

---

# RETI DI CALCOLATORI

---

## Corso A

**Autore**

Giuseppe Acocella

2025/26

<https://github.com/Peenguino>

Ultima Compilazione - November 9, 2025

# Contents

<b>1</b>	<b>Introduzione alle Reti</b>	<b>4</b>
1.1	Tipi di Rete per Dimensione . . . . .	5
1.2	Gestione di Internet come Rete di Reti . . . . .	5
1.3	Reti e Mezzi d'Accesso . . . . .	5
1.3.1	Schema dell'Accesso DSL . . . . .	5
1.3.2	Tipi d'Accesso - Cavo, Aziendale, Domestico, Mobile e Satellitare . . .	6
1.4	Nucleo della Rete - Commutazione Circuito vs Pacchetto . . . . .	7
1.5	Metriche - Prestazioni della Rete (Latenza, Throughput, Prod. Rate/Ritardo)	8
1.6	Modello Stratificato . . . . .	9
1.6.1	Stack Protocollore TCP/IP . . . . .	9
1.6.2	Stack Protocollore ISO/OSI - Cenno Storico . . . . .	10
1.7	Breve Introduzione alla Sicurezza di Reti . . . . .	10
<b>2</b>	<b>Livello Applicazione</b>	<b>11</b>
2.1	Introduzione a Paradigmi, Protocolli e Tipi d'Utilizzo . . . . .	11
2.2	TCP vs UDP nel Contesto del Livello Applicazione . . . . .	12
2.2.1	API e Socket . . . . .	12
2.3	Protocollo HTTP - HyperText Transfer Protocol . . . . .	13
2.3.1	Versioni HTTP, Persistenza e Pipelining . . . . .	14
2.3.2	Messaggi HTTP . . . . .	15
2.3.3	Content Negotiation . . . . .	16
2.3.4	Metodi dell'HTTP . . . . .	16
2.3.5	Web Caching . . . . .	17
2.3.6	Cookies . . . . .	17
2.4	Protocollo Telnet . . . . .	18
2.5	Protocollo SMTP - Simple Mail Transport Protocol . . . . .	19
2.6	Protocolli POP3/IMAP . . . . .	20
2.7	DNS - Domain Name System . . . . .	20
2.7.1	Gerarchia dei Name Servers . . . . .	21
2.7.2	Protocollo DNS . . . . .	23
<b>3</b>	<b>Livello Trasporto</b>	<b>25</b>
3.1	Introduzione al Livello Trasporto . . . . .	25
3.2	TCP vs UDP e Multiplexing/Demultiplexing . . . . .	26
3.3	Trasporto UDP . . . . .	27
3.4	Principi di Trasferimento Affidabile . . . . .	28
3.4.1	RDT 1.0 - Trasferimento su Canale Affidabile . . . . .	29
3.4.2	RDT 2.0 - Trasferimento su Canale con Errori su Bit . . . . .	29
3.4.3	RDT 2.1 - Checksum per Controllo Corruzione ACK . . . . .	30
3.4.4	RDT 2.2 - Protocollo NAK-free . . . . .	31
3.4.5	RDT 3.0 - Timer in Attesa di ACK . . . . .	31
3.4.6	Prestazioni e Reale Utilizzo del Canale . . . . .	31
3.4.7	Approcci di Pipelining - Go-Back-N (GBN) e Selective Repeat (SR) .	32
3.4.8	Riassunto Meccanismi Affidabili . . . . .	34

3.5	Protocollo TCP . . . . .	34
3.5.1	Formato Segmento TCP . . . . .	35
3.5.2	Gestione della Connessione TCP - Handshaking e Chiusura . . . . .	36
3.5.3	Trasferimento Affidabile del TCP . . . . .	38
3.5.4	TCP RTT, RTO, Stime Timeout e Finestre di Ricezione . . . . .	39
3.5.5	Controllo di Flusso del TCP . . . . .	40
3.5.6	Controllo di Congestione del TCP . . . . .	40
3.5.7	Tipologie di TCP che Istanzano Algoritmo di Controllo Congestione - (Reno, Tahoe, Cubic e Vegas) . . . . .	41
3.5.8	Throughput, Equità e Cenni di Attacchi TCP . . . . .	43

# 1 Introduzione alle Reti

Come facciamo a costruire una rete di computer che sia scalabile e che supporti **svariati tipi di applicazioni** come streaming, messaggistica e videochiamate? Internet ci permette di farlo, e tecnicamente è un insieme di miliardi di host (servers, laptops, smartphones) connessi tra loro.

**Internet - Vista dei Componenti** Questa connessione è permessa da specifici componenti:

1. **Link di Comunicazione:** Fibra ottica, rame, onde radio, microonde.
2. **Dispositivi di Interconnessione:** I più comuni sono:
  - (a) **Switch:** Dispositivo che collega tra loro **più host**.
  - (b) **Router:** Dispositivo che collega tra loro **più reti**.
3. **Reti:** Insieme di host, dispositivi di interconnessione e link gestiti da una stessa organizzazione.

**Internet - Vista dei Servizi** Da un punto di vista di servizi offerti invece Internet:

1. E' un **infrastruttura** che offre servizi alle **applicazioni distribuite**.
2. Offre un **interfaccia di programmazione** (socket) che permette alle applicazioni, distribuite su **host diversi**, di scambiarsi informazioni.

**Internet - Vista delle Entità Software**

1. **Applicazioni:** Elaborano e si scambiano informazioni.
2. **Protocolli:** Regolano la trasmissione di queste informazioni, definendo quindi il formato e l'ordine dei messaggi scambiati tra due o più entità in comunicazione. Due esempi noti di protocolli internet sono:
  - (a) **Transmission Control Protocol (TCP)**
  - (b) **Internet Protocol (IP)**

**Standardizzazione di Protocolli** I protocolli di Internet vengono standardizzati da diversi enti, come:

1. Internet Engineering Task Force (IETF). Si basano su una procedura detta RFC (Request for Comments) che seguendo passi rigorosi definisce gradualmente nuovi standard per Internet.
2. IEEE 802 LAN Standards.
3. World Wide Web Consortium (W3C): Comunità internazionale che sviluppa standard aperti per favorire lo sviluppo del Web.

## 1.1 Tipi di Rete per Dimensione

Descrizione dei tipi di rete per estensione fisica:

1. **Personal Area Network (PAN)**: Connessione personale, ad esempio una Bluetooth.
2. **Local Area Network (LAN)**: Reti circoscritte ad un'area limitata, utilizzano cavi o mezzi wireless per connettersi e sono solitamente proprietarie.
3. **Metropolitan Area Network (MAN)**: Rete di calcolatori che non supera mai l'estensione fisica indicativa di una "città".
4. **Wide Area Network (WAN)**: Reti che cercano di interconnettere host separati da distanze geografiche tramite cavi in fibra ottica o ponti radio.

## 1.2 Gestione di Internet come Rete di Reti

Ogni rete locale si espone alle altre reti grazie agli **Internet Service Provider (ISP)**. Come possiamo immaginare però non connettiamo ogni punto terminale ad ogni altro punto terminale perchè provocherebbe un'alta complessità da un punto di vista di collegamenti. Di conseguenza saranno presenti dei **Global ISP** che si occupano della gestione del route della comunicazione. Non esiste un solo **Global ISP** ma **molteplici** e questi si connettono tra loro con dei **Peering Point** su punti d'incontro tra le reti detti **Internet Exchange Point (IXP)**.

**Content Provider Network** Idealmente si segue lo schema descritto sopra, ma a volte grandi aziende (Amazon, Google) si posizionano trasversalmente a questa organizzazione descritta, in modo tale da gestire la propria rete per avvicinare i propri servizi all'utente finale.

## 1.3 Reti e Mezzi d'Accesso

Descriviamo questi elementi caratterizzanti dell'accesso ad una rete:

1. **Rete d'Accesso**: Rete che connette fisicamente un sistema terminale al **edge router**, ossia il primo router da sorgente a destinazione.
2. **Mezzi d'Accesso**: Come passiamo fisicamente dal calcolatore al edge router:
  - (a) **Cablati (Vincolati)**: Doppino, coassiale, fibra.
  - (b) **Wireless (Non Vincolati)**: Onde Radio, microonde, infrarossi.

### 1.3.1 Schema dell'Accesso DSL

Uno dei primi schemi d'accesso è stato quello Digital Subscriber Line (DSL), dove la compagnia telefonica rappresentava l'ISP. Risultava quindi necessario splittare la linea telefonica già esistendo, suddividendo il traffico in bande di frequenze diverse:

1. **Canale Telefonico**:  $0 - 4kHz$
2. **Upstream**:  $4 - 50kHz$
3. **Downstream**:  $50kHz - 1MHz$

In questo modo sarà necessario un multiplexer di accesso alla linea digitale (DSLAM) che permetta di separare i due tipi di comunicazione. Esistono **versioni diverse** di **questo schema**, riferiremo infatti allo schema **xDSL**, proprio perchè è possibile migliorare le caratteristiche della connessione ad esempio basandosi su una connessione in **fibra** tra DSLAM ed ISP, ad esempio:

1. **Fibre To The Cabinet (FTTC)**: Standard VDSL e VDSL2.
2. **Fibre To The Derivation Point**.
3. **Fibre To The House (FTTH)**: Caratterizzato dai seguenti elementi:
  - (a) Fibra ottica fino all'interno delle abitazioni.
  - (b) Fibra uscente dalla centrale locale con terminatore ottico di linea **OLT** viene condivisa da diverse abitazioni.
  - (c) Il terminatore ottico di rete **ONT** invece è connesso ad uno splitter tramite fibra dedicata, questo permette la gestione fino al centinaio di abitazioni.
  - (d) L'**OLT** si connette al router dell'ISP e tramite questo ad Internet.

### 1.3.2 Tipi d'Accesso - Cavo, Aziendale, Domestico, Mobile e Satellitare

Descriviamo queste tipologie di accesso alla rete:

1. **Accesso Via Cavo**: Il segnale ottico viene convertito in segnale elettromagnetico e inviato sulle linee di cavi coassiali per la distribuzione del segnale alle varie abitazioni. La velocità potrebbe degradare a causa della distanza o della congestione del canale condiviso tra più abitazioni.
2. **Accesso Aziendale**: Nelle aziende i sistemi sono collegati al router di frontiera con una LAN.
3. **Accesso Domestico**: Utilizzate tecnologie Ethernet o WiFi per creare LAN domestiche.
4. **Accesso Mobile**: Infrastruttura radio mobile definita dagli ISP, può subire degrado di prestazioni a causa di distanza oppure ostacoli.
5. **Accesso Satellitare**: Accesso permesso da trasmettitori terrestri che si connettono a satelliti geostazionari (GEO) oppure satelliti a bassa quota (LEO). Questo sistema risulta essere il più duttile e non richiede complesse installazioni, ma potrebbe risultare inferiore da alcuni punti di vista come la latenza più alta.

## 1.4 Nucleo della Rete - Commutazione Circuito vs Pacchetto

Come determiniamo il percorso effettuato da un messaggio? Come trasferiamo dalla porta di uscita a quella di ingresso di due calcolatori? A queste domande risponde il tipo di commutazione.

**Commutazione di Circuito** Si instaura un cammino tra gli host in comunicazione e tutte le risorse del canale sono dedicate a loro. Questo causa un utilizzo esclusivo del canale e conseguente spreco di risorse. Per poter permettere molteplici comunicazioni utilizzando questo tipo di commutazione esistono due politiche:

1. **Frequency Division Multiplexing (FDM)**: Divido il canale in bande di frequenze e le dedico alle diverse comunicazioni.
2. **Time Division Multiplexing (TDM)**: Dedico tutto il canale alla coppia di host ma per una finestra di tempo limitata.

**Commutazione di Pacchetto** Il flusso dati di una comunicazione viene suddiviso in **pacchetti**, e gli viene assegnato un **header** che mantenga dei metadati riguardanti il flusso dati originale. Questo tipo di instradamento segue queste fasi:

1. **Suddivisione in pacchetti ed assegnamento header.**
2. **Instradamento** del singolo **pacchetto indipendentemente** dagli altri. Pacchetti di flussi dati diversi possono condividere i canali di comunicazione.
3. Fase di **Store and Forward**, ossia prima che il commutatore (router) consegni i pacchetti dovrà prima aspettare che siano del tutto arrivati. Il tipo di Multiplexing quindi è statistico e non prefissato. Seguendo questa politica potremmo incorrere anche in problemi di **perdita di pacchetti** in base al tipo di gestione del buffer di ciascun commutatore. Non abbiamo quindi alcuna garanzia sulle prestazioni utilizzando questa tecnica.

