

# SISTEMI DI ELABORAZIONE E TRASMISSIONE DELL'INFORMAZIONE

Appunti di [Malchiodi Riccardo](#)

## RFC

Le pubblicazioni sugli standard di Internet vengono dette **Request For Comment** (RFC). Gli RFC sono il modo in cui vengono definiti i protocolli internet. Sono documenti di tipo testuale che raccolgono tutte le specifiche di un certo protocollo.

## Struttura rete Internet

La rete Internet è un sistema molto complesso.

Bisogna quindi usare tutte le tecniche che conosciamo per poter arrivare a semplificare il discorso e focalizzarsi su alcuni aspetti della rete.

Il modo classico per poter gestire la complessità della rete internet è quello di suddividere la rete in diversi livelli.

La rete si divide in 5 livelli:

Livello	Nome	Protocollo
5	<b>Applicativo</b>	HTTP (esempio)
4	<b>Trasporto</b>	TCP / UDP
3	<b>Rete</b>	IPv4 / IPv6
2	<b>Datalink</b>	Ethernet
1	<b>Fisico</b>	-

## Invio dei messaggi Internet

L'invio dei messaggi attraverso la rete avviene tramite due dispositivi:

- Host(sender)
- Host(receiver)

Il passaggio dei messaggi tramite internet avviene secondo la tecnica del **store & forward**.

In un sistema store & forward, ogni pacchetto di dati (ovvero un piccolo blocco di informazioni) viene memorizzato temporaneamente in un nodo intermedio, come un router o un gateway, prima di essere inoltrato al nodo successivo sulla rete. Questo processo avviene ripetutamente fino a quando il pacchetto non raggiunge la sua destinazione finale.

Per giungere al destinatario può essere necessario passare per Host intermedi.

L'instradamento dei messaggi avviene tramite salti successivi **multi hop**.

## Spiegazione di ciascun livello

### Livello fisico(1)

Il livello fisico si occupa della trasmissione dei dati in forma di segnali (elettrici, ottici o radio) attraverso il mezzo fisico, come cavi di rame, fibre ottiche o onde radio. Le sue funzioni principali sono:

- Definire il mezzo fisico attraverso cui i dati viaggiano (ad esempio, cavi Ethernet, fibra ottica, Wi-Fi).
- Gestire la trasmissione e la ricezione di bit (0 e 1) sotto forma di segnali.
- Sincronizzazione dei segnali tra i dispositivi.
- Regolare velocità di trasmissione e modulazione dei segnali.

Non si preoccupa del contenuto dei dati, ma solo della modalità con cui i dati vengono trasferiti fisicamente tra i dispositivi.

### Livello DataLink(2)

Il **livello di collegamento dati** si occupa della **trasmissione diretta di pacchetti** tra due nodi fisicamente connessi, fornendo un canale di comunicazione affidabile su cui il livello di rete può basarsi. Mentre il livello di rete si occupa di instradare i pacchetti su più reti, il livello di collegamento dati opera a livello locale tra i dispositivi che condividono lo stesso mezzo fisico, come due router adiacenti o un computer e uno switch. Quando un datagramma del livello di rete deve essere trasferito tra due nodi adiacenti (come due router lungo il percorso verso la destinazione finale), viene incapsulato in un **frame** dal livello di collegamento dati. Il frame include non solo il datagramma del livello di rete, ma anche informazioni di controllo specifiche del collegamento, come l'indirizzo **MAC** (*Media Access Control*) dei nodi di partenza e di destinazione.

### Livello Di Rete(3)

La funzionalità principale del livello di rete è di fornire l'instradamento.

Questo protocollo definisce la comunicazione tra 2 host attraverso il loro indirizzo IP.

Abbiamo due versioni di indirizzo IP:

- **IPv4**: usa 32 bit per l'indirizzamento

- **IPv6:** usa 128 bit per l'indirizzamento

Il protocollo IP fornisce comunicazione logica tra host. il suo modello di servizio viene chiamato **best-effort**, questo significa che IP “fa del suo meglio” per consegnare i datagram tra host comunicanti, ma non offre garanzie.

Per questo motivo viene definito protocollo non affidabile.

Gli indirizzi **IPv4** sono suddivisi in 2 parti:

- Sottorete (subnet)
- Numero dell'host

L'idea è che la nostra rete Internet è composta da un insieme molto grande di sottoreti connesse tra di loro attraverso un numero molto elevato di router.

La prima parte dell'indirizzo serve per individuare un'intera sottorete. Mentre la seconda parte dell'indirizzo ci aiuta ad individuare una delle macchine chiamate **host** contenute all'interno di quella sottorete.

L'instradamento consiste nel preoccuparsi soltanto della parte **subnet** dell'indirizzo per recapitare il messaggio sulla sottorete di appartenenza.

## Livello di Trasporto(4)

il livello di trasporto di internet trasferisce i messaggi del livello di applicazione tra punti periferici gestiti dalle applicazioni.

Abbiamo due protocolli di trasporto (che andremo ad approfondire successivamente): **TCP** e **UDP**.

*TCP* fornisce un servizio **orientato alla connessione** mentre quello *UDP* fornisce un servizio **non** orientato alla connessione

## Livello di applicazione(5)

Il livello di applicazione è la sede delle applicazioni di rete e dei relativi protocolli e ha il compito di fornire **interfacce** e strumenti che permettono agli utenti di comunicare attraverso una rete. Si occupa di gestire la comunicazione tra applicazioni software, client di posta elettronica o servizi di streaming.

## Datagrammi

Un datagramma è costituito da due parti principali:

- **Header:** parte iniziale del datagram e contiene informazioni di controllo per la corretta consegna dei dati, come l'indirizzo di origine, di destinazione, ecc.
- **PayLoad:** è la parte che contiene i dati effettivi da inviare (sarebbe il contenuto del messaggio).

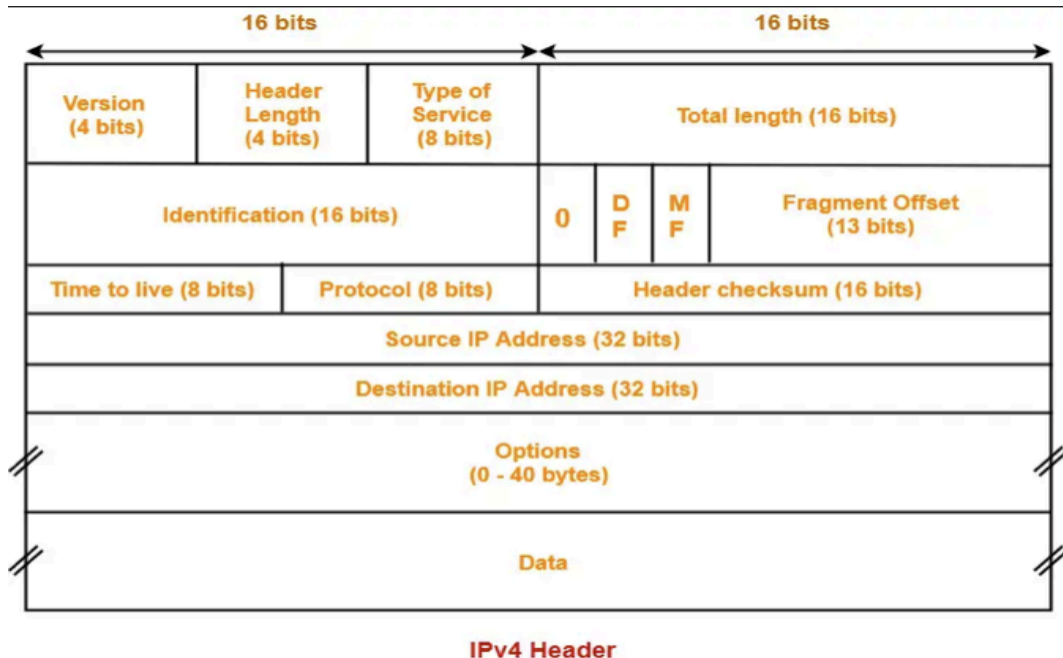
Il **datagram** cerca di usare il tragitto più corto per arrivare a destinazione .

Un datagram è qualcosa di molto contenuto come può essere, ad esempio, una lettera.

Come una lettera il datagramma è caratterizzato da un'intestazione che contiene gli indirizzi del mittente e del destinatario.

Non abbiamo però nessuna garanzia che il messaggio arrivi a destinazione.

Nel caso di **IPv4** l'header è suddiviso in questo modo:



**i primi 4 bit** sono interpretati come un intero tra 0 e 15 e si chiamano numero di versione (0100).

Nel caso IPv6 questi 4 bit assumono valore 0110.

La lettura dei **secondi 4 bit** mi permettono di capire la lunghezza dell'intestazione.

Gli altri 8 bit si chiamano **tipo di servizio**.

I 16 bit mi dicono **la lunghezza** (in byte) del datagramma.

I successivi 32 bit sono suddivisi in 3 parti:

1. **Identificatore**: rappresentata su 16 bit
2. Serve per gestire la frammentazione del datagramma: 16 bit suddivisi in 3 bit chiamati **Flag**.
3. suddivisa anche lei in 3 parti:

- 1 Byte che prende il nome di **Time To Live (TTL)**.
- 2 Byte chiamato **next level protocol**
- I 16 bit rimasti sono chiamati **checksum**

il **TTL** è un numero che viene inizializzato a un certo valore deciso dal mittente del datagramma.

L'utente può scegliere a suo piacere il suo valore iniziale purché sia compreso tra l'intervallo 0-255.

Questo numero serve per gestire il numero di **Hop** che possono essere effettuati dal datagramma. Un router quando si vede arrivare un datagramma va a vedere il valore contenuto nel campo **TTL**, lo decrementa di 1 e poi se ottiene come risultato 0 cancella il datagramma. Se invece il risultato di questo decremento è maggiore di 0 lo inoltra all'Hop successivo.

Se un datagramma non riesce a raggiungere la destinazione entro il numero di hop massimo, il pacchetto viene scartato e viene inviato un messaggio di errore al mittente, utilizzando il protocollo **ICMP**.

Uno dei motivi per cui un datagramma non arriva a destinazione è quello di settare il **TTL** con un valore troppo basso.

Il **next level protocol** indica se stiamo utilizzando un protocollo TCP o UDP.

Il **checksum** è una tecnica di verifica degli errori utilizzata in vari protocolli di rete per garantire l'integrità dei dati trasmessi.

- **Vantaggio:** il checksum è facile da calcolare
- **Svantaggio:** non tutti gli errori possono essere riconosciuti.

L'header si conclude con un indirizzo mittente e un indirizzo ricevente.

## Modalità Datagram (UDP)

La modalità datagram si basa su un host mittente, un host destinatario e la rete internet, che attraverso dispositivi chiamati **router** definisce il percorso che i messaggi devono percorrere per arrivare a destinazione.

I messaggi in questione vengono chiamati **datagrammi**.

Il protocollo UDP è descritto in *RFC-768*.

A livello di rete la principale funzionalità è l'instradamento, mentre a livello di trasporto la principale funzionalità è il **multiplexing**.

Il compito di trasportare i dati dei segmenti a livello di trasporto verso la giusta socket (il socket verrà spiegato più avanti) viene detto **demultiplexing**.

