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 ha 5 livelli:

Livello	**Nome**	**Protocollo**
5	**Applicativo**	HTPP (esempio)
4	**Trasporto**	TCP / UDP
3	**Rete**	IPv4 / IPv6
2	**Datalink**	Ethernet
1	**Fisico**	-

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)

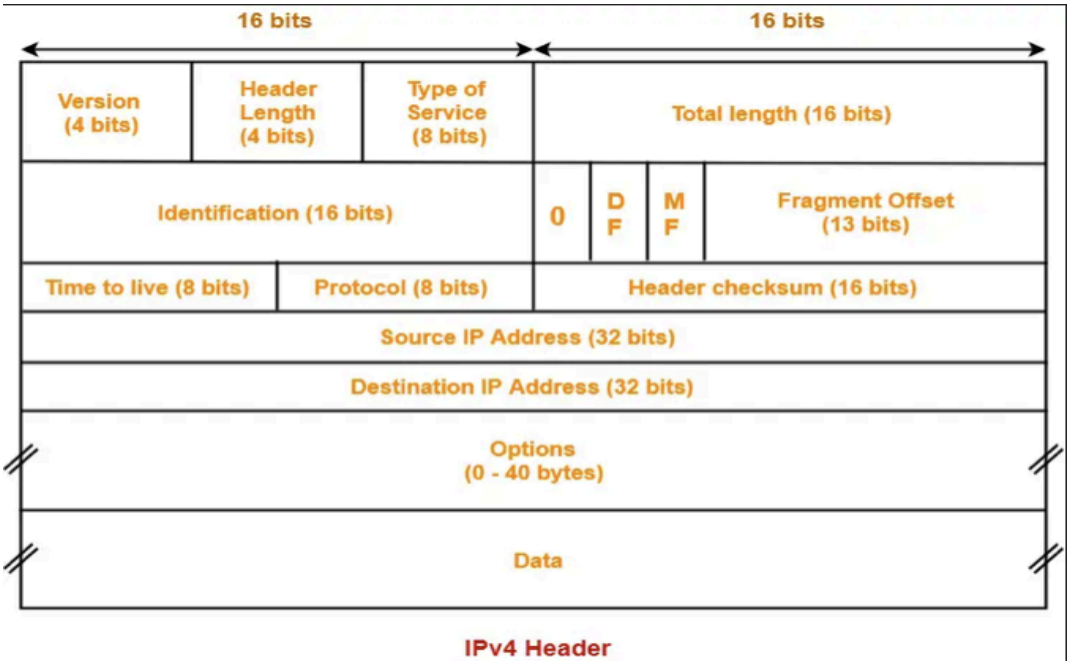
Il livello di applicazione è la sede delle applicazioni di rete e dei relativi protocolli e ha il compito di fornire **interfacce** e strumenti che permettono agli utenti di comunicare attraverso una rete. Si occupa di gestire la comunicazione tra applicazioni software, client di posta elettronica o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



IPv4 Header

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)

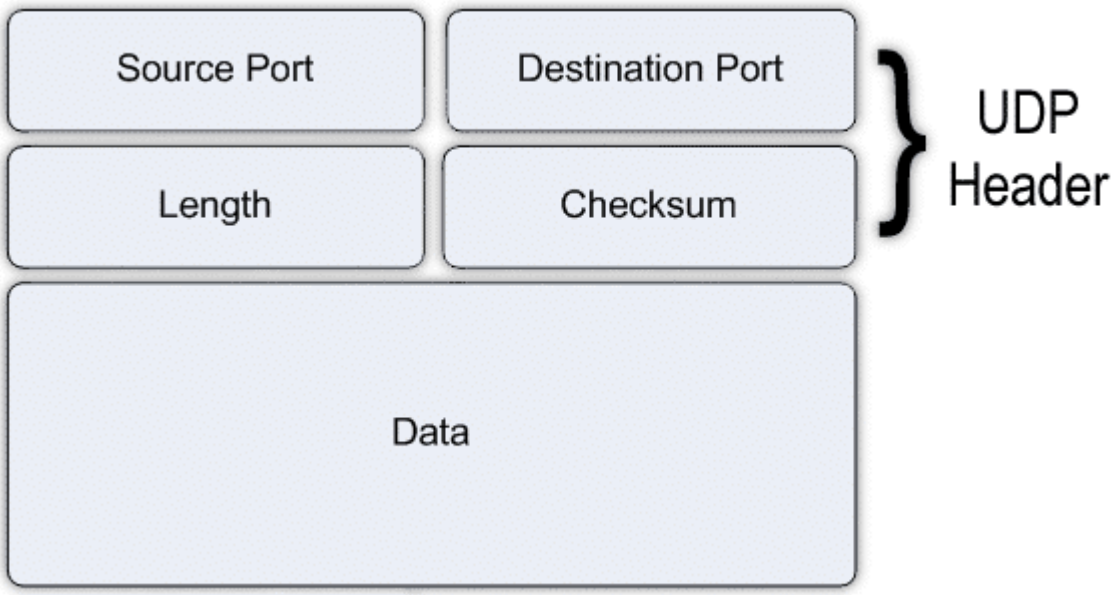
La modalità datagram si basa su un host mittente, un host destinatario e la rete internet, che attraverso dispositivi chiamati **router** definisci il percorso che i messaggi devono percorrere per arrivare a destinazione.
I messaggi in questione vengono chiamati **datagrammi.**
Il protocollo UDP è descritto 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.