**Confronto Caratteristiche Commutazione Circuito/Pacchetto** Elenchiamo pro e contro di ciascuna tecnica:

1. **Commutazione Circuito - Pro:**
  - (a) Prestazioni garantite.
  - (b) Tecnologie di switching efficienti.
  - (c) Tariffazioni per ISP semplici da definire.
2. **Commutazione Circuito - Contro:**
  - (a) Segnalazione e configurazione delle tabelle di switching.
  - (b) Poca ottimizzazione del uso di risorse.

### 3. Commutazione Pacchetto - Pro:

- (a) Utilizzo ottimale delle risorse.
- (b) Non richiede fase di setup e tabelle di switching predefinite.

### 4. Commutazione Pacchetto - Contro:

- (a) Tecnologie di inoltro non efficienti.
- (b) Ritardi variabili.
- (c) Rischio di congestione.

## 1.5 Metriche - Prestazioni della Rete (Latenza, Throughput, Prod. Rate/Ritardo)

Descriviamo due metriche fondamentali delle prestazioni della rete:

1. **Latenza:** Tempo richiesto dal primo bit partito dalla sorgente fino all'arrivo a destinazione. Questa definizione è ideale, infatti dipende anche dal tipo di commutazione, quella di pacchetto con Store & Bound introduce altre difficoltà. Nello specifico le **cause specifiche di ritardo nella commutazione di pacchetto** sono:

#### (a) Ritardo di Elaborazione del Nodo

- i. Controllo errori sui bit.
- ii. Determinazione porta di uscita.

#### (b) Ritardo di Accodamento

Componente di ritardo più complessa da stabilire, si basa su una **condizione di stabilità** ossia un rapporto  $\rho = \frac{L \cdot a}{R}$  dove  $a$  è velocità media di arrivo,  $R$  velocità di trasmissione ed  $L$  lunghezza media dei pacchetti in bit. La condizione  $\rho < 1$  è detta **condizione di stabilità**.

- i. Attesa di trasmissione.
- ii. Dipende da intensità e tipo di traffico.

#### (c) Ritardo di Trasmissione L/R

Tempo impiegato dal router per trasmettere un pacchetto sul link, con  $R(bit/sec)$  indichiamo il rate, ossia la velocità di trasmissione del collegamento mentre con  $L(bit)$  la lunghezza del pacchetto. Il ritardo sarà quindi  $d_{trasm} = \frac{L}{R}$ . Quindi rappresenta il ritardo di passaggio nel commutatore.

#### (d) Ritardo di Propagazione

Quanto un bit ci mette per spostarsi fisicamente, rappresenta il ritardo di passaggio nel link.

Il ritardo complessivo è calcolato come somma di tutti i ritardi calcolati sopra.



2. **Throughput** Rappresenta la capacità effettiva di un link da host  $A$  verso host  $B$ , difficile da definire formalmente, ma diamo questi due riferimenti:

- (a) **Throughput Istantaneo**: Velocità in *bit/sec* a cui host  $B$  riceve in ogni istante.
- (b) **Throughput Medio**: Rapporto tra quantità  $L$  di dati trasferiti e il tempo  $T$  impiegato per il loro trasferimento, ovvero  $\frac{L}{T}$ .

Un potenziale bottleneck influenzerebbe pesantemente questa metrica, in caso quindi di più di un collegamento  $q_i$  il calcolo sarà  $\min(q_1, \dots, q_n)$ . Il Throughput quindi è differente dalla velocità di trasmissione del collegamento che si pone come upper bound alla metrica del Throughput.

3. **Prodotto Rate per Ritardo** Il prodotto rate per ritardo rappresenta il "volume" del collegamento, ossia il massimo numero di bit che possono riempire il collegamento ad un certo istante.

## 1.6 Modello Stratificato

Lo schema base di gestione delle reti è di tipo **stratificato**, ossia ciascun livello ha le proprie caratteristiche e responsabilità seguendo i due criteri di:

- 1. **Separation of Concern**: Divisione di interessi e responsabilità tra i livelli.
- 2. **Information Hiding**: Si espone solo l'interfaccia di ciascun livello ma non la sua implementazione.

Quindi ogni livello fornisce **servizi** al livello superiore e scambia informazioni con i livelli adiacenti tramite **interfacce**. In questo caso anche in caso di rifattorizzazione di un livello gli altri livelli sarebbero indipendenti dalle modifiche apportate a meno di modifica dell'interfaccia. Bisogna quindi definire dei protocolli che guidino la gestione di questi livelli.

**Cosa è un Protocollo** In protocollo si specificano **sintassi** di un messaggio, la **semantica** di un messaggio e **azioni** da intraprendere in caso di ricezione di messaggio.

### 1.6.1 Stack Protocollare TCP/IP

Si compone di 5 livelli (ciascun livello sarà guidato a sua volta da protocolli):

- 1. **Livello Applicazione**: Rappresenta delle applicazioni di rete, oppure collegamenti logici e scambi di messaggi end-to-end tra due processi su host diversi.
  - (a) **Protocolli**: **ftp, smtp, http**.
  - (b) **Entità Informazione del Livello**: L'informazione a questo livello è detta **messaggio**.

2. **Livello Trasporto:** Trasferimento dati end-to-end tra processi su host remoti.
  - (a) **Protocolli:** tcp, udp.
  - (b) **Entità Informazione del Livello:** L'informazione a questo livello è detta:
    - i. **Segmento** se in **Protocollo TCP**.
    - ii. **Datagramma Utente** se in **Protocollo UDP**.
3. **Livello Rete:** Trasferimento e instradamento di **datagrammi** dalla sorgente alla destinazione.
  - (a) **Protocolli:** IP o ICMP.
  - (b) **Entità Informazione del Livello:** L'informazione a questo livello è detta **datagramma**.
4. **Livello Link:** Trasferimento dati in **frame** attraverso il collegamento tra elementi di rete vicini.
  - (a) **Protocolli:** PPP o Ethernet.
  - (b) **Entità Informazione del Livello:** L'informazione a questo livello è detta **frame**.
5. **Livello Fisico:** Trasferimento dei bit di un frame sul mezzo trasmissivo.

### 1.6.2 Stack Protocollore ISO/OSI - Cenno Storico

A differenza dello Stack TCP/IP, che era stato definito dall'IETF, l'ISO/OSI voleva essere uno standard effettivo e formale, definito invece dall'ISO. Si differenziava dall'TCP/IP perchè aggiungeva due livelli ossia Presentazione e Sessione (compresi nel livello Applicazione nel TCP/IP). Non dimostrò nessun pratico vantaggio al TCP/IP quindi viene solo usato come riferimento formale.

## 1.7 Breve Introduzione alla Sicurezza di Reti

Un software malevolo può penetrare in un host e può rendere questo host parte di una botnet che distribuisce richieste spam. Esistono varie tecniche:

1. **DDoS** (Distributed Denial of Service):
  - (a) **Bandwidth Flooding:** Inondazione di pacchetti
  - (b) **Connection Flooding:** Inondazione di richieste TCP
2. **Sniffing:** Man in the middle host C collegato ad un mezzo condiviso può ottenere copia di ogni pacchetto trasmesso tra due host A e B.
3. **Spoofing:** Man in the middle host C che immette pacchetti spacciandosi per un altro utente tra due host A e B.

**Parametri e Criteri di Sicurezza** Esistono dei metodi per difendersi da potenziali attacchi:

1. **Autenticazione:** Dimostrazione della propria identità.
2. **Crittografia:** Garantisce confidenzialità delle informazioni.
3. **Integrità ed Autenticità:** Assicurare che i dati non siano stati alterati e provengano da fonti affidabili.
4. **Controllo Accessi:** Limitare l'uso di risorse in base ai ruoli.
5. **Utilizzo di Firewall:** Filtrare il traffico tramite dispositivi hardware o software dedicato.

## 2 Livello Applicazione

### 2.1 Introduzione a Paradigmi, Protocolli e Tipi d'Utilizzo

Livello che definisce programmi applicativi che comunicano tra i vari host. Si seguono in genere due paradigmi di comunicazione tra host in questo livello:

1. **Client/Server:** Host server sempre attivo che attende richieste dei vari host client.
2. **Peer to Peer:** Tutti i client alla pari, spesso utilizzato nella condivisione di file.

**Protocolli** La comunicazione si basa su protocolli aperti (HTTP, SMTP) oppure proprietari. Una web app ad esempio si basa sul protocollo HTTP, invece un client di posta elettronica può utilizzare il protocollo SMTP.

**Socket e API** Ogni processo, essendo che comunicherà con **processi di altre macchine**, dovrà essere **identificato** dall'**IP della macchina** (da 32 bit) e dalla **porta dedicata al processo** (da 16 bit). Questo avviene grazie all'astrazione dei socket, che utilizzando IP e porta permettono ai processi di scrivere/leggere da ciascun lato.

**Caratterizzazione di un Applicazione** Ciascuna applicazione dovrà scegliere un protocollo da utilizzare in base alla sua caratterizzazione ed in base a specifici criteri:

1. **Throughput:** Tasso effettivo di trasferimento dati, ad esempio un client di mail non avrà bisogno di un alto Throughput.
2. **Integrità dei Dati:** Richiesta d'integrità del flusso dei dati, ad esempio una stream potrebbe non richiedere un'integrità assoluta.
3. **Sensibilità ai Ritardi:** Richiesta di una bassa latenza, ad esempio un videogioco multiplayer potrebbe richiedere una bassa latenza.

L'analisi di questi criteri in relazione all'applicazione permette la scelta del protocollo di trasporto (livello inferiore), quindi nel nostro caso se TCP (orientato allo stream) o UDP (orientato al messaggio).

## 2.2 TCP vs UDP nel Contesto del Livello Applicazione

Si sceglie un protocollo del livello trasporto in base alle caratteristiche dell'applicazione in questione. Descriviamone le caratteristiche:

### 1. TCP

- (a) Orientato allo stream.
- (b) Necessario un primo setup tra client e server, questo ha un **costo** in termini di **latenza**.
- (c) Utilizzo di un buffer limitato d'appoggio per lo stream.
- (d) Nessuna garanzia su proprietà di **Timing** e **Throughput**.
- (e) Attualmente utilizzato in diversi contesti.

### 2. UDP

- (a) Orientato al messaggio.
- (b) Non necessario alcun setup.
- (c) Nessuna garanzia su proprietà di **Timing**, **Throughput**, **Trasferimento Dati**, **Controllo Flusso**, **Ampiezza Minima di Banda**.
- (d) Si accettano potenziali perdite di pacchetti.

### 2.2.1 API e Socket

**API - Application Programming Interface** Un API è un insieme di regole che un programmatore deve rispettare per utilizzare delle risorse. Se un processo a livello applicazione vuole comunicare un messaggio ad un altro host allo stesso livello allora dovrà interagire con il sistema operativo che implementa i sottostanti livelli dello stack TCP/IP attraverso un'interfaccia.

**Interfaccia Socket** API che viene utilizzata da interfaccia tra gli strati di applicazione e trasporto, è l'API di Internet per eccellenza. Dato che un host abbiamo detto essere individuato grazie ad un IP da 32 bit, identificheremo un processo e la relativa socket con la porta da 16 bit.

Per fare un esempio di API basato su uno dei protocolli del livello trasporto citati prima, mostreremo una bozza di setup tra client e server in TCP e qualche pseudocomando:

#### 1. Client:

```
1 //Apertura Connessione
2 connection TCPOpen(IPaddress, int)
3
```

## 2. Server:

```
1 //Binding
2 int tcpbind(int port)
3 //Attesa Richieste
4 connection TCPaccept(int)
5 //Spedire Dati
6 void TCPsend(connection, data)
7 //Ricevere Dati
8 data TCPreceive(connection)
9 //Chiusura Connessione:
10 void TCPclose(connection)
11
```

**Meccanismi di Sicurezza in TCP/UDP** Nativamente questi livelli non implementano alcun meccanismo di sicurezza, dunque è stato aggiunto un layer con un socket TLS (Transport Layer Security) che garantisce proprietà come cifratura o integrità dei dati.