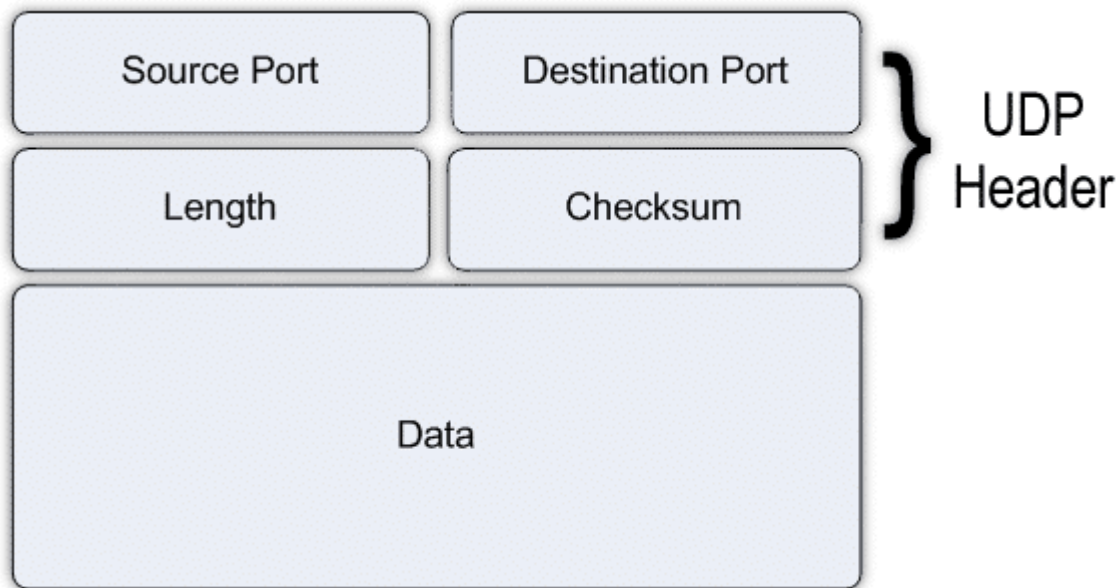
Il compito di radunare frammenti di dati da diverse socket sull'host di origine e incapsulare ognuno con intestazioni a livello di trasporto per creare dei segmenti e passarli al livello di rete è detto **multiplexing**.

In sintesi:

- **multiplexing:** è una tecnica che consente di **combinare più segnali o flussi di dati** e trasmetterli simultaneamente su un unico canale di comunicazione o mezzo fisico. Lo scopo del multiplexing è ottimizzare l'utilizzo delle risorse di rete.
- **demultiplexing:** è il processo complementare al **multiplexing**. Consiste nel **separare** i diversi segnali o flussi di dati che sono stati combinati e trasmessi attraverso un singolo canale di comunicazione. L'obiettivo del demultiplexing è recuperare i singoli flussi di dati originali a partire da un segnale che ha subito multiplexing (viene chiamato *multiplexato*).

Ora parliamo del **protocollo UDP**.

Abbiamo un'intestazione molto semplice:



I primi 16 bit sono il numero di porta sorgente, i successivi 16 bit sono il numero di porta destinazione. Abbiamo poi altri 32 bit che vengono interpretati come la lunghezza del messaggio e i successivi 16 bit che sono il checksum.

Dopodichè abbiamo il Payload che contiene il messaggio.

Per fare un controllo di integrità utilizziamo il checksum a 16 bit.

Il controllo di integrità nel caso **UDP** si riferisce all'intero messaggio.

## Come avviene la comunicazione UDP

Ecco come avviene la comunicazione tra due host (Host A e Host B) utilizzando UDP:

### 1. Preparazione dei dati:

Host A decide di inviare dei dati a Host B. Prima di inviare, prepara i dati che desidera inviare, ad esempio un messaggio, un pacchetto audio o video.

### 2. Creazione del datagramma:

Host A crea un datagramma UDP. Questo datagramma contiene:

- **Intestazione UDP:** include informazioni necessarie, come:
- **Porta sorgente:** la porta utilizzata da Host A per inviare il datagramma.
- **Porta di destinazione:** la porta su cui Host B è in ascolto.
- **Lunghezza:** la lunghezza totale del datagramma.
- **Checksum:** un valore per la verifica dell'integrità dei dati.

- **Carico utile:** i dati che Host A vuole inviare a Host B.

### 3. Invio del datagramma:

Host A invia il datagramma al livello di rete (esegue una *send()* ).

Durante la send A deve specificare l'indirizzo dell'area di memoria che contiene il datagramma che deve essere inviato.

### 4. Trasmissione attraverso la rete:

Il pacchetto IP contenente il datagramma UDP viene inviato attraverso la rete. Può passare attraverso vari router e nodi lungo il percorso fino a raggiungere Host B.

### 5. Ricezione del datagramma:

Quando il pacchetto arriva a Host B, il livello di rete riceve il pacchetto IP e lo passa al livello UDP.

Host B verifica il checksum presente nell'intestazione UDP per controllare se ci sono stati errori durante la trasmissione. Se il checksum è valido, significa che i dati sono integri.

### 6. Elaborazione dei dati:

Host B utilizza le informazioni nell'intestazione UDP per determinare quale applicazione (identificata dalla porta di destinazione) deve ricevere i dati.

Il datagramma UDP viene quindi passato all'applicazione corrispondente su Host B, che può elaborare i dati ricevuti.

### 7. Parte finale:

A differenza dei protocolli orientati alla connessione come TCP, non viene inviata alcuna conferma a Host A riguardo alla ricezione del datagramma. Se Host B non riceve il datagramma, non lo comunica a Host A.

Una caratteristica importante della comunicazione datagram (come detto prima) è la non affidabilità. Infatti non abbiamo nessuna garanzia che il destinatario riceva tutti i messaggi. Proprio per questo il protocollo UDP non è affidabile.

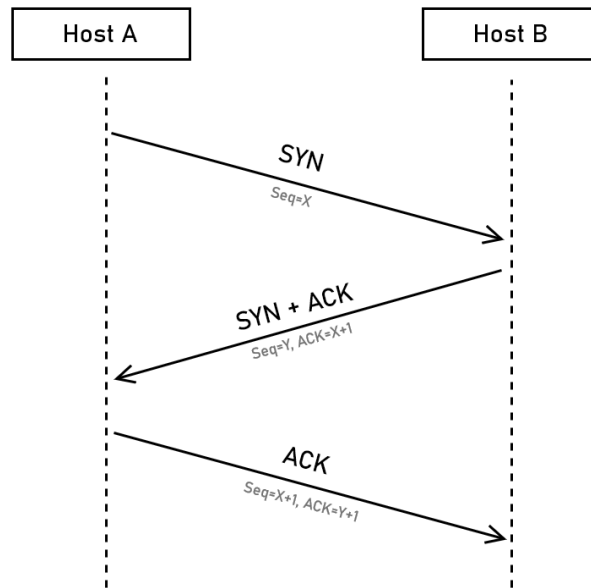
## Modalità Stream (TCP)

TCP fornisce un servizio orientato alla connessione.

Stream definisce un canale virtuale tra mittente e destinatario. Il protocollo di connessione viene chiamato **3- way handshake**.

A grandi linee funziona in questo modo:

1. L'iniziativa di aprire la connessione viene presa dal client e manda un primo messaggio di richiesta al server.
2. Il server riceve il messaggio e risponde al client
3. infine se il client riceve la risposta da parte del server invia a sua volta un terzo messaggio per confermare l'apertura della connessione.



*schema di questi 3 passaggi*

Ora vediamo la struttura del TCP:

Abbiamo una parte di intestazione formata da:

- Numero di porta sorgente (16 bit)
- Numero di destinazione (16 bit)
- Numero di sequenza (32 bit)
- Numero di acknowledgment (32 bit)
- Finestra di ricezione / receive windows (16 bit)
- Lunghezza dell'intestazione (4 bit)

Numero di Porta Sorgente	Numero di Porta Destinazione				
Numero di Sequenza					
Numero di Acknowledge					
S Y N	A C K	U R G	P S H	R S T	F I N
Lunghezza Header		Receive Window			

Sono presenti inoltre 6 bit dedicati ai **flag**.

Il bit **ACK** viene usato per indicare che il valore trasportato nel campo di acknowledgment è valido.



I bit **RST**, **SYN** e **FIN** sono usati per impostare e chiudere la connessione.

Infine abbiamo PSH e URG (non ne abbiamo parlato a lezione).

## Spiegazione completa 3-way Handshake

Supponiamo di avere due Host **H1** e **H2**, il primo rappresenta il Client mentre il secondo il Server.

Ecco come avviene:

1. Il Client per **aprire la connessione** deve compilare l'header da inviare al Server. Attiva il flag **SYN** per richiedere l'apertura del messaggio. Tutti i suoi altri flag assumono valore 0 per il momento. Nei primi 16 bit scrive il numero di porta sorgente (Es. 50127). Viene riempito anche il numero di porta destinazione (ovviamente il valore sarà quello del Server).  
Il numero di sequenza (per motivi di sicurezza) viene generato casualmente (Es. 3145).  
**ACK** per ora è vuoto

2. H2 deve decidere se aprire la connessione .  
Nel caso in cui il Server decidesse di NON aprire la connessione viene attivato il flag **RST** e la comunicazione si interrompe.  
Nel caso in cui, invece, sia disposto di aprire la connessione risponde al Client attivando:
  - il flag **SYN** per comunicare la sua volontà di aprire la connessione.
  - il flag **ACK** per comunicare che il messaggio è stato ricevuto con successo.

Il server inserisce nel numero di sequenza un numero casuale(ES. 1234567), nel numero acknowledge inserisce il **SEQ#** del client + 1 (31416)

3. Il Client riceve la risposta del Server e invia un terzo messaggio per confermare che la connessione è stata aperta con successo.  
Per fare ciò attiva il flag **ACK**, il flag **SYN** viene spento ( = 0 ) in quanto ormai non serve più.  
Il Client incrementa il suo stesso **SEQ#** di 1 ottenendo , nel nostro esempio, 31416.  
Mentre per quanto riguarda il numero **ACK** viene inserito il **SEQ#** del Server

incrementato di 1 (1234568).

Adesso possiamo inserire nel **payload** il messaggio che vogliamo inviare.

## TIMEOUT

Nel contesto del **protocollo TCP**, un **timeout** è un intervallo di tempo entro il quale il mittente si aspetta di ricevere un **acknowledgment (ACK)** per i dati inviati. Se non riceve una conferma (ACK) entro questo tempo, il mittente assume che il pacchetto sia andato perso o che ci sia stato un problema nella trasmissione, quindi lo ritrasmette.

La soglia di timeout dipende dalla distanza dei due host e dalle caratteristiche fisiche di essi; queste variabili vengono considerate tramite il calcolo di una metrica detta **Round-Trip-Time (RTT)**.

In parole più semplici RTT sarebbe il tempo che impiega un pacchetto di dati per viaggiare dal mittente al destinatario e tornare indietro sotto forma di risposta o conferma

## DNS

Esistono due modi per identificare gli host: Il nome e l'indirizzo IP. Le persone preferiscono il primo mentre i router il secondo.

Al fine di conciliare i due approcci è necessario un **servizio in grado di tradurre gli hostname nei loro indirizzi IP**. Questo servizio è il **DNS** (domain name system).

Un DNS è come un elenco telefonico di internet. Quando vuoi visitare un sito web, digiti un nome, come "google.com". Il DNS traduce quel nome in un indirizzo IP, che è una serie di numeri che identifica il server dove si trova il sito. In questo modo, il computer può trovare e connettersi al sito giusto.

Il protocollo DNS utilizza **UDP** e la **porta 53**.

## Struttura DNS

i DNS si distribuiscono ad albero:

- **Root Server:** forniscono gli indirizzi IP dei TLD server.
- **Top-level domain(TLD) server:** forniscono gli indirizzi IP dei server autoritativi.  
I TLD sarebbero i nomi che si trovano alla fine degli indirizzi web. Ad esempio .com, .it, .uk ecc

- **DNS autoritativi:** sono i DNS che contengono i dati specifici del nome del dominio. rispondono alle richieste per quel dominio e ne forniscono i dati relativi.

Se voglio raggiungere una determinata informazione devo poter scorrere questo albero, per farlo esistono due algoritmi di ricerca.