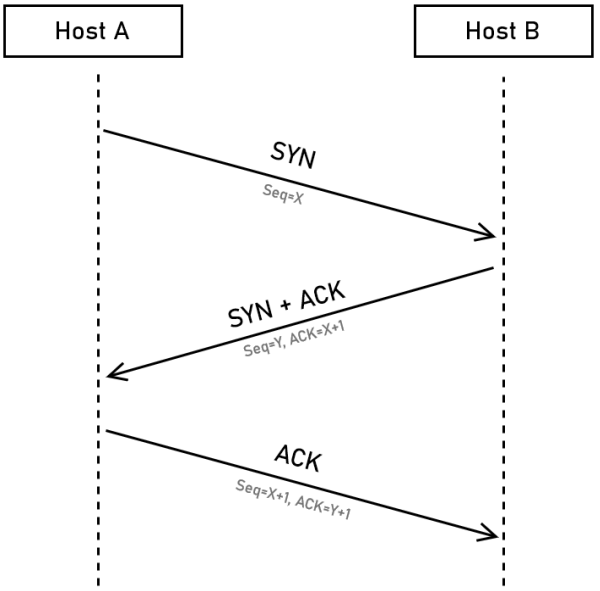
7. **Parte finale:**
   A differenza dei protocolli orientati alla connessione come TCP, non viene inviata alcuna conferma a Host A riguardo alla ricezione del datagramma. Se Host B non riceve il datagramma, non lo comunica a Host A.

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 volte un terzo messaggio per confermare l'apertura della connessione.

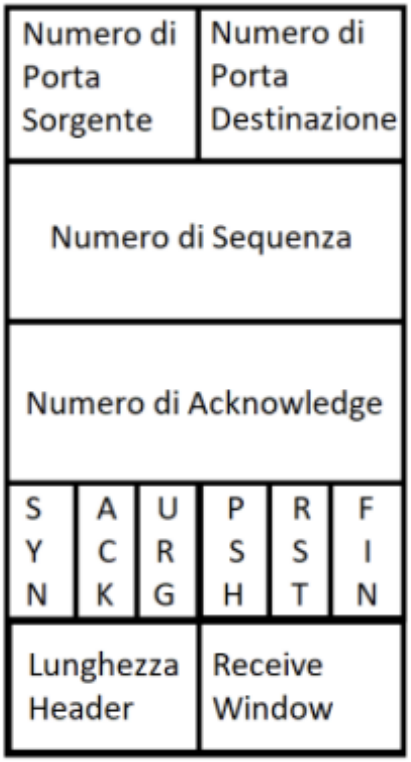


schema di questi 3 passaggi

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 recinzione / receive windows (16 bit)
- Lunghezza dell'intestazione (4 bit)



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 s**ervizio 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 addizionali. C'è la sezione domande, la sezione risposte, la sezione risposte autoritative e la sezione risposte addizionali.

- **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 addizionale:** 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 (Network Time Protocol) 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'epoch 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** (t**0**) 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.