## 2.3 Protocollo HTTP - HyperText Transfer Protocol

Protocollo del livello Applicazione che risulta essere **generico**, **stateless** e **basato sulla tipizzazione** che non deve per forza essere HTML, si negozia infatti il tipo di contenuto.

Inizialmente nato per le pagine HTML, attualmente utilizzato anche per la comunicazione tra applicazione tramite API REST che permettono esposizione di servizi.

Dati un client ed un server, il client tramite un **browser** ottiene una **pagina HTML** e tante **risorse dereferenziabili** con **URL**.

**URI - Uniform Resource Identifier** Permette l'identificazione di una risorsa:

1. **URN - Uniform Resource Name:** Sottoinsieme dell'URI che deve identificare univocamente una risorsa anche quando questa cessa di esistere.
2. **URL - Uniform Resource Locator:** Sottoinsieme dell'URI che identifica una risorsa attraverso il suo meccanismo d'accesso.

**Sintassi di un URL** Un URL segue questo schema per essere definito:

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

1. `< user > : < password >` è attualmente deprecata.
2. `< port >` è spesso omessa, dato che se non definita viene utilizzata la porta standard 80.
3. `< path >` segue lo stesso schema di un file system, dato che storicamente i server web erano statici e l'accesso ad una risorsa avveniva tramite il suo path.

Istanziando al protocollo HTTP come `< scheme >` otteniamo:

`http // < host > : < port > / < path > ? < query >`

**Client/Server HTTP** Protocollo basato su richiesta/risposta tra client e server. La **risposta** è **stateless**, ossia non dipende dalla richiesta effettuata. Il protocollo su cui si basa, del livello trasporto, è il TCP, dato che risulta necessario prima **setappare** una connessione **tra server e client**.

### 2.3.1 Versioni HTTP, Persistenza e Pipelining

**HTTP/1.0 - Connessione Non Persistente** Nella versione HTTP 1.0 la connessione non era persistente, e per ciascuna risorsa richiesta veniva aperta una nuova connessione TCP tra client e server. Questo comportava un overhead significativo, poiché ogni richiesta (ad esempio per immagini, fogli di stile, script) necessitava di una nuova connessione, aumentando la latenza e il carico sul server.

**HTTP/1.1 - Connessione Persistente e Pipelining** Con HTTP/1.1 è stata introdotta la **connessione persistente**: una singola connessione TCP può essere riutilizzata per più richieste e risposte tra client e server, riducendo l'overhead di apertura e chiusura delle connessioni. Inoltre, HTTP/1.1 supporta il **pipelining**, cioè la possibilità di inviare più richieste bufferizzando tramite una pipe di richieste. Tuttavia, il pipelining non è stato ampiamente adottato a causa di problemi di gestione dell'ordine delle risposte e di compatibilità.

### 2.3.2 Messaggi HTTP

In questo protocollo il messaggio può essere o una **richiesta** o una **risposta**. E' facilmente riconoscibile il tipo di messaggio già dalla sua **startline**, ossia la prima linea del messaggio.

1. **HTTP Request:** Composta da due parti:

(a) **Request Line:**

METODO URI VERSIONE\_HTTP

ad esempio:

GET /dump/page.html HTTP/1.1

(b) **Header:** Insieme di coppie (nomi-valori), che mantengono informazioni riguardanti formati preferiti, charset o encodings, ad esempio:

Accept: text/plain; text/html, image/png, application/json

Accept-Charset iso-8859-5, unicode-1-1

Durante questa fase è possibile che si instauri una negoziazione tra client e server sulla versione di HTTP da utilizzare. Dato che la comunicazione partirà sempre da una richiesta lato client, in base al confronto tra le versioni dell'HTTP in utilizzo di client e server si stabilisce quale usare.

Un messaggio di HTTP Request deve avere necessariamente una request line.

Un messaggio HTTP Response deve necessariamente avere una status line.

2. **HTTP Response:**

(a) **Status Line:**

VERSIONE\_HTTP STATUS\_CODE FRASE

ad esempio:

HTTP/1.1 200 OK

dove lo status code segue questo criterio:

1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error

(b) **Response Headers:** Informazioni relative alla risposta che non possono essere inserite nella Status Line, come ad esempio la **location della risorsa** oppure il **software** utilizzato dal **server** per handle la richiesta.

- (c) **Entity Headers:** Mantiene informazioni come:

Content-Base	URI Assoluta per dereferenziare il body
Content-Encoding	Codifica del body
Content-Language	Lingua del body
Content-Type	Tipo del body
Expires	Validità Temporale, usato nel Caching
Last Modified	Ultima modifica della risorsa, usato nel Caching

### 2.3.3 Content Negotiation

Il client conosce l'URL più generico della risorsa a cui vuole riferirsi. In base ai parametri forniti nell'header della richiesta sarà il server a dereferenziare l'URL più specifico, magari in base alle informazioni sulla lingua, il tipo di compressione o il formato.

### 2.3.4 Metodi dell'HTTP

Descriviamo le caratteristiche di ciascun metodo messo a disposizione da questo protocollo:

1. **OPTION:** Permette di listare quali siano i metodi chiamabili su una risorsa.

(a) **Sintassi:** `OPTION [RISORSA]`

2. **GET:** Permette la richiesta di una risorsa, anche in maniera **parziale** o **condizionale**.

(a) **Sintassi:** `GET [RISORSA]`

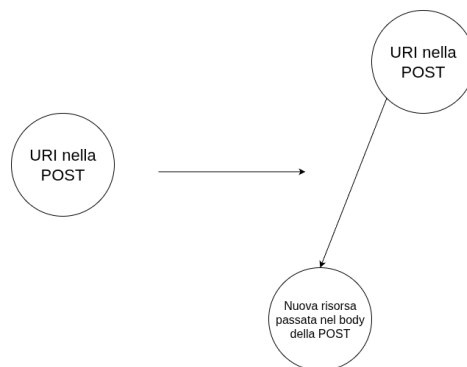
Un comune utilizzo di una GET condizionale è quella basata sull'*If-Modified-Since*, che permette di ottenere la risorsa solo se questa è stata modificata.

3. **HEAD:** Permette la richiesta solo della *response line* e dell'*header* di una risorsa, senza il suo *body*.

(a) **Sintassi:** `HEAD [RISORSA]`

4. **POST:** Permette che il server accetti l'*entità* passata nel *body* della richiesta come **nuova subordinata** della risorsa identificata dall'*URI*.

(a) **Sintassi:** `POST [RISORSA]`



Sarà quindi responsabilità del server trovare un nuovo URI per la nuova risorsa.



5. **PUT/DELETE**: Permette di inserire (PUT) o cancellare (DELETE) una risorsa all'URI passato al metodo.

(a) **Sintassi**: PUT/DELETE [*RISORSA*]

In questo caso invece il client dovrà conoscere a priori l'URI della nuova risorsa che sta inserendo.

**Metodi Safe ed Idempotenti** Definiamo queste due proprietà e listiamo i metodi che le rispettano:

1. **Metodi Safe**: Non hanno effetti collaterali, dunque non modificano la risorsa, quindi i metodi: GET, HEAD, OPTION e TRACE.
2. **Metodi Idempotenti**: Hanno lo stesso effetto se eseguiti una o  $n$  volte sulla stessa risorsa, quindi i metodi: GET, HEAD, PUT, DELETE, OPTION, TRACE.

### 2.3.5 Web Caching

Immaginando di avere una rete locale, assumendo di avere  $\{host_1, \dots, host_k\}$  nella rete, cosa succede se  $host_1$  richiede una risorsa su un host remoto  $host_x$  e lo stesso viene fatto da  $host_2$  subito dopo? Questo procedimento attualmente non viene per nulla ottimizzato, di conseguenza sono state formulate politiche di caching che permettono, tramite l'utilizzo di host intermedi, di mantenere le informazioni riguardanti le richieste per non ripeterle.

1. **Server Proxy**: Un approccio di **Web Caching** infatti è quello del **Server Proxy**, che si pone tra gli host della rete locale ipotizzata ed il server remoto. Se il server proxy caches infatti le richieste allora il secondo host potrebbe acquisire quelle informazioni senza effettuare richieste al server remoto ma solo richiedendole al server proxy stesso.
2. **User Agent Cache**: Un metodo alternativo è quello che si basa sul mantenimento in locale da parte del browser delle risorse visitate da un utente.

### 2.3.6 Cookies

Il protocollo HTTP è per definizione stateless, di conseguenza risulta necessario un meccanismo che permetta di mantenere uno stato utente riguardante la sua navigazione su una applicazione web. Vengono quindi utilizzati per autenticazione, ricordare il profilo utente e creare sessioni su un protocollo stateless.

Si vuole quindi avere la possibilità di numerare e far riconoscere gli utenti sul sito presentando ogni volta un cookie. Il meccanismo nello specifico si basa su questi passaggi:

1. **Riga di intestazione nel messaggio di risposta HTTP.**
2. **Riga di intestazione nel messaggio di richiesta HTTP.**
3. **Cookie file** mantenuto dal **browser** dell'user.
4. **Database di gestione cookie del server web.**

**Cookie First/Third Party** Esistono due tipologie diverse di cookie:

1. **First Party Cookie:** Creati direttamente dal sito web che l'utente sta visitando per mantenere informazioni riguardanti lo stato.
2. **Third Party Cookie:** Memorizzano informazioni sensibili riguardanti gli utenti, sono creati e memorizzati da un dominio diverso rispetto a quello del sito web visitato.

## 2.4 Protocollo Telnet

Protocollo di terminale il cui scopo è quello di permettere l'uso interattivo di macchine remote. Chiaramente i due calcolatori devono poter comunicare a prescindere dai sistemi operativi che utilizzano, di conseguenza si utilizza un **interfaccia minima a caratteri**.

**Protocollo TELNET [RFC 854]** Questo modello standardizzato si basa su un programma client ed uno server, nello specifico si seguono questi passi:

1. Il client stabilisce una **connessione TCP** con il server.
2. Il **client** acquisisce l'**input dei tasti** e lo invia al server, successivamente **accetta i caratteri** che il **server manda indietro** e li visualizza sul terminale utente.
3. Il server accetta la connessione TCP e trasmette i dati al sistema operativo locale.

Il client si connette alla porta 23 del server e la connessione TCP persiste per la durata della sessione di login.

**NVT - Network Virtual Terminal** L'obiettivo di TELNET è quello di poter operare con quanti più calcolatori diversi e di conseguenza sistemi operativi eterogenei. Di conseguenza si definisce un **terminale virtuale** che permette la conversione dei comandi dei vari terminali locali e viceversa, creando una vera e propria interfaccia comune di riferimento. Si definiscono quindi dei set di caratteri universali per trasformare quelli delle macchine locali in comunicazione tra loro.

**SSH - Secure Shell** Lo standard attuale per comunicazioni tra macchine remote via terminale è SSH, ossia un architettura di protocolli nata per sostituire TELNET ed i suoi problemi di sicurezza. Uno dei servizi principali di SSH è quello di tunneling, infatti grazie al port forwarding possiamo mettere in comunicazione due processi su due macchine diverse mappando le loro porte a quella di SSH, lasciando quindi viaggiare la comunicazione tra i due processi tramite il tunnel SSH.

## 2.5 Protocollo SMTP - Simple Mail Transport Protocol

Uno dei protocolli applicativi più antichi, si basa su due "protagonisti":

1. **User Agent:** Client dell'user che permette l'editing e l'invio di messaggi di posta.
2. **Mail Server:** Server dei provider che devono archiviare i messaggi ricevuti, dato che il tipo di comunicazione tra i vari User Agent è asincrona, di conseguenza dovranno immagazzinare in una coda i messaggi in uscita.

**Indirizzo Mail - Sintassi** Solitamente un indirizzo è indicato con:

local-part @ domain-name

dove con *local-part* si indica la specifica "cassetta" dell'user nel mail server, mentre *domain-name* indica l'effettivo mail server.

**Spooling dei Mail Server** I Mail Server seguono una tecnica di spooling, ossia tentano di inviare i messaggi di posta settando una connessione TCP con la macchina destinazione. Se la connessione viene aperta e il trasferimento va a buon fine allora il Mail Server cancella la copia della mail in locale. Altrimenti, se qualcosa dovesse andare storto ritenterebbe la connessione TCP in maniera periodica fino ad un certo intervallo di tempo, se viene superato allora si scarta il messaggio, notificando il mittente della condizione.