## Algoritmo Ricorsivo

L'algoritmo ricorsivo funziona come un gioco di telefono: quando un Client richiede un nome di dominio a un server DNS, se il server non ha già l'informazione, gira la richiesta a un altro server. Inizialmente, il server contatta un **server di root**, che lo guida a un server di dominio di primo livello (**TLD**), come .com o .org. Poi, il server TLD indirizza la richiesta al server specifico che ospita il dominio. Questo processo continua fino a ottenere la risposta finale, con ogni server che si occupa di una parte della ricerca. È un metodo efficiente, poiché il server che riceve la richiesta non deve fare tutto il lavoro da solo; si affida ad altri server per ottenere l'informazione necessaria.

## Algoritmo iterativo

L'algoritmo iterativo adotta un approccio diverso, in cui il server **DNS** cerca di risolvere il nome di dominio direttamente. Quando riceve una richiesta, se non ha la risposta a portata di mano, inizia a contattare i server in sequenza. Prima si rivolge a un server di root, il quale fornisce l'indirizzo del server **TLD** da contattare. Successivamente, contatta il server **TLD**, il quale lo guida verso il server specifico che contiene l'indirizzo **IP** richiesto. In questo caso, il server **DNS** è autonomo e gestisce ogni passaggio della ricerca senza passare la responsabilità ad altri.

**Metodo Migliore:** Il Client preferirà il metodo ricorsivo, mentre per il Server il metodo migliore è quello iterativo per gestire meglio le richieste da parte dei vari client.

**L'approccio che si usa** è quello di usare un server locale che si interfaccia con il client che può essere messo in **modalità ricorsiva**, rispondendo così al client dopo aver raccolto tutte le informazioni. interagisce con la gerarchia dei server, che potranno rispondergli in maniera iterativa (che verrà salvata in maniera ricorsiva per la risposta al client). Questo server potrebbe essere limitato (nel senso che non può essere contattato da tutti). I client potrebbero richiedere la stessa domanda più volte, quindi il server locale avrà un meccanismo di **memoria organizzato con una cache**, come effetto collaterale abbiamo anche la riduzione del traffico di rete.

## Header DNS

**Intestazione del DNS** : prevede 6 numeri divisi in 16 bit: il numero identificativo, il flag, il numero di domande, il numero di risposte, il numero di risposte autoritative e il numero di risposte aggiuntive. C'è la sezione domande, la sezione risposte, la sezione risposte autoritative e la sezione risposte aggiuntive.

- **Risposta normale:** Questa risposta proviene dalla **cache** del server DNS. Il server restituisce informazioni già memorizzate, quindi non deriva direttamente da un'interazione con un server autoritativo.
- **Risposta autoritativa:** risposta calcolata in questo momento sulla base di un'interazione con un server autoritativo (sono sicuro che è valida in quanto appena calcolata)
- **Risposta aggiuntiva:** Questo tipo di risposta fornisce informazioni extra non richieste dal Client.

**Possono sorgere problemi in diverse situazioni:** un attacco di **cache-poisoning** sul contenuto della cache nel server locale; un meccanismo di phishing ecc...

Problematiche abbastanza gravi, Possono essere mitigate con delle contromisure: l'id è un numero casuale da 16 bit, che viene memorizzato fino a quando non ottengo risposta, per sapere da chi devo ancora riceverla.

## Protocollo NTP

Il protocollo NTP è un protocollo fondamentale per la sincronizzazione degli orologi di una rete. (RFC di riferimento: 5905)

Il nostro computer, anche se non connesso alla rete, ha la capacità di misurare il tempo grazie a un orologio interno (**clock**). All'interno del sistema, c'è un processore dotato di un clock, che tipicamente opera a una frequenza elevata. I processori sono progettati per ottimizzare il consumo energetico e ridurre la dissipazione di calore, e per farlo utilizzano **frequenze di clock** variabili. Tuttavia, è importante notare che il clock della CPU non è quello che usiamo per misurare il tempo reale. Il clock utilizzato per misurare il tempo è separato dal processore; è integrato nella scheda madre e funziona come un orologio, di solito basato su **tecnologia al quarzo**. La velocità di accesso a questo orologio potrebbe non essere sufficiente per fornire un'indicazione precisa dell'ora esatta. Questo porta a un difetto nel clock hardware del sistema, che può causare lentezza e possibili errori. Per ovviare a questo problema, **il computer legge l'orologio fisico solo una volta durante il boot**. Successivamente, l'ora viene salvata nella **RAM** e il sistema operativo utilizza un clock virtuale. Inoltre, il sistema impiega un "**alarm clock**", che è utilizzato per aggiornare l'orologio virtuale. Questo alarm clock genera degli interrupt a cadenze regolari, permettendo così di incrementare il clock virtuale di uno ogni volta che è necessario.

## Connettere la macchina alla rete

Quando colleghiamo il computer alla rete, la macchina client si connette a un server **NTP** (Network Time Protocol). Invia una richiesta al server, che risponde fornendo l'ora esatta. Successivamente, il client confronta l'ora ricevuta dal server con il suo orologio interno. Se l'orario dell'orologio interno risulta errato, il client lo corregge di conseguenza. Questo processo avviene attraverso una comunicazione di tipo client-server, utilizzando il **protocollo UDP**.

Come fa il server ad avere un orario più preciso del nostro?

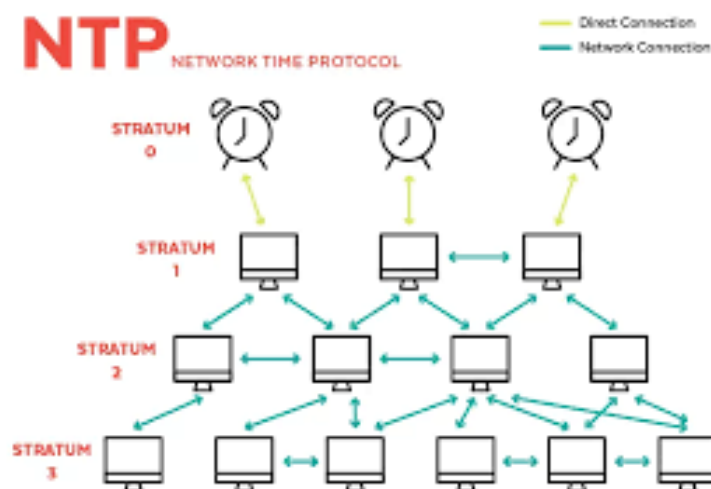
**La precisione negli orologi meccanici** dipende dalla temperatura, dalle condizioni ambientali ecc. che cambia la geometria e quindi l'oscillazione del bilanciere; anche negli orologi al quarzo la frequenza di oscillazione può variare in base alla temperatura.

**Posso studiare il fenomeno e correggere la frequenza in base alla temperatura.**

L'orologio migliore è l'**orologio atomico**, basato sul decadimento radioattivo, fenomeno fisico che non dipende dalle condizioni ambientali

## Struttura NTP

**NTP ha una struttura gerarchica**, la cui idea è quella di poter identificare diversi tipi di riferimento del tempo, chiamati strati, numerati da 0 a 15 (da massimo livello di precisione a minimo livello di precisione).



**strato 0:** orologio atomico

**strato 1:** computer collegato direttamente con un orologio atomico (con rete locale); può funzionare come server

**strato 2:** computer distante collegato a internet, può funzionare come server

**strato 3:** computer collegato ad un computer di strato 2

**strato n:** computer collegato ad un computer di strato n-1

**Un client può collegarsi a qualsiasi strato a partire dallo strato 1**, ma è preferibile scegliere quelli a livello più basso, poiché offrono maggiore precisione. Inoltre, i computer in questa rete

possono funzionare sia come client che come server, creando così un'architettura **peer-to-peer (P2P)**.

**La rappresentazione del tempo** nel protocollo NTP è simile a quella utilizzata nel sistema **Unix** e si basa su due parole da 32 bit: una rappresenta il numero di secondi e l'altra il numero di nanosecondi. Il valore temporale è rappresentato come un numero razionale, con la virgola posizionata tra le due parole.

L'epoca definisce il valore zero, ossia il punto di partenza di questo contatore. Nel caso del protocollo NTP, la rappresentazione è quella del **UTC (Universal Time Coordinated)**, che inizia dall'epoca del **01/01/1900**. Il massimo numero rappresentabile prima dell'overflow si estende fino al **2038**.

Nel sistema Unix, che adotta una rappresentazione simile ma leggermente diversa, ci sono alcune differenze significative:

- Il numero ha un segno che consente di tornare indietro rispetto all'epoca.
- L'epoca in Unix è fissata al **01/01/1970**, data comunemente considerata la nascita del sistema Unix.
- Anche nel sistema Unix si presenta un problema di overflow, e il limite di rappresentazione si estende fino al **2032/2036** (la data precisa non è ben definita).

Per affrontare il problema dell'overflow, si prevede di aumentare il numero di bit a disposizione, passando a una rappresentazione su **64 + 64 bit**.

Quando un client richiede l'ora a un server utilizzando il protocollo UDP, non è garantito che riceverà una risposta. Durante questo processo, viene utilizzata una **marca temporale (timestamp)** per tenere traccia degli orari. Il Client prepara un messaggio che include il **timestamp (t0)** segnato dal suo orologio. Quando il server riceve la richiesta, inserisce immediatamente il suo **timestamp (t1)**. Successivamente, il server applica un secondo **timestamp (t2)** prima di inviare la risposta al client. Quando il messaggio arriva di nuovo al client, quest'ultimo aggiunge un ulteriore **timestamp (t3)**.

Ecco un riepilogo dei timestamp coinvolti:

- **t0**: è l'indicazione dell'orario dal client al momento della partenza del messaggio, che potrebbe essere errata.
- **t1**: rappresenta il vero orario di arrivo della richiesta presso il server.
- **t2**: indica il vero orario di partenza della risposta dal server.
- **t3**: è il falso orario di ricezione della risposta da parte del client, poiché l'orologio del client potrebbe essere impreciso.

Il **RTT (Round Trip Time)** è il tempo totale impiegato per il viaggio di andata e ritorno del messaggio, che può essere calcolato utilizzando i timestamp.

**Per ricevere un'indicazione affidabile** dobbiamo richiedere per un certo numero di volte fino a raccogliere una statistica significativa. **In una situazione realistica dell'implementazione del NTP:** parto con un'indicazione di una certa quantità di server NTP che possono essere contattati; ho un **elenco di server** che posso contattare, devo fare più richieste ai vari server (che possono essere di qualsiasi livello). **Il tutto viene organizzato in cicli:** nel primo ciclo lancio la richiesta, aspetto per un tempo predefinito e raccolgo le risposte fino a quel tempo; passo al ciclo successivo. Possiamo partire con una certa frequenza (esempio una richiesta al minuto), continuo per tot minuti: dopo tot cicli ho raccolto una statistica, sulla quale mi posso basare per togliere dalla lista dei server che non rispondono nel tempo ecc. Controllo il contenuto delle risposte, calcolo il round trip time, e cercherò di tenere chi ha un Rtt costante ed eliminare quelli che ne hanno uno variabile. Calcolo poi l'errore dei server, e in base a quello controllo l'affidabilità dei vari server (esempio: se 5 server dicono che l'orologio è indietro di 1 minuto, e un server dice che l'orologio è avanti di un minuto, quest'ultimo sarà sbagliato). Continuo fino a quando non trovo i server più affidabili e a quel punto aggiusto il mio tempo in base ai più affidabili: a questo punto sono sincronizzata con questo server, e quindi non devo più "testarlo" ad ogni ciclo.

## Socket

Un socket è un interfaccia di programmazione usata dal **Posix** (insieme di specifiche che facilita la scrittura di software) per accedere alle funzionalità di rete. Un socket è una sorta di porta di comunicazione che permette il passaggio di informazioni da un'applicazione alla rete e viceversa. Esso deve essere messo in comunicazione con dispositivi fisici della macchina, in particolare dalla **NIC**.

Abbiamo due tipi di socket:

- **Stream socket:** orientati alla connessione basati su protocolli affidabili come **TCP**.
- **Datagram socket:** non orientati alla connessione basati su protocolli **UDP**.