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

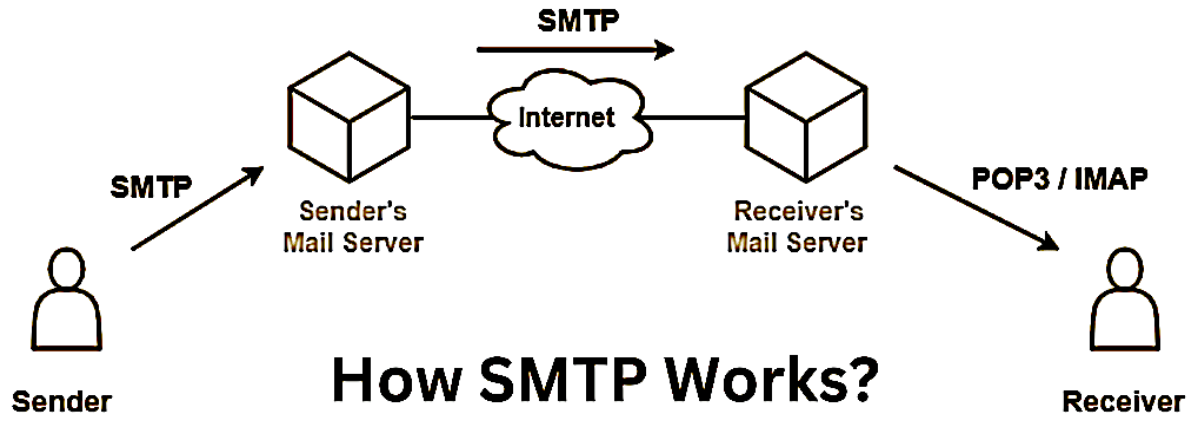
I passaggi che deve svolgere il Server sono:

- 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

Utilizza il protocollo **TCP** sulla **porta 25**.
Il protocollo **SMTP** è uno dei più vecchi usati in rete (prima versione 1981, Rfc 788, ora Rfc 5321).
Asincrono: funziona come servizio postale, il messaggio viene recapitato indipendentemente dalla "prontezza" del destinatario, rimane pronta finché non viene letta.



Per realizzarlo ci sono server intermedi (di posta elettronica), il messaggio viene inviato al server, che lo manda a sua volta ad un altro server.
Qualità servizio: best effort, server fa del suo meglio nel recapitare il messaggio senza dare garanzie, non può sapere se e quando client sarà disponibile a riceverlo
Destinazione dei messaggi: casella postale, file di testo dell'utente destinatario, nel quale viene scritto il contenuto del messaggio ricevuto; il file contiene inoltre tutti i vecchi messaggi. È compito poi del destinatario di accedere alla mailbox, un tempo con il login al server, ora con altri protocolli.
A potrebbe connettersi direttamente al server di posta di **B,** non c'è necessità di **hop** in smtp, ma sono utili. La trasmissione avviene con codifica ascii a 7 bit (non byte da 8 bit); c'è un linguaggio, quindi il client può mandare comandi, codificati con 9 lettere minuscole, il server risponde con numeri codificati in ascii. I messaggi sono divisi in due parti: **intestazione** e **corpo.** Nell'intestazione, le parole chiave danno informazioni (from, to, subject, date, ecc.).

Per dividere l'header dal payload, c'è una riga vuota (**\c\n\c\n, due terminatori di fila**); le righe devono terminare con **\c\n (\c\n = <cr><lf>).**
Ogni riga può essere lunga massimo 75 caratteri, mentre i messaggi possono avere una lunghezza variabile; il messaggio termina con **.\c\n** in una riga (**=<cr><lf>.<cr><lf>**). Per inviarlo c'è una sequenza di **escape.**
Vengono codificati allo stesso modo per tutto il "viaggio". Il **Client** apre la connessione **TCP** con il server su porta 25, prima di mandare comandi aspetta conferma con numero 220 + una parte testuale con indirizzo del server (esempio. <smtp.dibris.unige.it><cr><lf>).
Il primo comando che il client può dare è "helo", con l'indirizzo del client (dibris.unige.it). il server risponde con **250.** Il comando successivo potrebbe essere "**mail from:**" con l'indirizzo del mittente (quindi indirizzo del client + user) (esempio. Malchiodi@dibris.unige.it). Il server risponde di nuovo con **250** (che significa ok), poi "**rcpt to:**" + indirizzo destinatario. Client risponde con **250** e poi con il comando **"data"** il Client richiede al server il permesso di poter mandare il messaggio da inviare. Il Server manda come risposta **354** = "okay, mi aspetto il messaggio". Ricevuto il messaggio, il server risponde con **250,** poi client usa "**quit**" e il server risponde 221 (arrivederci).
Quindi a questo punto il server prova a instaurare una connessione con l'altro server; se fallisce, arriva la notifica al mittente. Solo il proprietario di mailbox può accedervi, deve autenticarsi. Subito l'autenticazione non era richiesta al mittente, ma questo crea problemi di sicurezza: uno può connettersi a server e mandare enorme quantità di messaggi (spam); ora i server non accettano connessioni da client sconosciuti.