**Gestione Alias dei Mail Server** Gestione a puntatori di indirizzi di casella postale, ne esistono di due tipi:

1. **Uno a Molti:** Un singolo indirizzo alias riferisce a più indirizzi mail reali.
2. **Molti a Uno:** Un singolo indirizzo mail reale ha molti indirizzi alias che lo riferiscono.

**Protocollo SMTP [RFC5321]** Si stabilisce una connessione TCP sulla porta 25, si segue una politica PUSH da parte di ogni client fino a quello precedente al destinatario. Quindi dal primo user fino all'ultimo Mail Server tutti agiranno da client nei confronti del server destinatario successivo. Tre fasi:

1. Handshaking a livello applicativo.
2. Trasferimento messaggi in coda.
3. Chiusura.

**Comandi SMTP** Questo standard definisce specifici comandi:

1. HELO < client identifier >
2. MAIL FROM: < reverse-path > < CRLF >
3. RCPT TO: < forward-path > < CRLF >
4. DATA
5. QUIT

**Formato Messaggi SMTP** I comandi sono formato ASCII, mentre le risposte sono ASCII a 7 bit. Nello specifico, il messaggio è così composto:

1. **Header:** Contiene le linee d'intestazione, come ad esempio *To*, *From*, *Subject*, che **non** corrispondono ai parametri passati ai comandi SMTP citati prima.
2. **Body:** Contiene il messaggio effettivo, composto da caratteri ASCII a 7 bit.

**MIME - Multipurpose Internet Mail Extension** Fino ad ora i body erano composti solo da caratteri ASCII a 7 bit. Si definisce quindi una codifica per trasferimento media e testo non-ASCII. Si aggiungono quindi dei parametri nelle intestazioni per dichiarare l'utilizzo del MIME nel body.

## 2.6 Protocolli POP3/IMAP

Fino ad ora abbiamo descritto il protocollo SMTP che pusha l'informazione fino all'ultimo mail server. Essendo però un tipo di comunicazione asincrona, sarà il client del destinatario a pullare il messaggio. Si definiscono quindi i protocolli POP e IMAP.

1. **POP3:** Il client apre una connessione TCP sulla porta 110, si autentica con utente e password e richiede l'elenco dei messaggi e li preleva uno alla volta.
2. **IMAP:** Più feature di POP3, permette la gestione logica in folder dei messaggi memorizzati sul server e permette di estrarre solo alcuni componenti dei messaggi.
3. **HTTP:** In alternativa, anche HTTP può effettuare richieste ai Mail Server tramite browser.

## 2.7 DNS - Domain Name System

Vogliamo avere la possibilità di dereferenziare nomi e non indirizzi numerici, dato che questi ultimi risultano essere più difficili da ricordare. La domanda quindi è come **associare indirizzi IP a nomi**. Inizialmente la gestione di queste associazioni era statica, ma attualmente risulta impossibile utilizzare questa politica, date le dimensioni di Internet.

**Architettura Centralizzata vs Distribuita** Potremmo immaginare un unico server con relativo unico database a cui riferirsi per risolvere queste associazioni. Questo però ci vincolerebbe ad una **pesante dipendenza** da quest'ultimo. Questo dovrebbe gestire miliardi di query al giorno, ed in caso di fallimento del server si bloccherebbe tutto Internet. Si preferisce quindi un architettura distribuita.

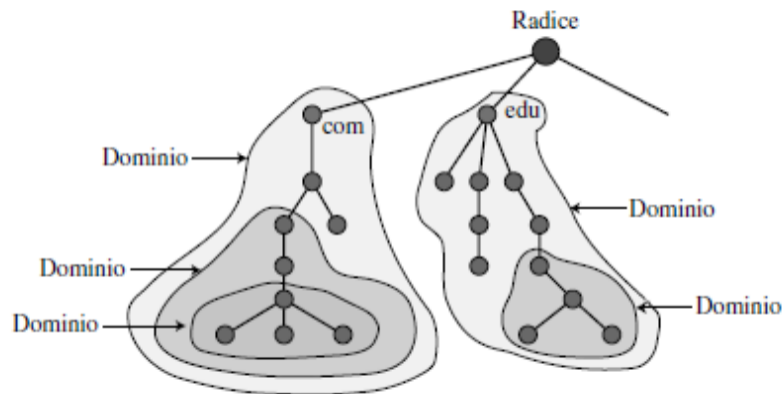
**Composizione Essenziale del DNS** Si compone di tre elementi fondamentali:

1. Uno **schema di assegnazione** dei **nomi** gerarchico basato sui domini.
2. Un **database distribuito** contenente nomi e corrispondenti IP.
3. Un **protocollo** per la **distribuzione delle informazioni** tra **name servers**:
  - (a) **Name Servers:** Comunicano per risolvere nomi.
  - (b) **Porta:** Utilizzano porta 53 in UDP (sono a volte TCP).

**Servizi Offerti dal DNS** Elenchiamo i servizi offerti dal DNS:

1. Risoluzione nomi (hostnames) in indirizzi IP.
2. **Host Aliasing:** Più alias possono dereferenziare lo stesso hostname, e di conseguenza, lo stesso indirizzo IP.
3. **Distribuzione Carico:** In caso di potenziale carico elevato assegnato ad un server, si fa corrispondere un singolo hostname a più IP per evitare problemi di sovraccarico su una macchina specifica.

**Struttura Gerarchica di Gestione Domini** Lo spazio dei nomi è definito da una **struttura gerarchica ad albero** dove ogni nodo è identificato da un **etichetta**, ossia un **nome di dominio**.

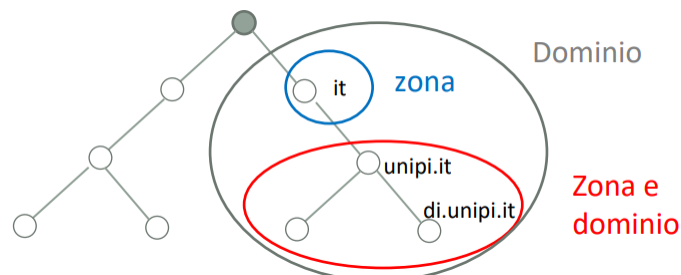


I domini top-level (.com, .edu, .gov, .net, .org) vengono gestiti dallo IANA.

### 2.7.1 Gerarchia dei Name Servers

Avendo definito il **DNS** come un **database distribuito** in una **gerarchia** di più **name servers**, allora bisogna definire cosa sia un **name server**, ossia un programma che gestisce la conversione da nome di dominio ad indirizzo IP.

**Zone e Domini di Name Servers** Tutte le informazioni per le risoluzioni di nomi sono ripartite su più name server, dunque ciascuno di essi immagazzina le info relative alla propria **zona** ed i riferimenti ai **name servers** dei **domini di livello inferiore**.



**Ruoli dei Server nella Gerarchia** Elenchiamo tutti i ruoli e le loro responsabilità:

1. **Root Name Server:** Responsabile dei record della zona radice, riconosce tutti i domini TLD (Top Level Domain) e conosce il server TLD che li risolve, quindi restituisce le info sui name server TLD. A questo ruolo corrispondono circa 13 indirizzi IP.
2. **Top Level Domain Server:** Mantiene le info dei nomi di dominio che appartengono ad un certo TLD, e restituisce informazioni sui name server di competenza dei sottodomini.
3. **Authoritative Name Server:** Responsabile di una certa **zona**, memorizza nome ed indirizzo IP di un insieme di host di cui può effettuare la traduzione. Questo ruolo può essere suddiviso in altri due sottoruoli:
  - (a) **Primary Authoritative Name Server:** Mantengono il file di zona.
  - (b) **Secondary Authoritative Name Server:** Ricevono il file di zona e offrono il servizio di risoluzione.

A questa gerarchia si **aggiunge** un **ulteriore ruolo**, che **non appartiene** propriamente alla **gerarchia**, ma che si occupa da prima interfaccia nel caso in cui un client (es. Browser) tenti di riferire ad un nome:

1. **Local Name Server:** Un server appartenente ad un ISP (Internet Service Provider) che assume il ruolo di proxy, per fare prima un controllo nelle tabelle locali per cercare la risoluzione del nome richiesto dal client, e solo successivamente, in caso di fault, richiede la query effettiva al DNS, nello specifico ad un Root Name Server.

**Query Ricorsiva/Iterativa** Esistono due tipi diversi di risoluzione dei nomi:

1. **Query Ricorsiva:** Il Local Name Server richiede una conversione completa di nome.
2. **Query Iterativa:** Viene restituito al client o al Local Name Server il riferimento al server successivo da contattare per completare la risoluzione di nome.

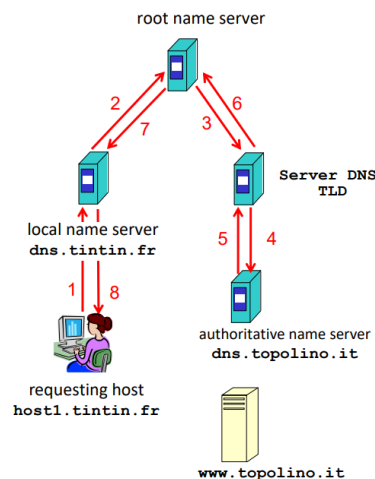


Figure 1: Query Ricorsiva

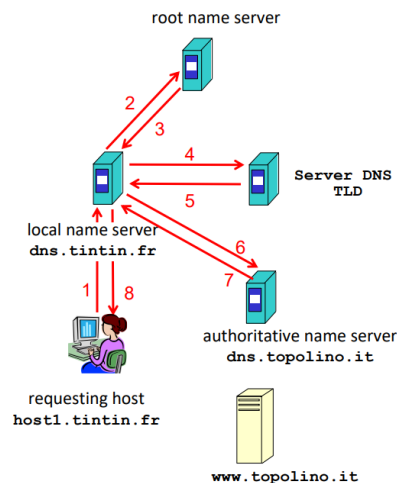


Figure 2: Query Iterativa

**Caching DNS** Nel DNS si utilizza molto il caching per ridurre richieste che rimbalzano continuamente tra server. Solitamente quindi si mantiene in cache di server DNS il dato in cache in modo non autorevole, e i server invalideranno queste informazioni dopo un periodo fissato. Un altro approccio potrebbe essere quello di mantenere in cache direttamente degli indirizzi IP di server DNS TLD (Top Level Domain), evitando di riferirsi più volte ai server radice.

**Record DNS** Un DNS è un database distribuito di *resource records* (RR) che seguono la seguente sintassi:

RR format : (name, value, type, ttl)

dove gli elementi vengono così interpretati:

1. **TTL**: Tempo residuo del record in cache.
2. **Name** e **Value**: Dipendono da type.
3. **Type**: Esistono vari tipi:

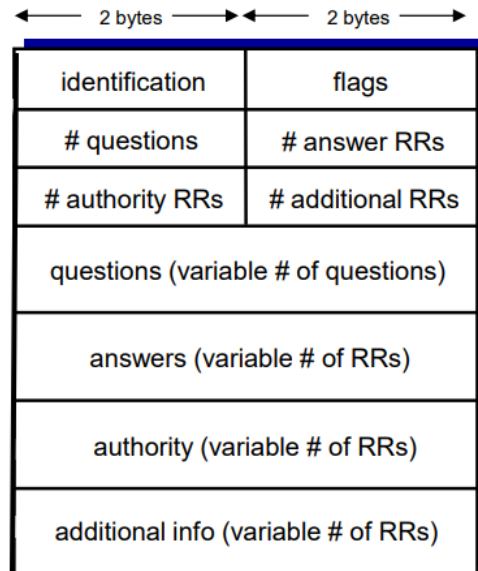
Type	Name	Value
A	hostname	indirizzo IP
CNAME	hostname (sinonimo)	nome canonico dell'host
NS	nome di dominio	hostname dell'Authoritative Name Server per quel dominio
MX	nome di dominio	nome canonico del server di posta associato a name

### 2.7.2 Protocollo DNS

L'effettivo protocollo del livello applicazione che si occupa di questa risoluzione di nomi utilizza prevalentemente **UDP** sulla **porta 53**. L'applicazione è detta **resolver**, effettua polling al server DNS e dopo  $n$  richieste o richiede ad un altro server oppure informa l'applicazione con il fallimento.

**Perchè UDP e Quando TCP** Viene utilizzato il protocollo UDP per una questione di riduzione di ritardi, riduzione di overhead causato dal setup di TCP e perchè solitamente le query DNS richiedono pochi dati, quindi rispettano le dimensioni di un pacchetto UDP. A volte viene utilizzato il protocollo TCP, ma questo accade solo se è necessaria l'affidabilità del trasferimento oppure se si sfiorano i 512 byte dei pacchetti UDP.

