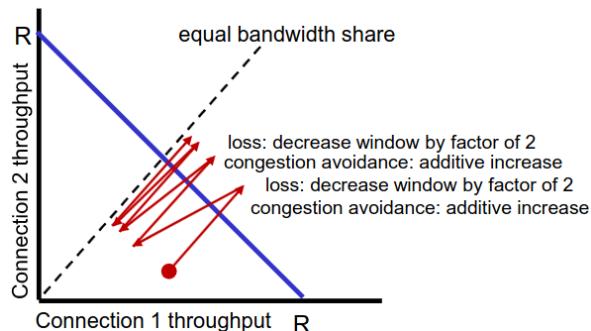


Figura 37: TCP CUBIC.

Nella pratica connessioni con RTT più piccolo variano più velocemente la loro finestra di congestione e raggiungono throughput superiori a connessioni con RTT maggiore.

**Connessioni Parallelle** Un'applicazione può aprire connessioni multiple in parallelo tra due host in modo da garantirsi prestazioni migliori. Infatti gli host che aprono più connessioni parallele sfruttano l'equità di TCP a proprio vantaggio.

Esempio:

Collegamento con rate R con 9 connessioni esistenti:

- se new-app apre una sessione TCP, ottiene un rate  $R/10$ ;
- se invece new-app apre 11 sessioni TCP, ottiene  $R/2$ .

## 4.2 UDP

UDP è un protocollo meno complesso rispetto a quanto abbiamo visto con TCP. Offre meno servizi ma è più indicato in contesti dove occorre un completo controllo della temporizzazione (non intesa come timing, ma per applicazioni time-sensitive come la trasmissione di dati multimediali).

UDP offre un servizio di consegna a **massimo sforzo**. I datagrammi possono essere persi o consegnati fuori sequenza. Viene anche detto **orientato al messaggio** poiché ogni datagramma è indipendente dagli altri e i processi che sfruttano questo protocollo devono inviare messaggi di dimensioni limitate, che possono essere incapsulati in un datagramma UDP.

### Caratteristiche

- Assenza di connessione, quindi non vengono introdotti ritardi dovuti alla gestione della connessione;
- Semplicità, gli header sono formati da soli 8 byte e non presentando alcun tipo di controllo presenta un alto rate di invio;
- Checksum facoltativo;

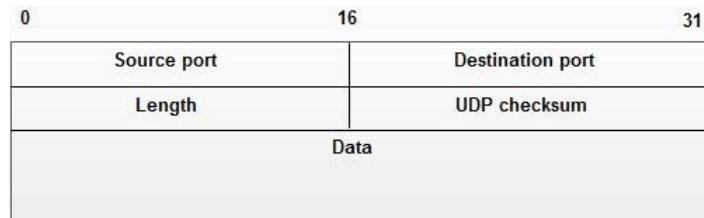


Figura 38: Formato datagramma UDP.

Il checksum, se attivo, viene calcolato su tutto il datagramma e parte dell'header IP. Quando il destinatario riceve un datagramma corrotto lo scarta senza comunicare niente al mittente.

**Calcolo del Checksum** Viene inizialmente settato il campo checksum a 0, il datagramma viene suddiviso in parole da 16 bit che vengono sommate in complemento a 1, con eventuali riporti sommati al risultato, e infine si calcola il complemento a 1 del totale. Questo valore verrà poi inserito nel campo checksum dell'header.

Il ricevente per controllare la correttezza del pacchetto calcola la checksum del datagramma ricevuto (incluso il campo checksum)

- se il valore risultante è 0, allora il pacchetto non ha subito corruzioni;
- altrimenti il pacchetto è corrotto e viene scartato.

NB: Questo procedimento è il medesimo utilizzato in TCP, nel quale però è obbligatorio.

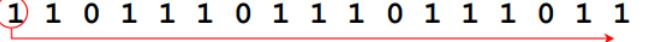
	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
																
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Figura 39: Calcolo del checksum.

#### 4.2.1 Vulnerabilità di UDP

Nonostante la maggiore semplicità di UDP questo, a differenza di TCP, è maggiormente suscettibile ad azioni malevoli. I due principali attacchi ad UDP sono:

- **UDP Flood**:
- **UDP Amplification**.

Il primo è un attacco di tipo **DOS** (Denial of Service) in cui un gran numero di pacchetti UDP viene inviato a un server obiettivo, con porta destinazione scelta in modo random. Il server prima controlla se sono in esecuzione programmi che sono attualmente in attesa di richieste sulla porta specificata. Se nessun programma riceve pacchetti su quella porta, il server risponde con un pacchetto ICMP per informare il mittente che la destinazione non è raggiungibile. Come risultato dell'utilizzo delle risorse da parte del server di destinazione per controllare e quindi rispondere ad ogni pacchetto UDP ricevuto, le risorse possono esaurirsi rapidamente, con conseguente denial-of-service al traffico normale.

Nel secondo caso si sfrutta il fatto che UDP non verifica l'indirizzo IP di origine del datagramma e che alcuni protocolli applicativi che usano UDP possono generare risposte molto più grandi della richiesta iniziale. In questo modo è facile forgiare un pacchetto con indirizzo IP di origine arbitrario (in questo caso è l'IP della vittima). L'attaccante invia numerosi pacchetti UDP con l'indirizzo IP di origine falsificato ad un server di destinazione (o amplificatore). Il server risponde alla vittima (anziché all'attaccante), creando un attacco un attacco DOS di riflesso.