## Organizzazione dell'Interfaccia di Programmazione

L'interfaccia di programmazione viene organizzata attraverso varie **chiamate di sistema** che svolgono una parte delle operazioni legate alla ricezione, alla trasmissione e, in caso di un **socket TCP**, anche all'apertura della connessione.

La system-call **socket()** sul client lo inizializza, aprendo il file di comunicazione nell'applicazione; la chiamata restituisce un intero che corrisponde al file descriptor associato ad esso.

Un **file descriptor** è un'interfaccia di basso livello che rappresenta un riferimento a una risorsa gestita dal sistema operativo.

il file descriptor ottenuto da **socket()** è il punto di riferimento per tutte le operazioni di comunicazione che il client deve eseguire tramite il socket, permettendo di trattare il canale di comunicazione come una risorsa gestita dal sistema operativo.

# Stream Socket

Supponiamo di avere due Host, Client e Server.

Per poter stabilire una connessione entrambi devono svolgere una serie di compiti specifici:

## Client

Il Client crea un socket con la system call **socket()**, successivamente esegue la funzione **connect()** che permette di mandare un messaggio **SYN** verso il Server, in questo modo richiede l'apertura della connessione.

Si spediscono e ricevono dati. Nel caso di socket orientati alla connessione generalmente si usano le system call **send()** e **recv()**. Si possono anche usare le system call **write()** e **read()**. Si chiude il socket con le system call **shutdown()** (solo per TCP) e **close()**.

## Server

Lato "server", i passi per stabilire una connessione possono essere schematizzati ad alto livello come segue:

- Si crea un socket con la system call **socket()**.
- Si associa il socket a un indirizzo e a una porta predefinita usando la system call **bind()**.
- Si notifica al sistema operativo la disponibilità a ricevere richieste di connessione con la system call **listen()**.
- Si accetta la prossima richiesta di connessione proveniente da un client con la system call **accept()** (la quale può restituire indirizzo e porta del client che si vuole connettere). La system call **accept()** in realtà crea automaticamente un nuovo socket sul server, connesso col client, mentre il socket originario, usato per la negoziazione della connessione, rimane nello stato listening, e può essere usato per accettare altre richieste di connessione (tipicamente da altri client) sulla stessa porta del server. Se e quando il server decide di non accettare altre connessioni su quella porta, può chiudere il socket principale messo in listening mediante la system call **close()**.
- Si spediscono e ricevono dati usando il socket ausiliario creato automaticamente dalla **accept()**.
- Si chiude il socket ausiliario di comunicazione mediante le system call **shutdown()** e **close()**.

# Datagram socket

Quando si utilizzano i **datagram socket** la gestione della connessione è diversa rispetto ai **socket stream**, poiché **UDP è un protocollo senza connessione**. Ciò significa che non c'è un vero e proprio processo di "connessione" stabile come con TCP.



I pacchetti (o datagrammi) vengono semplicemente inviati dal client al server e viceversa, senza stabilire una connessione e senza garantire che i pacchetti arrivino o che arrivino in ordine.

## Client

I passaggi che deve svolgere il Client sono:

- Creare un socket con **socket()**
- Inviare datagrammi al server con **send\_to()**
- Ricevere risposte dal server con **recvfrom()**.
- Chiudere il socket con **close()**.

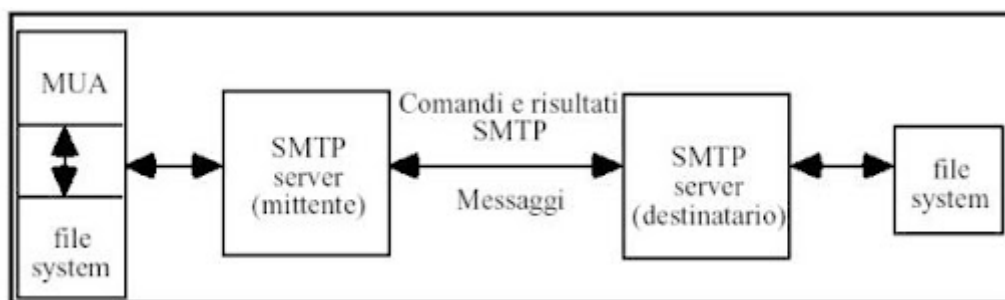
## Server

I passaggi che deve svolgere il Server sono:

- Creare un socket con **socket()**
- Associare il socket a un indirizzo IP e una porta con **bind()**.
- Ricevere datagrammi dai client con **recvfrom()**.
- Inviare risposte ai client con **sendto()**.
- Chiudere il socket con **close()**.

## Protocollo SMTP

Serve per gestire la posta elettronica, la struttura è la seguente:



Abbiamo un client che si connette a un server, **la connessione è di tipo TCP**, si lavora sulla porta 25, **il client usa SMTP per inviare i messaggi di posta elettronica**.

Quindi viene aperta la connessione, vengono mandati una serie di comandi e infine il server accetta di prendersi in carica di ricevere messaggi.

Quando il messaggio arriva a destinazione viene inserito nel **Mbox**(mail box) del destinatario.

Il compito del **SMTP** è quindi terminato. Dopodiché il destinatario dovrà prendersi cura di accedere al **Mbox** per visualizzare i messaggi che ha ricevuto.

Nella versione più semplice **SMTP** può funzionare da solo, permettendo al destinatario di accedere direttamente ai file della **Mbox** per vedere il contenuto.

# Evoluzione delle Necessità e Limiti di SMTP

Con il tempo, le esigenze si sono evolute, evidenziando alcune problematiche:

- **Connettività continua:** Una delle prime questioni riguarda la possibilità di avere un dispositivo sempre connesso alla rete. Infatti, per ricevere i messaggi in arrivo, la macchina che ospita la **Mbox** dovrebbe essere sempre connessa.
- **Messaggi in transito:** Quando la destinazione non è disponibile, i messaggi restano temporaneamente in transito. Dopo un certo periodo, il server **SMTP** che gestisce la consegna abbandona il tentativo di recapito e restituisce un errore.

**Soluzione intermedia:** Si può utilizzare un **server di posta** sempre connesso, diverso dalla macchina del destinatario finale. Questo server funge da punto di raccolta dei messaggi e, successivamente, il destinatario può accedervi tramite altri protocolli. Per gestire questi problemi sono stati aggiunti altri protocolli. Il primo è stato il protocollo **POP**. Sono state definite diverse versioni, al momento è utilizzata la **versione 3 (POP3)**, definita nell'**RFC 1939**, con estensioni **RFC 2449**. Si tratta di un protocollo estremamente semplice **basato SU TCP** per poter recuperare agevolmente i messaggi di posta elettronica e la porta usata è **la porta 110** da parte del server.

*Quindi cosa deve fare il client?* si deve connettere sulla **porta del server**.

Una volta aperta la connessione instaura un dialogo e questo dialogo è tipo di testuale, vengono scambiati messaggi di **7 bit**.

il client da una serie di comandi e il server risponde a questi comandi

i principali comandi sono:

- **Autenticazione:** Il client invia il comando **USER** seguito da **PASS**, rispettivamente per nome utente e password.
- **Lista dei messaggi (LIST):** Restituisce l'elenco dei nuovi messaggi con il numero d'ordine e la lunghezza in byte.
- **Recupero dei messaggi (RETR):** Permette di scaricare un messaggio specifico.
- **Cancellazione (DELE):** Rimuove il messaggio dal server una volta scaricato.

Se voglio memorizzare questi messaggi uso dei Buffer, cosa posso fare una volta che sono arrivati questi messaggi? posso scaricarli e quindi gli do un comando (**Retrive**).

**Per esempio:**

Con il comando **RETR 1** il Client richiede il contenuto del primo messaggio presente nella **Mbox** sul server. Questo comando consente di scaricare il messaggio e quindi di visualizzarne il contenuto, mantenendo però il messaggio anche sul server, nella **Mbox originaria**. Se, dopo aver scaricato una copia del messaggio, l'utente desidera eliminare l'originale dalla **Mbox** principale per liberare spazio, può farlo con il comando **DELE**. In tal modo, il messaggio viene rimosso definitivamente dal server. Se invece si vuole conservare il messaggio anche nella **Mbox** del server, basterà non eseguire il comando **DELE**.

## Estensioni a POP3

Sono state introdotte quindi delle estensioni al protocollo per far sì che l'utente possa vedere il mittente e il subject del messaggio prima di scaricarlo, quindi scaricare il messaggio parzialmente, scaricare solo l'**header**. Sfortunatamente queste estensioni non sono finite nello standard quindi non tutti i software hanno l'estensione.

Non è stato standardizzato perché hanno sviluppato un altro approccio, ossia il protocollo **IMAP**, la versione attualmente in uso è la 4 (RFC 3501).

## IMAP

Ci sono diverse versioni ma al momento usiamo la 4 il suo RFC ,quello originario, era il RFC 3501.

## Protocollo IMAP4: Accesso ai Messaggi sul Server di Posta

Con il protocollo **IMAP4** l'accesso alla **mailbox** del server di posta avviene tramite una **connessione esterna** e non direttamente dal server. La comunicazione avviene sulla **porta 143**.

- **Connessione e visualizzazione dei dati:** L'uso di **IMAP4** permette di visualizzare i messaggi sul server, **senza scaricarli** completamente sul dispositivo. Grazie a un meccanismo di **cache**, solo i dati a cui si accede di recente vengono memorizzati localmente sul client. Questa cache contiene i messaggi usati più frequentemente, mentre quelli meno recenti vengono eliminati per non eccedere la dimensione massima della **cache**.

- **Aggiornamento della cache:** Se l'utente cerca di aprire un messaggio non presente nella cache, questo viene temporaneamente copiato nella **cache**, sovrascrivendo le parti meno recenti. Questo sistema permette di mantenere un accesso rapido ai messaggi visualizzati di recente, mentre tutti i messaggi rimangono archiviati solo sul server e non vengono copiati in altre mailbox.

## Vantaggi e Svantaggi dell'IMAP

1. **Gestione dello spam:** Con IMAP, è possibile visualizzare l'oggetto dei messaggi prima di aprirli. Se un messaggio viene contrassegnato come "[spam]" nell'oggetto, è possibile decidere di non scaricarlo e ignorarlo.

2. **Archiviazione centralizzata:** I messaggi restano sul server, riducendo la necessità di spazio sul dispositivo client. In POP3, al contrario, è il Client a dover avere spazio sufficiente per archiviare tutti i messaggi.
3. **Accesso multi-dispositivo:** Il vantaggio principale di IMAP è la possibilità di accedere alla stessa mailbox da dispositivi multipli (es. smartphone, tablet, PC). Qualsiasi modifica (come la lettura di un messaggio) è immediatamente visibile su tutti i dispositivi, che sincronizzano lo stato della posta. Questa funzione richiede una gestione accurata degli accessi concorrenti per evitare conflitti tra i dispositivi.
4. **Sicurezza e autenticazione:** Sia POP3 che IMAP4 richiedono autenticazione, esponendo i dati a possibili intercettazioni. Per questo, sono state sviluppate versioni cifrate dei protocolli, che proteggono i dati tramite **crittografia end-to-end**, utilizzando la **porta 993**. Questa protezione garantisce che le credenziali e i dati rimangano sicuri anche durante il transito sulla rete.

Gli svantaggi invece sono:

1. **Dipendenza dalla connessione internet:** IMAP richiede una connessione stabile e continua per accedere e sincronizzare i messaggi sul server. Senza connessione, l'accesso alla posta è limitato solo ai messaggi già presenti nella cache.
2. **Maggiore utilizzo di spazio sul server:** Poiché i messaggi restano sul server, lo spazio disponibile potrebbe esaurirsi rapidamente, specialmente se non si eliminano i messaggi inutili o pesanti.
3. **Più risorse richieste:** IMAP può consumare più risorse (banda di rete e potenza di elaborazione) rispetto a POP3, a causa della sincronizzazione costante tra client e server.
4. **Gestione della cache:** Se la cache sul dispositivo non è ben gestita, potrebbe diventare lenta o causare problemi di prestazioni, soprattutto su dispositivi con poca memoria.
5. **Configurazione più complessa:** IMAP, con le sue opzioni avanzate, potrebbe risultare più complicato da configurare rispetto a POP3, specialmente per utenti meno esperti.