**Messaggi del Protocollo DNS** I messaggi di query e reply seguono lo stesso formato:



Dove ogni campo indica

1. **Identification:** Campo di 16 bit per l'identificazione di una query, la risposta ad una query usa lo stesso identification id.
2. **Flags:** Indicano se query o reply, se desiderata o disponibile la ricorsione e se la risposta è autorevole.
3. **Questions:** Nome richiesto e tipo di query.
4. **Answers:** RR (*resource records*) in risposta alla query.
5. **Authority:** Record per i server di competenza.
6. **Additional Info:** Informazioni utili aggiuntive.

**DNS e Sicurezza** Dato il funzionamento molto decentralizzato del DNS, esistono vari attacchi che sfruttano questa sua caratteristica:

1. **DNS Cache Poisoning:** Un attaccante invia risposte DNS falsificate ad un DNS Server o ad un client, avvelenandone la cache.
2. **Man-in-the-Middle Attack:** Vengono intercettate le comunicazioni tra un client ed un server DNS, ed in questo modo viene indirizzato un obiettivo verso host malevoli.
3. **DNS Hijacking:** Si restituiscono risposte non corrette alle query DNS, reindirizzando il client verso siti malevoli alterando le impostazioni DNS. Questo può essere fatto a diversi livelli, infatti può essere un Local, Router oppure Rogue Hijacking.



## 3 Livello Trasporto

L'obiettivo di questo livello è quello di creare una connessione logica tra processi applicativi su host diversi, assumono quindi di essere collegati a livello fisico, astruendo dalla reale connessione fisica di cui si occuperanno i livelli sottostanti. I protocolli di trasporto vengono eseguiti nei sistemi terminali, mantenendo la politica per cui si mantengono dumb i dispositivi intermedi e smart quelli terminali.

### 3.1 Introduzione al Livello Trasporto

In questo sottocapitolo definiamo concetti base del livello trasporto.

**Servizi del Livello Trasporto** Elenchiamo i servizi offerti da questo livello:

1. Offre servizi al livello applicazione, permettendo alle applicazioni di scegliere il modo in cui inviare/ricevere dati secondo uno **stile** a **messaggi** oppure a **stream**.
2. Comunica con il livello Rete sottostante, che si occupa a sua volta della consegna del datagramma all'host destinatario.

**Servizio Connection/Non Connection Based** Seguendo gli stili definiti sopra possiamo definire anche servizi orientati o meno alla connessione. Nello stile orientato alla connessione si instaura una logica di server/client dopo una fase di setup, mentre in uno stile non orientato alla connessione i dati vengono visti come messaggi indipendenti.

**Azioni del Livello Trasporto** Elenchiamo le azioni di questo livello in base a chi le esegue:

#### 1. Azioni Mittente:

- (a) Riceve un messaggio dal livello applicazione.
- (b) Determina i valori dell'header del segmento.
- (c) Crea il segmento.
- (d) Passa il segmento al livello Rete.

#### 2. Azioni Destinatario:

- (a) Riceve il segmento dal livello Rete.
- (b) Controlla i valori dell'header.
- (c) Estrae il messaggio del livello applicazione.
- (d) Smista il messaggio all'applicazione attraverso la socket.

## 3.2 TCP vs UDP e Multiplexing/Demultiplexing

I due protocolli di trasporto principali hanno caratteristiche differenti:

### 1. Protocollo TCP:

- (a) Orientato alla Connessione.
- (b) Orientato allo Stream.
- (c) Gestione della connessione.
- (d) Consegna affidabile dei dati.
- (e) Controlli di congestione e flow.

### 2. Protocollo UDP:

- (a) Senza connessione, non affidabile.
- (b) Orientato al messaggio.
- (c) Estensione del servizio *host to host* del livello rete.

Entrambi i protocolli però offrono controlli sugli errori e servizi di multiplexing e demultiplexing.

**Multi/Demultiplexing** Il livello di trasporto ha la responsabilità di smistare i messaggi ottenuti dal livello di rete tra i vari processi tramite la risoluzione di socket address. Questo risulta necessario se più processi utilizzano lo stesso protocollo di trasporto per la comunicazione su rete, cosa che accade molto spesso. Quindi abbiamo due attori duali:

- 1. **Mittente in Multiplexing:** Gestisce i dati da più socket e si occupa dell'aggiunta del header di trasporto.
- 2. **Destinatario in Demultiplexing:** Usa le informazioni nell'header per recapitare i segmenti alla corretta socket.

Nello specifico, nel demultiplexing vengono utilizzati dall'host indirizzi IP e numeri di porta per smistare il segmento verso la corretta socket dato che ogni datagramma si porta dietro un segmento del livello trasporto ed ogni segmento nell'header mantiene un numero di porta sorgente ed uno di destinazione.

**Definizione di Porta** Ogni comunicazione a livello trasporto è univocamente identificata tramite la combinazione  $IP/porta$ , una **porta** è quindi definita come un intero a 16 bit che viene assegnato ad un processo, più nello specifico è un **punto di demultiplexing** dei protocolli TCP o UDP. Esistono dei range di porte predefinite, come le *System Ports*, *User Ports* o *Dynamic Ports*. In generale è il sistema operativo ad occuparsi dell'assegnamento di porte.

**Multiplexing TCP/UDP** Esistono differenze tra i tipi di multiplexing dei due protocolli, infatti il protocollo TCP esegue il multiplexing basandosi su una quadrupla con tutte le informazioni su sorgente e destinatario ossia  $(IP_{sorg}, porta_{sorg}, IP_{dest}, porta_{dest})$ , mentre il protocollo UDP utilizza la coppia  $(IP_{dest}, porta_{dest})$ .

### 3.3 Trasporto UDP

Servizio di trasporto best effort, i datagrammi UDP possono essere persi o consegnati fuori sequenza. Questo protocollo è orientato al messaggio, quindi ogni datagramma è indipendente e di dimensioni limitate. Questo garantisce però una semplicità non indifferente, risultando vantaggioso in contesti di applicazioni time-sensitive.

**Impieghi e Caratteristiche del Trasporto UDP** E' facile e leggero da gestire, non esiste alcun controllo sulla congestione e le intestazioni non superano gli 8 byte. Viene impiegato in contesti di applicazioni multimediali, dove sono tollerate le piccole perdite, oppure in altre applicazioni come il DNS, HTTP/3 o SNMP (Simple Network Management Protocol).

**Azioni Mittente/Destinatario** Elenchiamo le azioni effettuate in base al tipo di attore:

#### 1. Azioni Mittente:

- (a) Riceve un messaggio del livello Applicazione.
- (b) Determina i valori dei campi dell'header UDP.
- (c) Crea lo user datagram.
- (d) Passa lo user datagram al livello Rete.

#### 2. Azioni Destinatario:

- (a) Riceve lo user datagram dal livello Rete.
- (b) Controlla la checksum
- (c) Estrae il messaggio di livello Applicazione.
- (d) Smista il messaggio attraverso la socket.

**Composizione di un Datagramma UDP** Descriviamo l'entità base del protocollo UDP:

numero porta sorgente	numero porta destinazione
lunghezza	checksum
payload	

Il messaggio si compone di 8 byte di intestazione, i numeri di porta per la fase di demultiplexing e la lunghezza del messaggio massima del datagramma UDP è di 65535 byte.

### 3.4 Principi di Trasferimento Affidabile

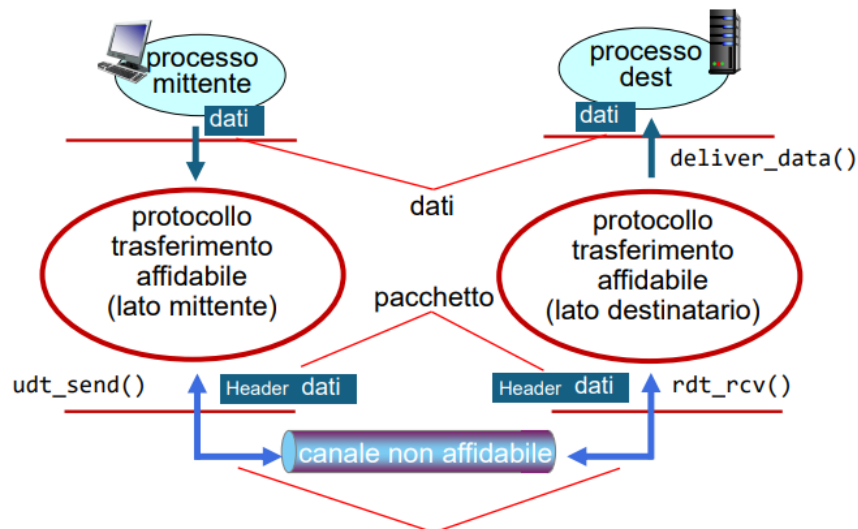
Definiamo in maniera incrementale un **protocollo generico** per il trasferimento di dati affidabile, quindi **senza errori su bit** o **perdite**. La complessità del protocollo dipende anche dalle caratteristiche del canale non affidabile su cui si sta appoggiando. In questo caso il **livello di rete** è considerato **non affidabile**, dato che può causare **perdite**, **corruzioni** o **consegne fuori ordine**.

**Interfacce per Trasferimento Affidabile** Definiamo le interfacce

1. **RDT**: Reliable Data Transfer
2. **UDT**: Unreliable Data Transfer

che comunicano grazie ai seguenti metodi:

1. *rdt\_send()*: Chiamata dall'applicazione per chiedere di consegnare i dati al processo destinatario.
2. *udt\_send()*: Chiamata da RDT per inviare il pacchetto al destinatario tramite il canale inaffidabile.
3. *rdt\_receive()*: Chiamata dal protocollo inaffidabile UDT quando il pacchetto arriva sul lato destinatario del canale.
4. *deliver\_data()*: Chiamata da RDT per consegnare i dati al livello applicazione.



### 3.4.1 RDT 1.0 - Trasferimento su Canale Affidabile

Questo schema si basa sull'**assunzione** che il **canale di rete** sottostante sia **perfettamente affidabile**. Quindi tutto il funzionamento si riduce a queste due entità che eseguono le seguenti operazioni:

1. Il **mittente** **invia** al livello di rete i dati che riceve dal livello applicazione, quindi viene effettuato l'**incapsulamento del messaggio**, componendo un **pacchetto**<sup>1</sup>.
2. Il **destinatario** **inoltra** al **livello applicazione** i dati che riceve dal livello di rete quindi viene effettuato il **decapsulamento del pacchetto** estraendo un **messaggio**.

### 3.4.2 RDT 2.0 - Trasferimento su Canale con Errori su Bit

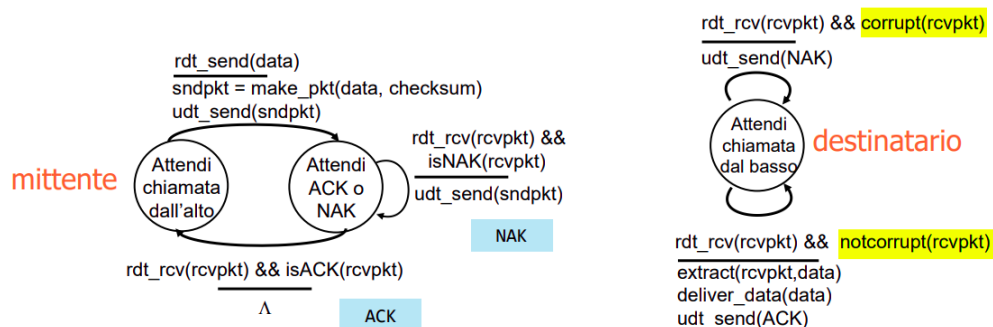
Indebolendo l'ipotesi dello schema precedente, assumiamo di poter ottenere **errori sui bit** a causa del **transito** sul **canale inaffidabile**.

**Ritrasmissione con ACK/NAK** Si utilizzano quindi riscontri positivi **ACK** per confermare la corretta ricezione e negativi **NAK** per richiedere ritrasmissione. I protocolli basati su ritrasmissione sono anche detti **ARQ** (Automatic Repeat ReQuest).

**Meccanismi Caratteristici dei ARQ** I protocolli ARQ si basano su questi meccanismi:

1. **Checksum** per rilevamento errori sui bit.
2. Meccanismo di **riscontro errore** esplicito con bit settato per **ACK o NAK**.
3. **Ritrasmissione** dei **pacchetti** ricevuti con errori.

Il tipo di protocollo ARQ che trattiamo è quello **stop and wait**, ossia il mittente invia un messaggio e poi aspetta riscontro positivo da parte del destinatario prima di inviare il nuovo pacchetto.



In questo schema però non si tiene conto di possibili corruzioni di ACK e NAK. Questo problema viene risolto nel RDT2.1 grazie ad una checksum per il controllo dell'ACK e alla ritrasmissione in caso di corruzione. Sarà necessario anche quindi un meccanismo di numerazione per capire se il pacchetto sia un duplicato o meno.

<sup>1</sup>Nel protocollo TCP si parla di segmenti, non di pacchetti, ma nel contesto di trasferimento affidabile generico parleremo di pacchetti.

### 3.4.3 RDT 2.1 - Checksum per Controllo Corruzione ACK

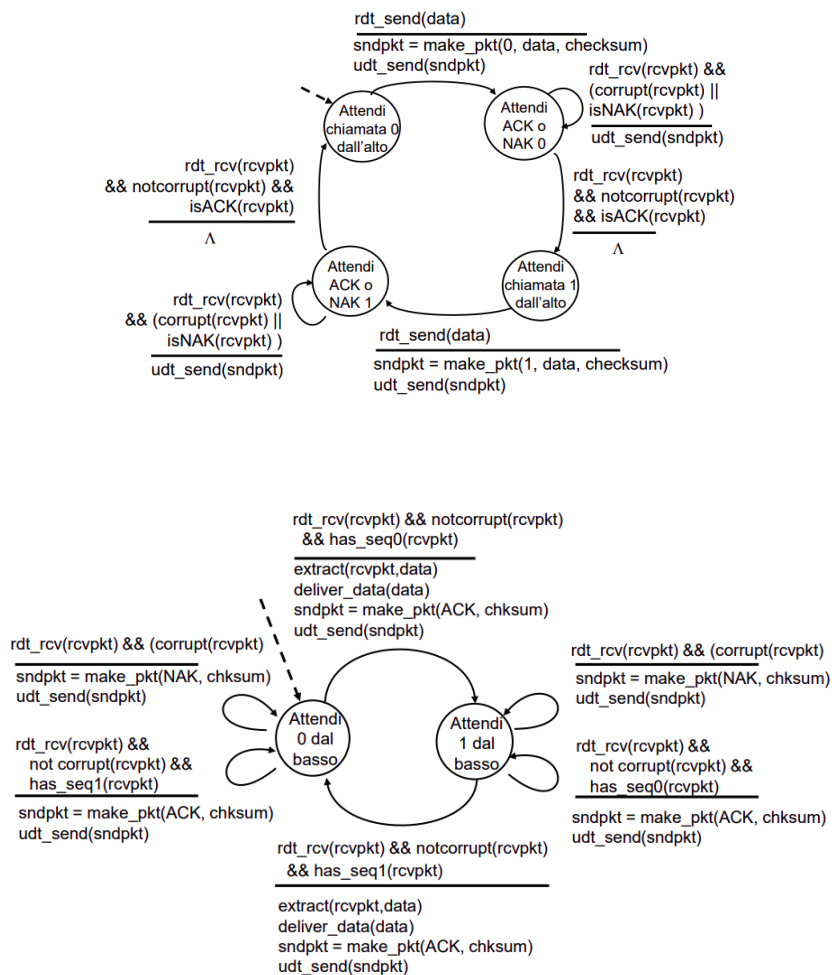
Viene aggiunto un numero di sequenza ai pacchetti lato mittente, che nel caso dello stop and wait di prima sarà quindi  $\{0, 1\}$ . Quindi i passi sono:

#### 1. Mittente:

- Aggiunta numero di sequenza al pacchetto.
- Verifica corruzione ACK/NAK tramite checksum.
- Quantità stati della macchina raddoppiati: Lo stato tiene conto della numerazione del pacchetto atteso.

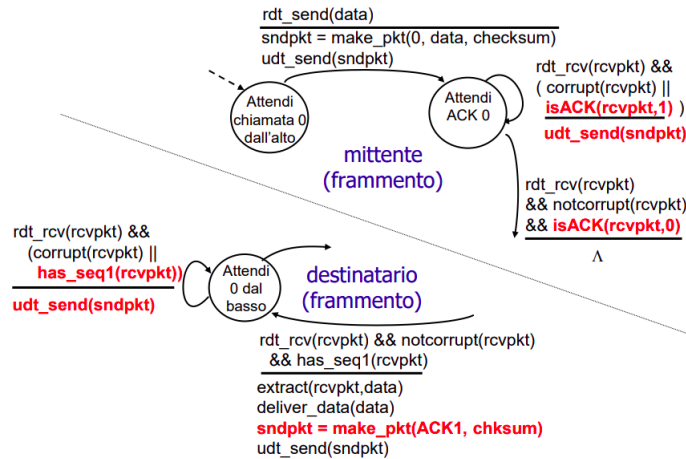
#### 2. Destinatario:

- Verifica se pacchetto ricevuto sia duplicato, grazie alla numerazione.



### 3.4.4 RDT 2.2 - Protocollo NAK-free

Il destinatario non invia NAK espliciti, ma invia gli ACK dell'ultimo pacchetto ricevuto correttamente. Quindi risulta importante includere il numero di sequenza del pacchetto da riscontrare. Quando il mittente riceve ACK duplicati, ritrasmette il pacchetto corrente.

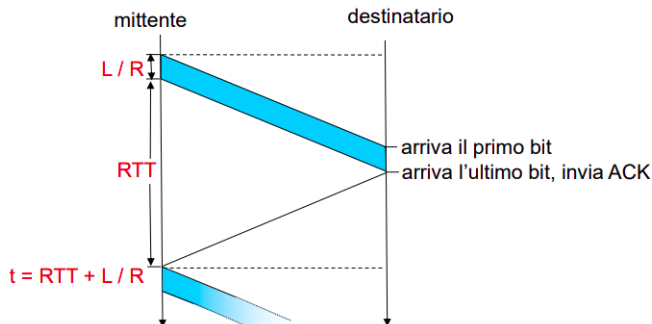


### 3.4.5 RDT 3.0 - Timer in Attesa di ACK

Se la perdita pacchetti esiste nel canale sottostante, allora il mittente potrebbe non ricevere alcuna risposta dal destinatario. Il mittente dunque aspetta ogni ACK per un massimo di tempo detto **timeout**. Se il pacchetto non è perso ma solo in ritardo, allora la ristrasmissione invia un duplicato, che sarà individuabile grazie al numero di sequenza da riscontrare.

### 3.4.6 Prestazioni e Reale Utilizzo del Canale

RDT3.0 risulta essere molto valido da un punto di vista funzionale, ma da un profilo di prestazioni notiamo che l'utilizzo del canale è molto ridotto a causa dell'attesa del mittente. Volendo quantificare l'utilizzo del canale possiamo definire questo schema con relativa formula:



$$U_{send} = \frac{L/R}{RTT + L/R} = \text{tempo utilizzato dal mittente per inviare pacchetti}$$

Questo tipo di problematiche si risolvono con approcci di **pipelining**.

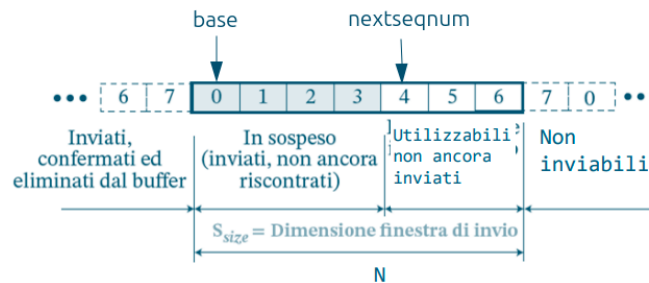
### 3.4.7 Approcci di Pipelining - Go-Back-N (GBN) e Selective Repeat (SR)

Esistono due approcci di pipelining: Go-Back-N (GBN) e Selective Repeat (SR).

**Go-Back-N (GBN)** Definiamo ogni responsabilità dei ruoli di mittente e destinatario.

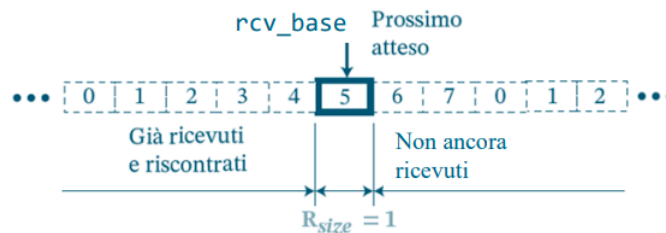
#### 1. Mittente

- (a) Il mittente può inviare fino ad  $n$  pacchetti, ossia la dimensione della sua finestra, senza ricevere ACK.
- (b) Si mantengono due puntatori sulla finestra:
  - i. *base*: primo pacchetto, il più vecchio non riscontrato.
  - ii. *nextseqnum*: prossimo pacchetto da inviare.
- (c) Possono essere riscontrati pacchetti in maniera cumulativa, scorrendo la finestra di  $n$  posizioni su cui si chiama la  $ACK(n)$ .
- (d) Il mittente ritrasmette il pacchetto e tutti quelli con un numero di sequenza maggiore di quello non riscontrato nel caso in cui scade il timer di quello puntato da *base*.



#### 2. Destinatario

- (a) Invia un ACK ogni volta che riceve un pacchetto.
- (b) L'ACK è sempre riferito all'ultimo pacchetto ricevuto in ordine.
- (c) Mantiene un solo puntatore *rcv\_base*.
- (d) I pacchetti ricevuti fuori ordine vengono scartati.



Questo tipo di approccio ha problemi di prestazioni quando il livello di rete è soggetto a numerose perdite di pacchetti.



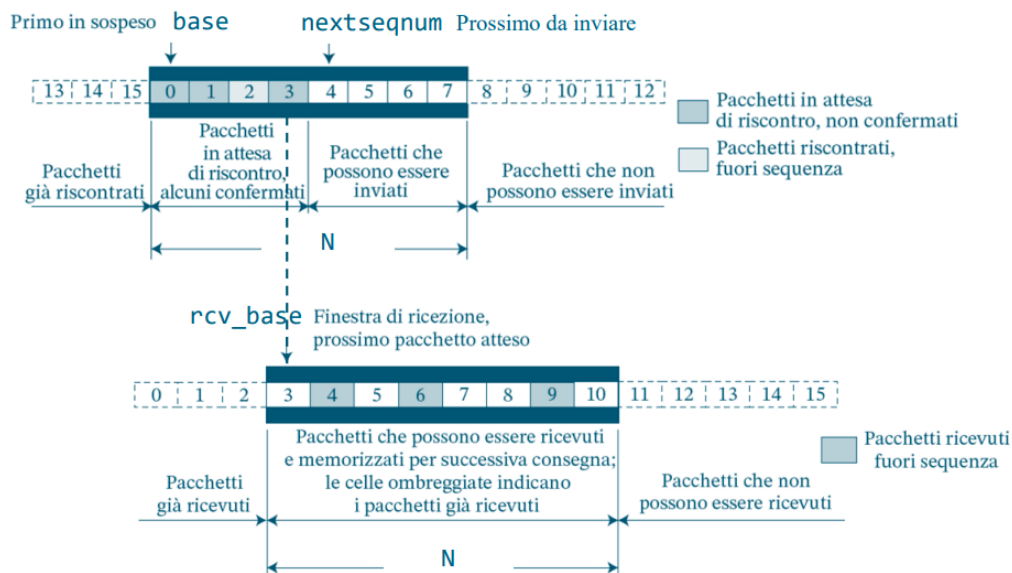
**Selective Repeat (SR)** Questo approccio utilizza ACK individuali per ciascun pacchetto ricevuto correttamente, si appoggia quindi ad un buffer da cui verranno estratti i pacchetti e consegnati in ordine all'applicazione. Nello specifico, vengono utilizzate **due finestre** di pacchetti, una **lato mittente** ed una **lato destinatario**.