## Alternative: WebMail

**WebMail** rappresenta un'altra possibilità per accedere ai messaggi di posta elettronica, tramite browser, e generalmente si basa su un sistema di **cache**. WebMail consente di leggere e gestire i messaggi senza scaricarli fisicamente sul client, rendendolo comodo per accessi temporanei e riducendo la necessità di spazio locale.

## Protocollo FTP

Il protocollo **FTP** (File Transfer Protocol), **RFC 959**, è un protocollo di **livello applicativo** per la trasmissione di dati tra host basato su **TCP** (porta 21). È particolarmente utile per la gestione remota di file: permette di caricare file dal client al server (**upload**) o di scaricarli dal server al client (**download**).

Il protocollo utilizza due connessioni distinte:

1. Una connessione **di controllo** per scambiarsi i comandi e le risposte.
2. Una connessione **di dati** per trasferire i file veri e propri.

Esistono due modalità principali:

- **Modalità attiva (Active Mode):** Il client comunica al server su quale porta accettare i dati. Questo però può creare problemi con **firewall** e **NAT**.
- **Modalità passiva (Passive Mode):** È il server a indicare al client su quale **porta** connettersi per il trasferimento. In questo modo, è il client ad aprire entrambe le connessioni (controllo e dati), rendendo questa modalità più compatibile con reti moderne e **firewall**.

## Modalità attiva

Il server apre la **connessione dati** mentre il client apre la **connessione di controllo**. Per aprire la connessione dati il server deve conoscere l'**indirizzo IP** e il **numero di porta del client**.

Entrambi si trovano nella **connessione di controllo** del client sulla **porta 20**.

La **modalità attiva** è ormai **obsoleta** in quanto vi erano presenti diverse problematiche di sicurezza. Un utente infatti poteva mandare al server un **indirizzo IP** diverso dal suo in modo da far scaricare al server uno o più file su un altro dispositivo potendo compiere un possibile attacco **DDoS**.

## Modalità Passiva

Il client si connette al server sulla porta **21** per inviare i comandi e ricevere risposte. Questa è la connessione principale per comunicare. Quando il client vuole trasferire un file o ottenere una lista di directory, invia il comando **PASV**. Il server risponde fornendo un **indirizzo IP** e un **numero di porta** su cui è in ascolto **per la connessione dati**. Dopo aver ricevuto queste informazioni, il client apre una **connessione** verso la porta indicata dal server. Il trasferimento avviene attraverso questa connessione.

## Svantaggi FTP

Gli svantaggi principali del protocollo FTP sono:

- **Dati non crittografati:** Le credenziali (username e password) e i dati trasferiti viaggiano in chiaro sulla rete, rendendoli vulnerabili ad attacchi come lo *sniffing*.
- **Attacchi DDoS:** Nella modalità attiva, un client malevolo può sfruttare il protocollo per indirizzare traffico verso un dispositivo non coinvolto.

- **Mancanza di autenticazione forte:** FTP di base non supporta meccanismi avanzati di autenticazione.

## Anonymous FTP

**Anonymous FTP** è una modalità di accesso ai server **FTP** che permette agli utenti di connettersi **senza dover utilizzare credenziali di autenticazione personali**, come un nome utente e una password specifici. In questa modalità, il server consente agli utenti di accedere utilizzando un account predefinito chiamato "**anonymous**".

### Come funziona?

1. **Connessione al server:**
  - L'utente si collega al server **FTP** e utilizza "**anonymous**" come nome utente (username).
  - Per la password, è spesso richiesto un indirizzo email (che di solito non viene verificato), ma in molti casi può essere lasciato vuoto o inserito un valore fittizio.
2. **Accesso limitato:**
  - L'utente ha accesso solo a determinate aree del server, configurate dall'amministratore per il download o l'upload di file pubblici.
  - Non può accedere ad aree protette o riservate del server.

## HTTP / 1.0

In **HTTP/1.0**, il client apre una connessione, invia una richiesta al server e riceve una risposta; dopodiché, la connessione viene chiusa. Questo significa che le connessioni sono **non persistenti**.

Il formato della richiesta è:

1. **prima riga:** contiene il metodo (ad es. GET), la risorsa (radice / per la homepage), e la versione del protocollo. Ogni riga termina con un accapo.
2. **Header:** dopo la prima riga, si specificano gli header..  
L'header che è obbligatorio dalla definizione del protocollo è **l'header host**. L'idea del funzionamento è : ho aperto una connessione, per aprire questa connessione ho dovuto indicare l'indirizzo del server (info già data) ma la devo ripetere qui (**nel get**).  
**L'unico header obbligatorio** all'interno di una richiesta è quello dell'host.  
Alcuni header opzionali quali:
  - **User-agent:** serve per dire l'identità del browser.
  - **Accept-language:** it <cr><lf> , preferisco la versione italiana
  - **<cr><lf>:** Serve per andare a capo.

Riposta standard :

HTTP/1.0 200 Ok <cr><lf>

Segue poi la parte degli header, seguita da una riga vuota e infine dal **body**.

### Header nella risposta

- **Connection close:** indica che la connessione sarà chiusa dopo la risposta (non persistente).
- **Date:** data di invio della risposta in GMT (per rendere indipendente il fuso orario).
- **Server:** versione del server contattato.
- **Last-Modified:** data dell'ultima modifica del file inviato, utile per gestire il caching.
- **Content-Length:** lunghezza del file in byte.
- **Content-Type:** tipo di contenuto, espresso in formato **MIME** (es. text/html).

## Proxy e Caching

Il **proxy** è un elemento intermedio tra client e server, che agisce come cache per i file del server. Il suo scopo principale è **ottimizzare le comunicazioni e migliorare le prestazioni**, grazie all'utilizzo di una cache per archiviare copie temporanee dei file originariamente presenti sul server. Dal punto di vista del client, il funzionamento non cambia: sembra di comunicare direttamente con il server, ma in realtà ci si sta interfacciando con il **proxy**.

La cache del proxy potrebbe contenere file obsoleti rispetto alla versione più aggiornata presente sul server. Per mantenere la consistenza, è necessario verificare periodicamente se i file nella cache sono ancora validi o se devono essere aggiornati.

Modo per risolvere questa cosa: Abbiamo visto che oltre al metodo **GET** abbiamo un metodo chiamato **head**; La differenza è che **GET** contiene tutta la parte di intestazione più file mentre **head** contiene solo la parte intestazione e non il file.

Possiamo chiedere il file tramite il metodo **head**. L'header aggiunto si chiama **last modified since** seguito da una data **GMT**.

La risposta da parte del server può essere di 2 tipi diversi: può essere **200 ok** con contenuto del file oppure ci può essere una risposta che è la risposta **304 not modified**.

In questa versione abbiamo un metodo di tipo get che è un get condizionale, ovvero condizionato con la data **last modified**.

**Richiesta condizionale** serve per verificare se il file è stato modificato nel caso il server risponde col file aggiornato.

## Stato e Cookie

Il protocollo **UDP** è senza stato (non conserva alcuna informazione). Se voglio fare una serie di richieste consecutive sarebbe molto utile sapere cosa ha fatto il server prima. E quindi dopo aver definito il protocollo senza stato si è creato questo meccanismo chiamato **cookie**.

Come funziona il meccanismo dei cookie? In linea di principio funziona così:

**set-Cookie:** Il server ad un certo punto decide di mandare l'**header set cookie**, deve essere seguita da una stringa di caratteri, quello che viene specificato dopo questo header viene praticamente letto dal client, il client prende nota di questo cookie, se non deve più contattare quel server allora quel cookie è stato inutile, se invece il client deve effettuare altre operazioni sul server, per esempio mandare richieste con altri **metodi get**, e nel momento in cui manda richieste aggiunge il **set cookie**. Prende la stringa di caratteri successiva e rimandarla al server sotto forma di cookie. La volta dopo il server si vede ritornare quella cosa (la sequenza) che aveva definito lui. È un modo per dire al server quello che ha fatto prima. Il Client così tiene traccia di quello che il server esegue prima. Il client se lo ricorda e lo manda indietro al server.

***Protocollo senza memoria ma con l'utilizzo dei cookie ci permette di tenere memoria di quello che fa il server.***

Il cookie viene definito da: un nome = valore

**set-cookie:** nome = valore

il Client si vede arrivare le informazioni e la interpreta come se fosse una dichiarazione di una variabile. E quando mando il prossimo metodo al server ripeto questa dichiarazione di variabile. Il fatto che i cookie abbiano un nome ci permette di settare più cookie. Questi cookie possono avere significati diversi ed essere utilizzati allo stesso momento. Ogni server gestisce i suoi cookie in modo totalmente indipendente da altri server. Si mette in relazione questa coppia con l'indicazione dell'host che viene contattato.

**I cookie possono scadere nel tempo**, possono avere un periodo di validità e dopodiché scadere. Succede che il Client manderà i cookie che si possono usare al momento mentre non invia i cookie scaduti.

Questa scadenza può essere definita in diversi modi, per esempio sotto forma di data con ad esempio il comando **cookie-expired**.

Un altro modo potrebbe essere quello di dire "questo cookie sarà valido per tot minuti / tot giorni" quindi **cookie a breve termine**.

Un altro modo per dare scadenza a questi cookie è quello di dire che il cookie è valido solo durante una sessione di lavoro.

I cookie della sessione possono essere memorizzati in una struttura dati del browser. I cookie permanenti devono essere memorizzati all'interno del **file system del client**.

Come li distinguo? guardando la data di scadenza, nel set cookie possono esserci parametri aggiuntivi tra cui la data di scadenza.

**Tracciamento** = circostanza in cui il server ha interesse di identificare il client

Cookie di tracciamento: do un identificatore per esempio **UTD = 5**.



# HTTP / 1.1

Serie più vecchia in uso, questa versione è definita nell' **RFC 2068** e successivamente aggiornata con la **versione 2616**.

La versione 1.1 introduce le **connessioni persistenti**, consente un guadagno di prestazioni nel caso in cui il server è lontano dal client dove il **RTT** influenza negativamente le prestazioni della comunicazione.

Tuttavia, in altre circostanze server e client sono vicini e la latenza è abbastanza bassa questo guadagno può non essere così marcato, per questo motivo sono stati introdotti ulteriori miglioramenti.

L'ottimizzazione più rilevante è l'introduzione del concetto della **pipeline**.

Il client apre una connessione verso il server e poi invia una o più richieste su questa connessione e riceve delle risposte dal server su questa stessa connessione. Anche mantenendo connessioni persistenti senza l'uso della pipeline succede che il client manda la richiesta e aspetta la risposta del server e successivamente può inviare una seconda richiesta.

**Pipeline:** il client non aspetta di ricevere una risposta prima di inviare una richiesta successiva, se ha più risorse da richiedere, le richiede senza aspettare le risposte, le risposte arriveranno.

Pipeline facile da implementare se il server utilizza un singolo thread per mandare le risposte.

**Cosa deve fare il server?** leggere la prima richiesta e inviare la prima risposta, legge poi la seconda richiesta e manderà la seconda risposta ecc.

Il tempo necessario per produrre le risposte dipende dalle richieste.

Ci metterò poco tempo se il file è corto, potrei avere un ritardo se il file è troppo grande.

**Come possiamo velocizzare il funzionamento del server ?** attraverso questi modi:

- Ottimizzare il parsing delle richieste.
- Implementare un **meccanismo di caching** per risposte frequenti.
- Limitare il numero massimo di thread attivi per evitare il sovraccarico.
- Migliorare il tempo di esecuzione delle operazioni di **I/O**.

Se i Client sono più di uno, per esempio 2. Avrò due thread che faranno il parsing delle richieste e poi questi lanceranno i vari thread di risposta. Se pongo un limite massimo di thread attivi l'efficienza dipenderà dal numero di client e di capacità del sistema.