## 1. Mittente

- (a) Invocazione dall'alto, se c'è un numero di sequenza disponibile allora invia il pacchetto.
- (b) In caso di  $timeout(n)$ , il pacchetto  $n$  viene ritrasmesso e resettato il timer.
- (c) La gestione di  $ACK(n)$  in  $[base, base + N + 1]$  è definita come:
  - i. Segna  $n$  come ricevuto.
  - ii. Se  $n = base$  allora si fa scorrere la finestra fino al prossimo pacchetto non riscontrato.

## 2. Destinatario

- (a) Se  $pktOk \in [rcv\_base, rcv\_base + N + 1]$  allora:
  - i. Invia  $ACK(n)$ , quindi un nuovo pacchetto è stato correttamente ricevuto.
  - ii. Se il pacchetto in questione è fuori ordine allora lo si salva nel buffer.
  - iii. Se il pacchetto è in ordine allora si effettua una consegna di gruppo di pacchetti consecutivi in ordine nel buffer e si lascia scorrere la finestra fino al prossimo pacchetto non ricevuto.
- (b) Se  $pktOk \in [rcv\_base - N, rcv\_base - 1]$  allora:
  - i. Invia  $ACK(n)$ , per indicare che il pacchetto già è stato riscontrato.
- (c) Altrimenti scarta il pacchetto.



### 3.4.8 Riassunto Meccanismi Affidabili

Listiamo una serie di meccanismi riguardanti il trasferimento affidabile con le relative caratteristiche:

- **ACK:** Indica al mittente che un pacchetto è stato ricevuto correttamente dal destinatario.
- **NAK:** Indica al mittente che un pacchetto è stato **non** ricevuto correttamente dal destinatario.
- **Ritrasmissione:** Permette al ricevitore di ricevere un pacchetto che è stato corrotto o perso in una trasmissione precedente.
- **Numeri di Sequenza:** Consente il rilevamento di duplicati lato ricevitore.
- **Checksum:** Utilizzato dal ricevitore per rilevare i bit invertiti durante la trasmissione di un pacchetto.

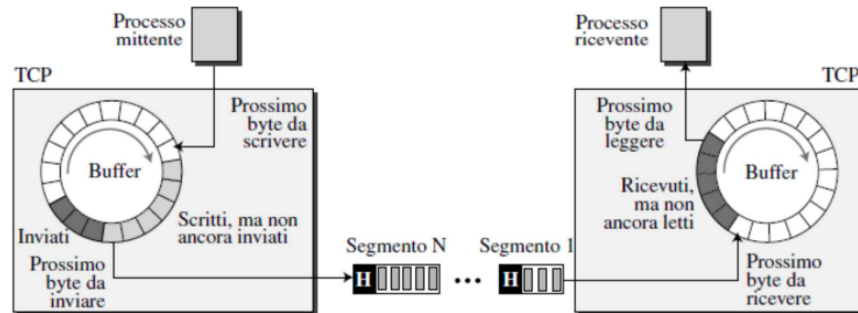
## 3.5 Protocollo TCP

Iniziamo la descrizione in verticale sul protocollo TCP.

**Proprietà del Protocollo TCP** Il protocollo TCP garantisce queste proprietà:

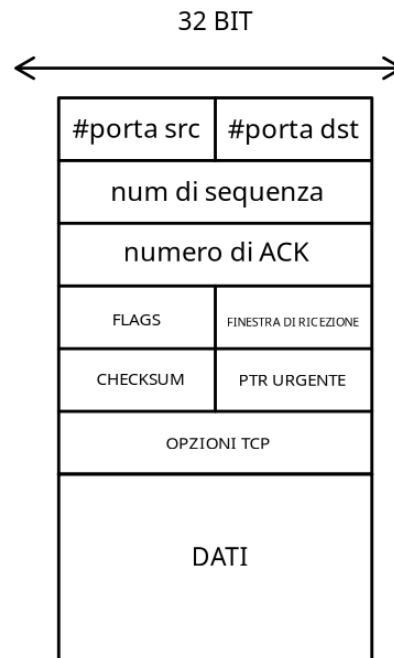
- **Orientato alla Connessione:** Si effettuano handshake prima dell'inizio della connessione, e lo stato della connessione risiede sui punti terminali della connessione e non su quelli intermedi.
- **Connessione Full Duplex:** La connessione punto a punto è per definizione bidirezionale.
- **Multiplexed:** Consente l'assegnamento di una connessione ad un determinato processo.
- **Controllo della Connessione:** Fornisce meccanismi di inizio e fine trasmissione.
- **Trasferimento di Dati Ordinato e Affidabile:** Permette il controllo di tutti i tipi d'errore quindi dati corrotti, segmenti persi, segmenti duplicati e segmenti fuori sequenza.
- **Controllo di Flusso:** Evita di spedire più dati di quanti non ne possa handle il destinatario.
- **Controllo di Congestione:** Permette di recuperare da situazioni di sovraccarico della rete.

**Bufferizzazione del Trasferimento** Il flusso di dati in byte fornito dall'applicativo viene suddiviso in segmenti TCP, e per poter permettere questo si utilizzano **due buffer** uno lato **mittente** ed uno lato **destinatario**. Quindi il flusso di byte viene partizionato in segmenti, ciascuno di essi avrà un header e viene consegnato al livello IP.



### 3.5.1 Formato Segmento TCP

Dato un flusso di **byte numerati** TCP suddivide questo stream in segmenti, mantenendo quindi un enumerazione sui segmenti in base a specifici byte, quindi un **segmento** è **identificato dal suo primo byte** di dati **nel segmento**. In questo modo il riscontro è rappresentato dall'ultimo byte ricevuto correttamente +1.



Descriviamo a pagina successiva questa struttura passo passo.

- **Porte sorgente e Destinazione**
- **Numero di Sequenza:** Si riferisce al primo byte nel flusso che caratterizza il segmento.
- **Numero di ACK:** Numero di sequenza del prossimo byte atteso, se **FLAG** = 1 allora è un ACK.
- **FLAGS di Bit Codice:** Sei FLAGS che hanno specifiche funzionalità:
  - **URG:** Campo puntatore urgente, da utilizzare se ci sono dati da trasferire in via prioritaria.
  - **ACK:** Se settata ad 1 indica che il numero di riscontro contiene dati significativi.
  - **PSH:** Se settata ad 1 causa il trasferimento dei dati immediatamente all'applicativo.
  - **RST:** Richiede il reset della connessione.
  - **SYN:** Sincronizza il Numero di Sequenza.
  - **FIN:** Se settata ad 1 indica la richiesta di chiusura della connessione.
- **Finestra di Ricezione:** Numero di byte che il destinatario<sup>2</sup> può ricevere.
- **Checksum.**
- **Opzioni TCP:** Permette la negoziazione di parametri, come ad esempio la dimensione massima di un segmento (MSS).

### 3.5.2 Gestione della Connessione TCP - Handshaking e Chiusura

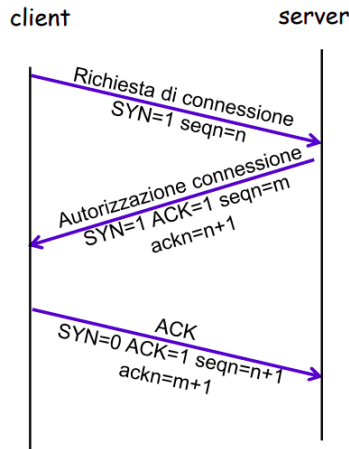
Analizziamo le fasi di setup, di effettivo trasferimento di dati e chiusura di una connessione TCP.

**Three Way Handshake** Prima di trasmettere effettivi dati mittente e destinatario si concordano sull'apertura di una connessione, stabilendo anche i parametri della connessione. I primi segmenti quindi non sono di dati effettivi, permettono anche l'inizializzazione dei due buffer, necessari per il controllo del flusso e della congestione. Al momento di arrivo del terzo segmento la connessione è instaurata.

Mostriamo a pagina successiva passo passo i dettagli dello scambio di questi 3 segmenti iniziali.

---

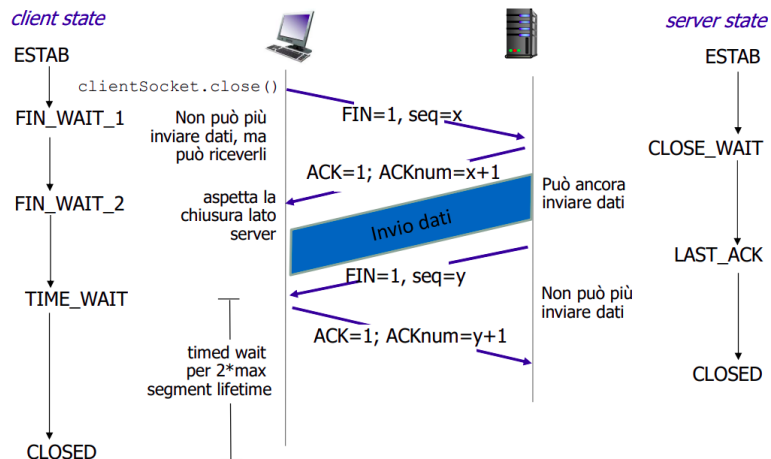
<sup>2</sup>Il destinatario in questo caso è chi sta componendo questo segmento, in veste di futuro destinatario, comunica all'altra parte quanto può ricevere nel suo buffer. Questo appiattimento dei ruoli di mittente e destinatario è causato dal fatto che dopo l'handshake i due attori sono posti allo stesso livello.



- Il **client** invia una **richiesta di connessione** ad un **server TCP**, settando come attiva la flag **SYN** e trasmettendo anche un **numero di sequenza iniziale**.
- Il server alloca i buffer e risponde con un segmento di connessione garantita detto **SYNACK**, nello specifico:
  - Con **SYN** attivo è significativo il valore di sequenza, che è il valore iniziale.
  - Anche **ACK** è attivo.
- Il client alloca il buffer dal suo lato e invia un riscontro positivo del messaggio del server.
  - Si setta **SYN = 0** per fare in modo che dopo questo terzo segmento inizi l'effettivo scambio di dati.

Quindi da questo momento in poi server e client sono sullo stesso livello.

**Chiusura di Connessione TCP** Client e server chiudono ciascuno il loro lato della connessione, inviando segmenti dove **FIN = 1**. Ciascuno risponderà ad un segmento del genere con un **ACK**.



Si noti che sulla precedente rappresentazione erano riportati degli stati di una connessione TCP. Ne esistono diversi e in particolare lo stato `TIME_WAIT` è lo stato finale in cui chi sta richiedendo la chiusura resta prima di passare alla reale chiusura. Questo stato è caratterizzato dalla presenza di un timer che lo fa terminare, e la durata di questo timer è correlata alla MSL, ossia **Maximum Segment Lifetime**. L'esistenza di questo stato garantisce l'implementazione in maniera affidabile della terminazione di una connessione in entrambe le direzioni e l'eliminazione di segmenti duplicati in rete.

**Chiusura Parziale di Connessione TCP** Se uno dei due lati lancia un segmento con `FIN = 1`, allora non potrà inviare dati ma potrà ancora riceverli mettendosi quindi in stato di `FIN_WAIT_2`.

### 3.5.3 Trasferimento Affidabile del TCP

Un segmento può essere smarrito o corrotto, quindi il TCP crea un servizio di trasferimento affidabile basandosi su questi meccanismi:

- **Checksum:** Permette di scartare segmenti corrotti.
- **Numero di Sequenza:** Enumerazione di segmenti basata sui primi byte di ciascun segmento.
- **Numero di Ricontro:** Numero di sequenza del byte che l'host sta attendendo dall'altro.
- **Ricontro Cumulativo:** Si riscontra fino al primo byte mancante nel flusso.
- **Timer.**

**Eventi Lato Mittente** Descriviamo il comportamento del mittente:

- Riceve i dati dall'applicativo.
- Incapsula di dati in segmenti e li numera con la logica a byte definita prima.
- Avvia il timer di ritrasmissione RTO solo se non è già assegnato a qualche altro precedente segmento.
- Viene effettuata una **ritrasmissione** se:
  - Scade un timeout: Ritrasmette il segmento che non è stato riscontrato, riavviando il timer.
  - Riceve tre **ACK** duplicati: Questo è indice di perdita del segmento successivo all'ultimo riscontrato, quindi viene effettuata una **fast transmission** prima della scadenza del timer.

I segmenti ottenuti fuori sequenza possono essere temporaneamente memorizzati dal destinatario, ma la loro gestione dipende dall'implementazione del TCP a cui stiamo facendo riferimento.