## Versione 2.0

è definita in **RFC 7540**. Introduce diverse ottimizzazioni per migliorare le prestazioni e ridurre la latenza. La principale innovazione è il **multiplexing**, che consente di gestire più richieste e risposte contemporaneamente su una singola connessione, evitando rallentamenti causati dal "blocco della coda".

**vantaggi di HTTP/2 rispetto a HTTP/1.1:**

- **Miglior utilizzo della banda:** Grazie alla compressione degli header e al multiplexing.
- **Riduzione della latenza:** Elimina molte delle inefficienze di **HTTP/1.1**, come il problema del blocco della coda (head-of-line blocking).

- **Migliore esperienza utente:** Risorse caricate più velocemente grazie al server push e alla prioritizzazione.

### Svantaggi di HTTP/2:

- **Richiede TLS (HTTPS)** nella maggior parte dei casi:
  - Sebbene non sia obbligatorio, HTTP/2 è quasi sempre implementato con **TLS**, il che può introdurre un sovraccarico iniziale per l'handshake.
- **Head-of-Line Blocking a livello TCP:**
  - Poiché si basa su TCP, il blocco della coda può ancora verificarsi quando un pacchetto viene perso, anche se il multiplexing riduce il suo impatto.

### Versione 3.0

definita in **RFC 9114**, innovazione è quella di abbandonare l'uso del **trasporto TCP** con l'uso di **UDP**. Vantaggi nell'uso di **UDP** come protocollo di trasporto è:

Evitare il **3-way handshake**, come svantaggio è la poca affidabilità.

Un altro vantaggio è poter comprendere meglio dopo aver visto il meccanismo chiamato **controllo di congestione**. Obiettivo fallito del **TCP** che rende il protocollo **TCP** meno utile dal punto di vista pratico rispetto **UDP**.

Il protocollo **TCP** prevede un meccanismo di controllo di flusso.

### Congestion control TCP

Il controllo di congestione **TCP** serve a evitare che i router intermedi perdano messaggi a causa del riempimento dei loro buffer di ricezione. Ogni router ha un buffer per ricezione e trasmissione: se il buffer di ricezione si satura, il router non può più inoltrare i dati. Per affrontare questo problema, il TCP include un meccanismo per regolare dinamicamente la quantità di dati inviati.

Come funziona:

- 1. Congestion Window (cwnd):** È una finestra che indica quanto spazio è disponibile nei **buffer** di rete lungo il percorso. Per calcolarla, si usa un algoritmo che parte con una fase di **slow start**.
- 2. Slow Start:** Si inizia inviando piccoli blocchi di dati e, ad ogni conferma di ricezione (**ACK**), la dimensione del blocco raddoppia (1, 2, 4, 8...). Questo continua fino a quando:
  - Non si ricevono più **ACK** (segno di congestione).
  - Si raggiunge il limite massimo della finestra di congestione.
- 3. Congestione rilevata:** Quando non arriva un **ACK**, significa che il buffer di un router lungo il percorso è pieno. La finestra di congestione viene ridotta, e si procede a testare incrementi più gradualmente per trovare il valore massimo sostenibile.

Ad esempio, se si perde un **ACK** con un blocco di dimensione 8, sappiamo che il limite è tra 4 e 8. Si procede allora con incrementi più piccoli per stabilire il valore preciso.

Problemi principali:

- **Condivisione della rete:** Altri utenti o connessioni (anche UDP) possono usare gli stessi router e riempire i buffer.
- **Dinamismo:** La disponibilità dei buffer cambia continuamente.

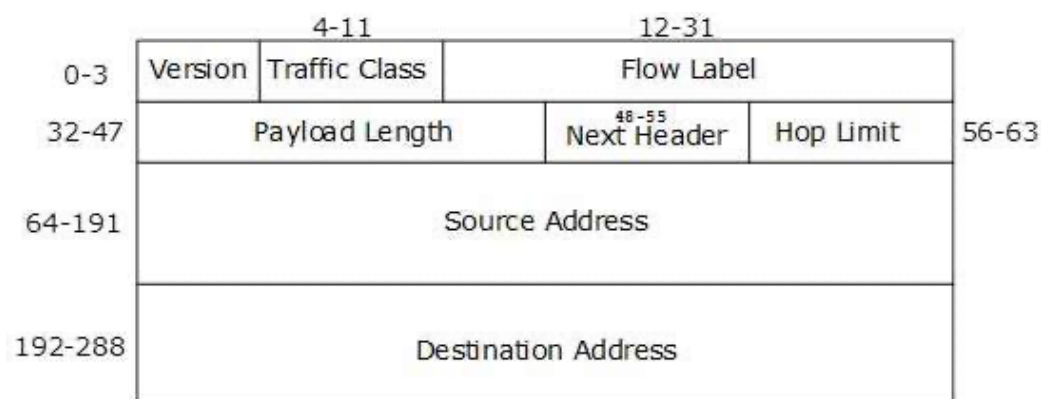
### Soluzione TCP:

TCP combina controllo di flusso e controllo di congestione. Per determinare la dimensione massima dei dati da inviare, usa il valore più piccolo tra la congestion window e la receive window (lo spazio disponibile sul buffer del destinatario).

Questo meccanismo consente a **TCP** di adattarsi dinamicamente, evitando perdite e mantenendo l'efficienza della rete.


## IPV6

Nei primi anni '90 si iniziò a sviluppare un successore di **IPv4**. Una prima motivazione di tale sforzo era legata alla considerazione che lo spazio di **indirizzamento IP a 32 bit** stava cominciando ad esaurirsi. Per soddisfare la necessità di un grande spazio di indirizzamento venne sviluppato il protocollo **IPv6**.



*header ipv6*

Version	IHL	Type of service		Total length		
Identification				D F	M F	Fragment offset
Time to live	Protocol		Header checksum			
Source address						
Destination address						
Options (0 or more words)						


  
**32 bits**

*header ipv4*

La lunghezza del **payload** può essere confrontata con il campo "**Lunghezza Totale**" della versione **IPv4**. Tuttavia, mentre in **IPv4** tale campo include anche l'**header**, in **IPv6** la lunghezza si riferisce esclusivamente al payload. Il campo *Next Header* svolge la stessa funzione del *Protocol* di **IPv4**: specifica il protocollo utilizzato per la decodifica del payload in **IPv6**. Inoltre, nel campo *Next Header*, è possibile specificare un altro header del livello 3. Infine, *Hop Limit* svolge la stessa funzione di **Time to Live (TTL)** in **IPv4**. La differenza principale è che, in **IPv6**, il mittente specifica il **numero massimo di hop** che il pacchetto può attraversare prima di essere scartato. Differenza significativa è il numero usato dei bit, indirizzi **ipv6 128 bit** mentre **ipv4 32**. **Le principali differenze** tra **IPv4** e **IPv6**, oltre alla lunghezza degli indirizzi, includono alcune caratteristiche fondamentali che saltano immediatamente all'occhio. In **IPv4**, l'header può avere una lunghezza variabile, con un numero determinato di campi fissi seguiti da campi opzionali. Questo approccio complica la decodifica dell'header da parte dei router, poiché i campi opzionali raramente vengono utilizzati nella pratica. Al contrario, **IPv6** utilizza un header di lunghezza fissa, che rappresenta un vantaggio significativo per la semplificazione della decodifica da parte dei router. Tuttavia, l'header fisso presenta alcuni svantaggi, come la ridotta flessibilità rispetto alla possibilità di aggiungere opzioni direttamente nell'header principale. Un'altra differenza sostanziale riguarda il trattamento della **frammentazione**. In **IPv4**, l'header include campi come *Identification*, *Flags*, e *Fragment Offset*, che gestiscono la frammentazione dei pacchetti. In **IPv6**, invece, non esiste più questa gestione della frammentazione a livello di rete; l'obiettivo è stato quello di eliminare completamente la frammentazione. **La frammentazione, in sintesi**, si verifica quando un pacchetto di rete deve essere diviso in parti più piccole per adattarsi ai limiti di dimensione del livello sottostante, come il livello **datalink**. Ad esempio, può accadere che un canale di comunicazione tra host e router sia in grado di gestire pacchetti di grandi dimensioni, ma altri nodi intermedi, con limiti più restrittivi, necessitano di frammentare i pacchetti. In **IPv6**, si è scelto di delegare la gestione di queste situazioni agli **endpoint**, evitando di gravare i router con questo compito. La frammentazione in modo semplice è : prendo il mio datagramma troppo

lungo e lo suddivido in due pezzi molto piccoli, al posto di mandare un solo pacchetto lo divido in 2 e mando le due metà. Come distinguo un pacchetto intero in uno suddiviso in 2 parti? lo faccio usando i campi ID. Fragmentation offset mi dice in che posizione si trova il frammento rispetto al datagramma originario. Il motivo per cui si è tolta la frammentazione è il seguente : Una volta arrivati a destinazione, arriva un frammento, un altro ecc e quando sono arrivati tutti li devo mettere insieme per riformare il datagramma originario. Questi frammenti non è detto che arrivino in ordine giusto, dell'ordine d'arrivo usiamo fragmentation offset ma se per caso mi arriva il 3 frammento per primo io devo prevedere di usare un buffer in cui mi aspetto che arriveranno altri 100 byte (nel caso ci sia scritto 1000 nel fragmentation offset). Dovrò aspettare una bella quantità di tempo e quindi c'è il rischio di caricare troppo il lavoro. Quindi si è deciso di levarla per evitare sprechi di memoria. Nell'ipv4 appena sopra al payload c'erano dei campi opzionali, potevano servire per definire del trattamento particolare di questi datagrammi..per esempio fornire delle caratteristiche di sicurezza usando tecniche crittografiche. Mettere come campo opzionale per esempio un algoritmo di crittografia. In IPv6, se vogliamo aggiungere funzionalità extra, possiamo utilizzare gli **Extension Headers**, che vengono posizionati tra l'header principale e il payload. Questi header permettono di specificare operazioni aggiuntive, come la compressione del payload, la crittografia, o altre configurazioni particolari. Gli **Extension Headers** funzionano come una serie di blocchi aggiuntivi che vengono letti sequenzialmente. Ogni header specifica il tipo di operazione richiesta e include un campo *Next Header* che indica se c'è un altro header aggiuntivo da elaborare o se si è arrivati al payload vero e proprio. Questa struttura rende IPv6 più flessibile, consentendo di aggiungere funzionalità avanzate senza appesantire l'header principale o complicare la gestione del protocollo. Dobbiamo operare la rete sapendo che la situazione non è uniforme in tutta la rete, alcuni router usano la versione 4 mentre altri usano la versione 6. Se voglio far comunicare questi router devo avere un modo per farlo, questo metodo è in parte facilitato nel riconoscere il protocollo usato all'interno del nostro datagramma. Leggiamo i primi 4 bit e leggiamo se si tratta della versione 4 o 6 . La versione 6 è compatibile con la versione 4 ovvero è in grado di funzionare come un datagram di ipv4 se riceve la versione 4.

## Routing

Il **routing** è il processo attraverso il quale i pacchetti di dati vengono inviati da un nodo (host o router) a un altro, attraversando la rete. Ogni router in una rete ha il compito di decidere come inoltrare i pacchetti sulla base dell'indirizzo di destinazione. Quando un pacchetto raggiunge un router, quest'ultimo esamina l'indirizzo di destinazione e consulta la sua **tabella di routing**, per determinare il percorso migliore verso la destinazione finale. Ogni tabella di routing contiene una serie di informazioni che associano un indirizzo di destinazione a un "next-hop", che rappresenta il router successivo da raggiungere, fino a quando il pacchetto non raggiunge la sua destinazione.

# Tabella di Routing

La **tabella di routing** è essenziale per il corretto funzionamento di un router. Essa viene popolata con informazioni che derivano da **algoritmi di routing** e **protocolli di routing**, che consentono al router di determinare il miglior percorso per i pacchetti. Ogni entry della tabella di routing include l'indirizzo di rete di destinazione e l'indirizzo del router successivo da raggiungere (next-hop). La tabella viene aggiornata periodicamente in base ai cambiamenti nella topologia della rete e alle informazioni ricevute dai router vicini. Ad esempio, se un router riceve un pacchetto destinato a un indirizzo IP che non è direttamente connesso a lui, esamina la sua tabella di routing per determinare quale router intermedio (next-hop) deve inviare il pacchetto. Questa decisione si basa sulle informazioni sulla rete, che includono le informazioni sulla distanza, la latenza o altri parametri di costo associati ai vari percorsi.

## Algoritmi di Routing

Per determinare il percorso migliore, si utilizzano diversi **algoritmi di routing**, che sono progettati per calcolare il percorso più efficiente verso una destinazione. Tra i più comuni:

### 1. Algoritmo di Dijkstra

È un algoritmo utilizzato per determinare il percorso più breve tra un nodo di origine e tutti gli altri nodi di una rete. Questo algoritmo è alla base di protocolli come **OSPF (Open Shortest Path First)**, che è ampiamente usato nelle reti di grandi dimensioni. Dijkstra è un algoritmo che funziona con una visione completa della rete, esaminando tutti i possibili percorsi per selezionare quello con il costo più basso (in termini di latenza, banda, o numero di hop).

### 2. Path Vector (BGP)

In grandi reti interdominio, come l'Internet, viene utilizzato l'algoritmo **Path Vector** nel protocollo **BGP (Border Gateway Protocol)**. BGP è utilizzato per il routing tra sistemi autonomi (AS) e si basa su una visione parziale della rete. Invece di calcolare il percorso più corto, come nei precedenti algoritmi, BGP seleziona i percorsi in base a politiche di routing configurabili, come il numero di hop attraversati, la preferenza del percorso o la larghezza di banda disponibile.

Oltre agli algoritmi, il **protocollo ICMP (Internet Control Message Protocol)** è fondamentale per il funzionamento della rete e per l'instradamento dei messaggi.

Un algoritmo *link-state* funziona facendo in modo che tutti i nodi della rete conoscano la mappa completa della rete, cioè come sono collegati tra loro e quanto "costa" ogni collegamento.

Un esempio famoso di questo tipo di algoritmo è **Dijkstra**. Funziona così:

1. **Raccolta delle informazioni:** ogni nodo della rete deve sapere com'è fatta la rete, quindi i nodi si scambiano informazioni sui collegamenti e i loro costi usando un messaggio speciale (broadcast).
2. **Creazione dell'albero dei percorsi migliori:** dato un nodo di partenza, l'algoritmo di Dijkstra calcola i percorsi più economici da quel nodo verso tutti gli altri, creando una specie di mappa dei percorsi migliori.
3. **Costruzione della tabella di forwarding:** con i dati raccolti, ogni nodo costruisce la propria tabella per sapere dove inviare i pacchetti per raggiungere ogni destinazione nel modo più efficiente.
4. **Funzionamento iterativo:** il calcolo si ripete più volte. Dopo  $k$  iterazioni, l'algoritmo conosce i percorsi per tutti i nodi che si trovano a una distanza  $k$  dal nodo di partenza.

Se un nodo non ha informazioni sul costo di un collegamento, assume che sia *infinitamente costoso*, cioè inutilizzabile.

Un algoritmo **distance-vector** funziona in modo diverso rispetto a un algoritmo *link-state*. Invece di conoscere tutta la topologia della rete, ogni nodo sa solo quali sono i suoi vicini e a quale "costo" può raggiungerli.

### Come funziona?

Ogni nodo mantiene e aggiorna una tabella chiamata **tabella delle distanze**, che indica la distanza minima (o il costo) per raggiungere ogni altro nodo della rete. L'algoritmo segue questi passaggi:

1. **Inizializzazione:** ogni nodo conosce solo i costi per raggiungere direttamente i suoi vicini. Per gli altri nodi, il costo iniziale è considerato infinito.
2. **Scambio di informazioni:** periodicamente, ogni nodo invia la sua tabella delle distanze ai suoi vicini. A loro volta, i vicini inviano le loro tabelle agli altri nodi.
3. **Aggiornamento delle distanze:** quando un nodo riceve una tabella da un vicino, la confronta con la propria e aggiorna i percorsi. Se trova un percorso più economico passando per un vicino, lo usa al posto del precedente.
4. **Convergenza:** dopo un po' di iterazioni, tutti i nodi conoscono i percorsi migliori per raggiungere ogni altro nodo.

## Protocollo ICMP

Il **protocollo ICMP (Internet Control Message Protocol)** è fondamentale per il funzionamento della rete e per l'instradamento dei messaggi. ICMP è utilizzato per la diagnostica e il controllo delle reti, permettendo la gestione dei messaggi di errore e il test della raggiungibilità tra host e router. Ad esempio, ICMP è utilizzato per il comando **ping**, che invia

un messaggio **Echo Request** e attende una risposta **Echo Reply**. Questo permette di verificare se un host è raggiungibile e quanto tempo impiega il pacchetto a viaggiare in rete.

Inoltre, ICMP viene utilizzato per segnalare errori di rete, come ad esempio quando un pacchetto non può essere consegnato (ad esempio, "Destination Unreachable" o "Time-to-Live exceeded"). Ogni volta che un pacchetto supera il numero massimo di hop consentiti (TTL, Time To Live), ICMP segnala l'errore al mittente. ICMP svolge anche un ruolo fondamentale nella scoperta della rete, aiutando i router a determinare i **vicini** (attraverso messaggi di tipo "Router Solicitation" e "Router Advertisement") e a mantenere aggiornata la topologia di rete.

## Ethernet

### *Standard Industriale IEEE 802.3*

La prima versione di Ethernet risale alla seconda metà degli anni '70 e ai primi anni '80. I primi esperimenti sulla tecnologia Ethernet furono condotti nel 1973, 1978 e 1980, ma lo **standard IEEE 802.3** fu ufficialmente pubblicato nel 1983. **IEEE 802.3** è uno standard di rete sviluppato dall'Institute of Electrical and Electronics Engineers (**IEEE**) per specificare il funzionamento delle reti cablate basate sulla tecnologia Ethernet. Definisce i parametri fisici e il controllo di accesso al mezzo (**MAC**), specificando come i dispositivi in una rete locale (**LAN**) possono comunicare.

Per identificare le diverse versioni del protocollo Ethernet, si utilizzano sigle come **10Base5**. In questa denominazione:

- **10** indica la velocità di trasmissione della rete in Mbps (10 Megabit al secondo);
- **Base** significa che la comunicazione avviene in banda base, ovvero senza modulazione di frequenza;
- **5** rappresenta il tipo di cavo utilizzato e la sua lunghezza massima supportata in centinaia di metri (nel caso di 10Base5, 500 metri).

Lo standard **IEEE 802.3** definisce sia il **livello fisico** (Livello 1) che il **livello di collegamento dati** (Livello 2) del modello OSI, regolando il modo in cui i dati vengono trasmessi e ricevuti sulla rete.

### **Il livello fisico**

Il mezzo trasmissivo inizialmente utilizzato era il **cavo coassiale**, costituito da:

- Un conduttore interno;
- Uno strato isolante elettrico;
- Un secondo conduttore esterno, realizzato con una calza metallica;
- Un ulteriore strato isolante per proteggere l'intero cavo.



L'idea originale era di installare questi cavi nei pavimenti delle stanze, ma ciò creava ingombri e intralci. Per risolvere il problema, si svilupparono tecniche per la realizzazione di pavimentazioni rialzate, sotto le quali far passare i cavi.

Un singolo cavo poteva estendersi per una certa lunghezza (ad esempio, 15 metri) e permetteva la trasmissione di messaggi tra i dispositivi collegati. Il principio di funzionamento prevedeva che più dispositivi potessero connettersi allo stesso cavo e comunicare tra loro inviando e ricevendo messaggi.

### **Connettività e problemi tecnici**

Nel caso di **10Base5**, la connessione dei dispositivi al cavo coassiale avveniva tramite un **morsetto**. Questo morsetto conteneva una vite che perforava il cavo e metteva in contatto il conduttore interno con il dispositivo.

Vantaggi di questa connessione:

- Possibilità di collegare più dispositivi allo stesso cavo.

Tuttavia, presentava anche diversi problemi:

- **Cortocircuiti** e cattivi contatti dovuti alla perforazione del cavo;
- **Problemi elettromagnetici**, con segnali che potevano riflettersi e creare interferenze (eco);
- **Distanza tra le connessioni**, che poteva influire sulla qualità della trasmissione.

Per evitare riflessioni del segnale e disturbi nella connessione, era necessario terminare il cavo con terminatori adeguati. Tutti questi aspetti sono dettagliatamente descritti nello standard **IEEE 802.3**, fornendo agli utenti le informazioni necessarie per implementare reti affidabili.

### **Trasmissione dei dati**

Nel caso di una rete con tre dispositivi (H1, H2 e H3), se H2 voleva inviare un messaggio a H3, doveva codificare i dati sotto forma di bit e trasmetterli uno alla volta. La trasmissione avveniva utilizzando un segnale modulato.

Ethernet utilizza un **canale di tipo broadcast**, ossia un canale in cui qualsiasi dispositivo può inviare una trasmissione e tutti gli altri possono riceverla, proprio come avviene con i segnali radio.

### **Il protocollo CSMA/CD**

Se anche H1 volesse inviare un messaggio a H3 contemporaneamente a H2, potrebbero verificarsi problemi di interferenza. Se entrambi trasmettono uno "0" il sistema potrebbe funzionare, ma se uno trasmette "0" e l'altro "1", si genera una collisione. Per gestire questa problematica, è stato sviluppato il protocollo **CSMA/CD** (Carrier Sense Multiple Access with Collision Detection).

### **Funzionamento di CSMA/CD**

Ogni dispositivo segue lo stesso comportamento. Ad esempio, nel caso di H2:

1. H2 vuole inviare un messaggio, quindi ascolta il cavo per verificare se qualcuno sta già trasmettendo.
2. Se il cavo è libero, può iniziare la trasmissione.
3. Se due dispositivi trasmettono contemporaneamente, si verifica una collisione.

In caso di collisione, il protocollo prevede i seguenti meccanismi di gestione:

- **Jamming**: i dispositivi interrompono l'invio del messaggio e trasmettono un segnale di disturbo per informare tutti della collisione.
- **Attesa variabile**: ciascun dispositivo attende un intervallo di tempo casuale prima di riprovare la trasmissione. Questo riduce la probabilità di una nuova collisione.

### 10Base2 e il cavo RG58

Nel 1988, la specifica **10Base2** introdusse una modifica al livello fisico, sostituendo il cavo originario con un cavo coassiale più sottile, chiamato **RG58**. Questo nuovo standard utilizzava connettori di tipo **BNC**, facilitando l'installazione e riducendo i problemi meccanici delle connessioni dirette.

## Indirizzamento MAC e indirizzo broadcast

Un aspetto fondamentale dello standard IEEE 802.3 riguarda la gestione degli indirizzi. Gli indirizzi **MAC** (Media Access Control) sono identificatori univoci di livello 2 e sono composti da 6 byte (48 bit). Anche se in una rete locale di piccole dimensioni basterebbero meno byte, l'uso di un indirizzo a 6 byte garantisce un numero enorme di combinazioni ( $2^{48}$  dispositivi unici).

Per evitare conflitti di indirizzi, lo spazio degli indirizzi MAC è suddiviso:

- I primi 3 byte identificano il costruttore della scheda di rete.
- Gli ultimi 3 byte sono assegnati univocamente dal costruttore.

Dato che il numero di produttori di schede di rete è inferiore a  $2^{24}$ , i grandi produttori hanno acquisito più prefissi per garantire un'assegnazione equilibrata degli indirizzi.

Se due dispositivi avessero per errore lo stesso indirizzo MAC, non ci sarebbero problemi finché non vengono collegati alla stessa subnet.