**Eventi Lato Destinatario** Descriviamo il comportamento del destinatario:

- Tutti i segmenti inviati contengono ACK.
- Se il destinatario **non ha dati da inviare** e **riceve un segmento in ordine** allora ritarda l'invio di ACK di  $500ms$
- Se il destinatario **riceve il segmento atteso** e il **precedente non è stato riscontrato** allora invia immediatamente un ACK.
- Se il destinatario riceve un **segmento fuori sequenza, mancante in una sequenza** oppure **duplicato** allora invia immediatamente un ACK.

### 3.5.4 TCP RTT, RTO, Stime Timeout e Finestre di Ricezione

Il tempo di timeout **RTO** è fondamentale per il funzionamento del TCP e non può essere né troppo breve né troppo lungo, va quindi accuratamente stimato. Ci basiamo quindi su due misure:

- **SampleRTT**: Misura del tempo trascorso da quando si invia un segmento a quando si riceve il suo riscontro.
- **EstimatedRTT**: Stima del valore precedente calcolata iterativamente come:

$$EstimatedRTT_{i+1} = (1 - \alpha)EstimatedRTT_i + \alpha SampleRTT$$

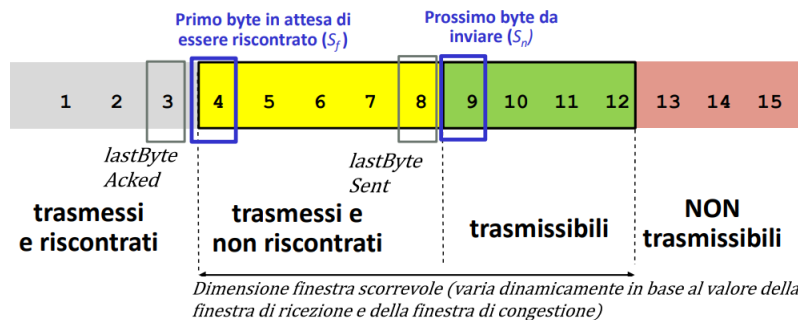
- **RTT<sub>DEV</sub>**: Stima della variabilità di RTT determinata con:

$$RTT_{DEV} = (1 - \beta)RTT_{DEV} + \beta |RTT_{SAMPLE} - RTT_{ESTIMATED}|$$

- **RTO**: Una volta ottenuti questi valori il timeout viene calcolato come:

$$RTO = RTT_{ESTIMATED} + 4RTT_{DEV}$$

**Finestra di Ricezione TCP** I dati inviati dall'applicativo sono mantenuti nel buffer di invio, e la trasmissione si basa su una sliding window che viene fatta scorrere a tempo di ricezione di un ACK.



### 3.5.5 Controllo di Flusso del TCP

Assumendo che sia mittente sia destinatario abbiano buffer, bisogna valutare cosa succeda nel caso in cui il **livello di rete** fornisca i dati a **velocità maggiore** rispetto a quella dell'**applicativo a consumarli** dal buffer.

L'obiettivo del **controllo di flusso** è quindi quello di mettere in **relazione** la **frequenza di invio del mittente** con la **frequenza di lettura** dell'applicazione **ricevente**.

Questo meccanismo è implementato tramite una variabile caratteristica **rwnd** mantenuta dal mittente calcolata come:

$$rwnd = RcvBuffer - (LastByteReceived - LastByteRead)$$

Il mittente fornirà questa variabile al destinatario nell'header del segmento e quest'ultimo verificherà se:

$$LastByteSent - LastByteAcked \leq rwnd$$

Esistono anche dei segmenti sonda di dimensione *1byte* che il mittente invia per ottenere la dimensione della **rwnd**.

### 3.5.6 Controllo di Congestione del TCP

Il fenomeno della congestione è causato da più sorgenti che tentano l'invio di dati tramite lo stesso percorso. Questo può provocare lunghi ritardi e perdita di pacchetti.

**Approcci per Controllo Congestione** Esistono due tipologie di controllo di congestione di rete:

1. **End To End**: I mezzi intermedi non forniscono alcun feedback, sono quindi gli host terminali tramite sintomi a rilevare la congestione della rete.
2. **Assistito dalla Rete**: I router segnalano la congestione<sup>3</sup>. Definito anche come **ECN** (Explicit Congestion Notification), dove si utilizzano **2bit** nell'**header IP** per indicare **congestione**. Nello specifico abbiamo un bit **ECE** settata dalla **destinazione** nel segmento di riscontro e un bit **CWR lato mittente** per indicare di aver ricevuto la notifica di congestione.

In TCP quindi il controllo di congestione è implementato imponendo a ciascun mittente di **limitare la frequenza di invio di pacchetti** sulla connessione in funzione della **congestione percepita**.

La dimensione della finestra di invio sarà quindi  $\min\{rwnd, cwnd\}$  con **rwnd** receiver window e **cwnd** congestion window. Quindi il rate di invio sarà:

$$rate_{invio} \leq \frac{\min\{rwnd, cwnd\}}{RTT}$$

---

<sup>3</sup>Questo infrange gli schemi impostati fino ad ora di totale isolamento tra i vari livelli del TCP/IP.



**Algoritmo Controllo Congestione di TCP** Si compone di tre fasi:

- Slow Start
- AIMD (Additive Increase Multiplicative Decrease)
- Reazione agli Eventi Perdita:
  - Fast Recovery
  - Reazione al Timeout

Prima della descrizione effettiva delle fasi è utile definire la **cwnd** (Congestion Window) misurata solitamente in **MSS** (Maximum Segment Size), ossia la massima quantità di dati trasportabile da un segmento.

**AIMD** Fase in cui TCP mittente aumenta proporzionalmente la propria finestra di congestione ad ogni ACK ricevuto:

- Ad ogni ACK la finestra di congestione viene incrementata, nello specifico aumento di 1 MSS per ogni RTT, quindi  $cwnd+ = \frac{1}{cwnd}$ .

**Slow Start** Fase in cui ad ogni ACK la cwnd viene incrementata di 1 MSS, quindi si raddoppia ad ogni RTT. Segue quindi un andamento esponenziale.

### 3.5.7 Tipologie di TCP che Istanzano Algoritmo di Controllo Congestione - (Reno, Tahoe, Cubic e Vegas)

**TCP Reno** Istanza delle fasi che abbiamo appena definito, aggiungendo anche gli eventi di perdita:

- **Gestione cwnd:** Viene definita una variabile soglia ad un valore alto.
  - Se  $cwnd < soglia$  allora si incrementa **cwnd** aumenta esponenzialmente in **Slow Start**.
  - Se  $cwnd > soglia$  allora si incrementa **cwnd** aumenta linearmente in **Additive Increase**.
- **Eventi di Perdita:** Esistono varie condizioni:
  - **Timeout:** Si setta  $soglia = cwnd$  e  $cwnd = 1$  MSS in **Slow Start**.
  - **3 ACK Duplicati:**  $soglia = 1/2cwnd$  e  $cwnd = soglia + 3MSS$  in **Fast Recovery**. In particolare in questa fase possono anche accadere altri eventi:
    - \* Se avviene un **timeout** si va in **Slow Start**.
    - \* Fino a che arrivano **ACK duplicati** allora  $cwnd = cwnd + 1MSS$ .
    - \* Se arriva un **ACK non duplicato** allora  $cwnd = soglia$  e si va in **Congestion Avoidance**.

A pagina successiva si mostra un esempio di controllo congestione in TCP Reno.

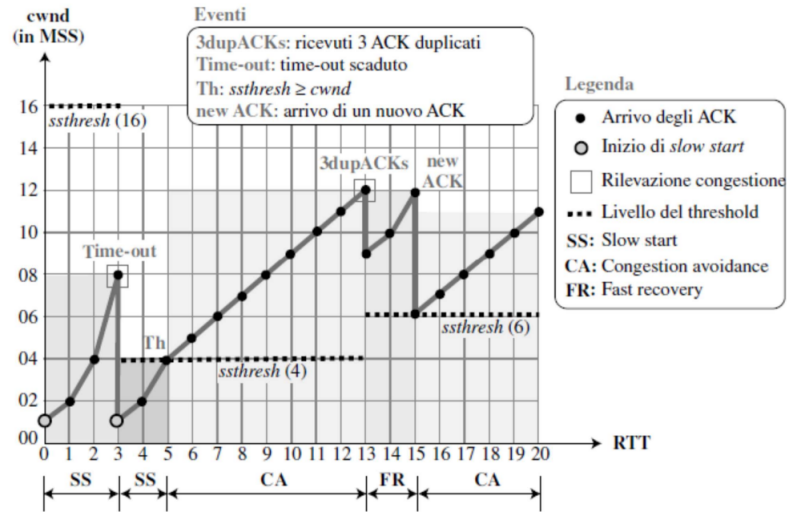


Figure 3: Gestione di congestione in TCP Reno

Si notano quindi i tre possibili stati ossia Slow Start SS, Congestion Avoidance CA e Fast Recovery FR.

**TCP Tahoe** Versione meno recente che comprende solo gli stati di Slow Start e Congestion Avoidance ed è così definito:

- **Timeout e 3 ACK Duplicati** sono gestiti allo stesso modo, ossia in **Slow Start** settando  $soglia = cwnd/2$  e  $cwnd = 1MSS$ .

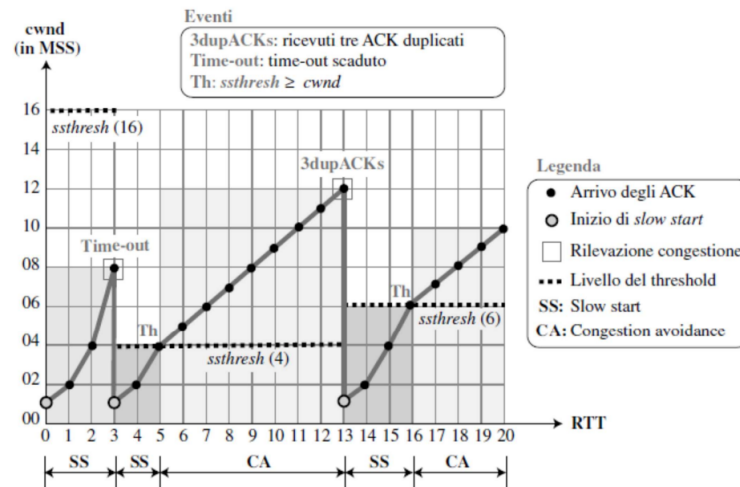
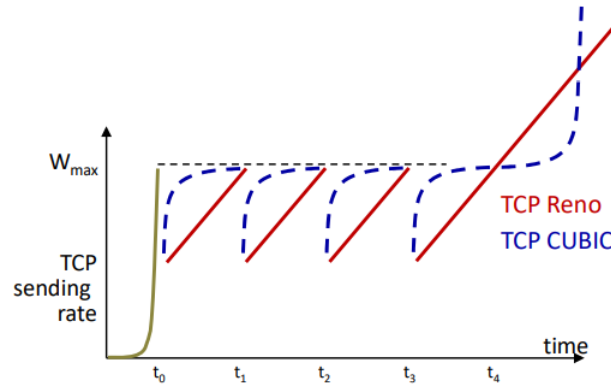


Figure 4: Gestione di congestione in TCP Tahoe

**TCP Cubic** Cerca di ottimizzare l'andamento "a dente di sega" del TCP Reno settando degli intervalli di ampiezza  $k$  e gli incrementi vengono così gestiti:

- Incrementi maggiori quando siamo lontani da  $k$ .
- Incrementi minori quando siamo vicini a  $k$ .



### 3.5.8 Throughput, Equità e Cenni di Attacchi TCP

Avendo mostrato i tre stati di TCP Reno, possiamo dire che macroscopicamente il TCP ha un andamento "a dente di sega". Da questo possiamo analizzarne il throughput, assumendo che il rate di trasmissione sia circa costante. Quindi:

- $W$ : Finestra di taglia massima.
- $w$ : Finestra di taglia minima, ossia  $w = \frac{W}{2}$ .
- **Throughput**: Se la finestra è  $W$  allora è  $\frac{W}{RTT}$ , invece dopo un evento di perdita quando la finestra è  $w$  il throughput diventa  $\frac{W}{2RTT}$ .

Calcolando quindi una media tra le due possibili casistiche otteniamo:

$$throughput = \frac{1}{2} \left( \frac{W}{RTT} + \frac{w}{RTT} \right) = \frac{3}{4} \frac{W}{RTT}$$