## Indirizzo Broadcast

Ethernet è stato progettato per operare in modalità broadcast, permettendo la trasmissione di messaggi a tutti i dispositivi sulla rete. Per questo motivo, si è deciso di riservare un indirizzo **MAC** speciale per il broadcast: **FF-FF-FF-FF-FF-FF**. Qualsiasi dispositivo che riceve un pacchetto con questo indirizzo deve accettarlo, indipendentemente dal proprio indirizzo **MAC**.

Grazie a questa architettura, Ethernet supporta sia la comunicazione unicast (tra due dispositivi) che broadcast (a tutti i dispositivi della rete).

## Hub e switch

Hub mantiene l'idea che viene chiamata dominio di collisione che implica l'uso del protocollo **CSMA/CD** mentre lo switch toglie di mezzo l'uso del protocollo **CSMA/CD** attraverso la tecnica del **store & forward**. Questi 2 tipi di rete sono simili ma hanno caratteristiche completamente diverse tra di loro.

***Quali sono i difetti della hub con CSMA ?*** il difetto principale oltre della banda limitata di comunicazione abbiamo questa idea che la collisione può avvenire (collisione significa perdita di messaggi e significa anche perdita di tempo). Nel caso di collisione dobbiamo aspettare una certa quantità di tempo prima di trasmettere e starei usando in modo inefficiente la banda di comunicazione.

Questo modo di gestire le collisioni implica una certa serie di complicazioni, la principale è che ci vuole un **buffer di trasmissione**. Questo buffer serve per gestire la ritrasmissione dello stesso **frame** e finché non ho superato l'intervallo di vulnerabilità non ho la garanzia che non ci sia un problema di trasmissione. Nella **versione switch** la situazione è diversa, abbiamo dei **buffer** ma non sono più buffer di trasmissione inseriti all'interno dell'host mittente. La comunicazione sarà **tra host mittente e switch** e non è soggetta a nessuna collisione. Con l'uso di switch sembra che si aumenti l'affidabilità e sembra che non sia necessario dover ritrasmettere gli stessi dati.

Questo è vero fino a un certo punto, è vero fino al punto in cui questi dispositivi hanno sufficienti **buffer liberi**. ***Cosa potrebbe succedere di grave per impedire la comunicazione?*** Un problema grave che potrebbe impedire la comunicazione è la scarsità di spazio nei buffer di uno switch. Se non c'è abbastanza memoria per tutti i frame in transito, alcuni pacchetti potrebbero essere scartati, compromettendo la trasmissione dei dati.

***Come si verifica questa carenza di buffer?*** un caso abbastanza evidente potrebbe essere quando usiamo un tipo di comunicazione broadcast. Nel caso di comunicazione broadcast si può creare un momento in cui i buffer non possono essere sufficienti per raccogliere tutti i frame che stanno viaggiando. ***Differenza tra switch più costoso e meno costoso?*** avere degli atteggiamenti diversi, semplicemente dal tipo di prezzo che l'azienda impone, un altro modo potrebbe essere quello di avere uno switch con più memoria o meno memoria. Avere più memoria mi permette di perdere meno passaggi dovuti al traffico di rete.

Abbiamo un **preambolo** per sincronizzare i clock dei vari dispositivi, poi abbiamo **indirizzo di destinazione** e poi quello sorgente che sono indirizzi **MAC** da 6 byte ciascuno e poi un campo 2 byte che viene chiamato tipo (type) poi abbiamo la parte dati (payload) e poi abbiamo altri 4 byte che sarebbero il controllo **CRC** per la verifica di integrità. Questa **verifica di integrità** serve per rilevare la presenza di collisioni. L'aggiunta di **CRC** (controllo di integrità) alla fine serve per verificare che tutti i bit ricevuti siano corretti.

La dimensione di payload è variabile e varia da un minimo di 46 a un massimo 1500 byte.

Dimensione minima di 64 byte. I 64 byte fanno riferimento da dest a crc. ***Come faccio a sapere***

quanto è lungo il payload? il campo type in realtà sarebbe la lunghezza del frame che può variare da un minimo di 46 fino a un massimo da 1500 byte, uso 2 byte per decodificare questa lunghezza.

Con questa idea io posso usare una quantità inferiore di byte per i frame corti e una quantità maggiore per i frame più lunghi.

## MTU

A livello superiore, il protocollo di rete deve rispettare il vincolo della **MTU**, ovvero la dimensione massima del frame trasmissibile dal datalink.

### Versione switch

Per risolvere il problema di non sapere a che indirizzo corrisponda il frame possiamo usare un'opportuna interfaccia. Ogni modello di switch può avere un meccanismo diverso dagli altri, per vedere i dettagli di realizzazione dobbiamo vedere marca, modello ecc.

In contesti **plug and play**, l'obiettivo è semplificare la configurazione e garantire una gestione automatica delle impostazioni di rete.

Inoltre, è fondamentale prestare attenzione alla sicurezza: **attacchi di tipo sniffing**, che includono l'alterazione degli indirizzi **MAC**, rappresentano un rischio poiché permettono a un malintenzionato di intercettare e analizzare il traffico di rete. **Negli hub**, in cui il traffico viene inviato a tutte le porte, questa vulnerabilità è particolarmente critica perché il filtro sui frame è assente, facilitando lo spionaggio delle schede di rete. Al contrario, **gli switch**, grazie alla loro capacità di inviare i frame solo al destinatario previsto, offrono un livello superiore di protezione contro tali attacchi, a condizione che siano configurati correttamente e dotati di adeguate misure di sicurezza.

## Protocollo ARP e Sicurezza

Il protocollo **ARP** (Address Resolution Protocol) è utilizzato per mappare un indirizzo IP a un indirizzo **MAC**. Si tratta di un protocollo molto semplice di tipo **domanda-risposta**, che opera a livello 2 pur coinvolgendo sia indirizzi **IP** che **MAC**. In pratica, si può predisporre un **frame Ethernet** contenente il messaggio **ARP** e inviarlo in modalità broadcast a tutte le macchine della rete, chiedendo "Chi possiede questo indirizzo IP?".

In questo frame, l'indirizzo di destinazione viene impostato su quello di broadcast, il campo source contiene il proprio indirizzo **MAC**, mentre il payload trasporta l'**indirizzo IP** della macchina cercata. **ARP** è essenziale per inviare messaggi all'interno della rete locale, in quanto non attraversa router intermedi: serve esclusivamente a determinare il **MAC** di un host per poter inviare uno o più frame direttamente a quel dispositivo.

Tuttavia, **ARP** presenta alcune inefficienze dovute al suo meccanismo di comunicazione broadcast, motivo per cui il protocollo viene utilizzato il meno possibile e supportato da un sistema di caching per ridurre il traffico. Uno dei principali problemi di sicurezza risiede nel rischio di cache poisoning: un attaccante può infatti inviare **risposte ARP** fasulle per alterare la cache, dirottando il traffico o eseguendo attacchi man-in-the-middle.

Per difendersi da queste vulnerabilità, si può optare per configurazioni manuali della rete che eliminino la necessità di **ARP** oppure implementare misure di sicurezza come il **DHCP** snooping e il **Dynamic ARP Inspection (DAI)**, che permettono di verificare la coerenza delle informazioni **ARP** e mitigare il rischio di **cache poisoning**.

## Mobilità nei Sistemi Wireless e Gestione degli Access Point

Per migliorare la connessione in ambienti wireless, è possibile implementare più **access point (AP)** distribuiti in modo strategico. Il protocollo wireless prevede lo spostamento di un client da un **AP** all'altro senza interruzioni della connessione, garantendo così una mobilità continua anche in presenza di diversi punti di accesso.

Questo tipo di mobilità non richiede necessariamente che gli **AP** siano connessi tra loro tramite una rete fissa, in quanto gli **AP**, essendo stazionari, possono essere direttamente collegati alla rete locale e offrire servizi di autenticazione e instradamento. Una modalità di connessione supportata è quella in cui un client si connette a due **AP** differenti, consentendo la comunicazione anche tra client connessi a **AP** diversi. Una delle sfide in questo scenario è l'interoperabilità tra standard diversi, come **802.11** per le reti wireless e **802.3** per quelle cablate. Il funzionamento della rete wireless prevede l'uso di **frame di controllo**, alcuni dei quali rientrano nella categoria dei frame di gestione (management). Questi frame contengono informazioni critiche che permettono al client di registrarsi sulla rete e rendere visibile l'esistenza di una sottorete.

Tra questi, il beacon frame rappresenta un messaggio trasmesso periodicamente dall'AP, il quale annuncia la propria presenza e fornisce informazioni sullo stato della rete. Il client, dopo aver ricevuto il beacon, sceglie l'AP a cui connettersi e invia un frame di gestione per richiedere l'apertura della connessione. Una volta stabilita la comunicazione iniziale, si presentano due modalità di funzionamento:

- **Open** (connessione *non autenticata*): il client manda la sua richiesta di connessione. Il client viene identificato dal suo indirizzo MAC.
- **PSK (identificato)**: Questa modalità offre un livello di sicurezza superiore. Il client invia una richiesta di connessione all'AP, che risponde con un challenge (una sorta di sfida). Il client cifra tale challenge utilizzando la password condivisa (pre-shared key) e invia la risposta cifrata. In questo modo, la password non viene mai trasmessa in chiaro e la connessione risulta sicura. Una volta completata la fase di autenticazione e, nel caso di

utilizzo di protocolli di sicurezza avanzata come **WPA2** o **WPA3**, il client può usufruire della rete wireless.

È importante notare che, sebbene i sistemi basati su **WPA2/3** offrano una protezione robusta, esistono attacchi, come quelli di tipo **DDoS**, che possono compromettere l'integrità del servizio. In alcuni casi, la modalità open viene adottata per semplificare l'accesso, specialmente in ambienti dove è difficile condividere una chiave segreta.

## Captive Portal e Gestione degli Accessi

Un esempio comune di implementazione della modalità open è il captive portal. Questa soluzione prevede che, al momento della connessione, il client venga reindirizzato a una pagina web (captive portal) in cui viene presentata una schermata per l'autenticazione o l'accettazione dei termini d'uso. Per differenziare la fase di registrazione iniziale dalla fase di utilizzo della rete, è possibile utilizzare metodi quali il filtraggio per lista di indirizzi MAC o altre tecniche di gestione degli accessi, che semplificano l'utilizzo del servizio e ne aumentano la sicurezza.

## Protocollo DHCP

Il **DHCP** è un protocollo che permette di ottenere automaticamente un indirizzo IP dinamico. Utilizza il protocollo di trasporto **UDP**, perché trasmette messaggi molto piccoli che possono andare persi (se l'indirizzo non arriva, la richiesta viene reiterata dopo un timeout). Il server è in ascolto sulla **porta 67**, **il client usa la porta 68**.

**DHCP** viene spesso detto protocollo **plug-and-play** per la sua capacità di automatizzare la connessione degli host alla rete.

Il protocollo DHCP si articola in quattro punti:

- **Individuazione del server DHCP:** Il primo compito di un host appena collegato è l'identificazione del server **DHCP** con il quale interagire. Questa operazione è svolta utilizzando un messaggio **DHCP discover in broadcast all'indirizzo IP 255.255.255.255** (serve per individuare i server DHCP disponibili), un client manda un pacchetto **UDP** nella **porta 67**. Questo pacchetto viene salvato in un **datagramma IP**.
- **Offerta del server DHCP:** un server **DHCP** che riceve un messaggio di identificazione risponde al client con un messaggio **DHCP offer** che viene inviato in broadcast a tutti i nodi della sottorete
- **Richiesta DHCP:** il client appena collegato sceglierà tra le offerte dei server e risponderà con un messaggio **DHCP request**, che riporta i parametri di configurazione
- **Conferma DHCP:** il server risponde al messaggio di richiesta **DHCP** con un messaggio **DHCP ACK** che conferma i parametri richiesti.