UNIVERSITÀ
DEGLI STUDI
DI PADOVA

RETI DI CALCOLATORI

Approccio top-down

(Versione 24/06/2018)

Stesura a cura di:
Stefano Ivancich

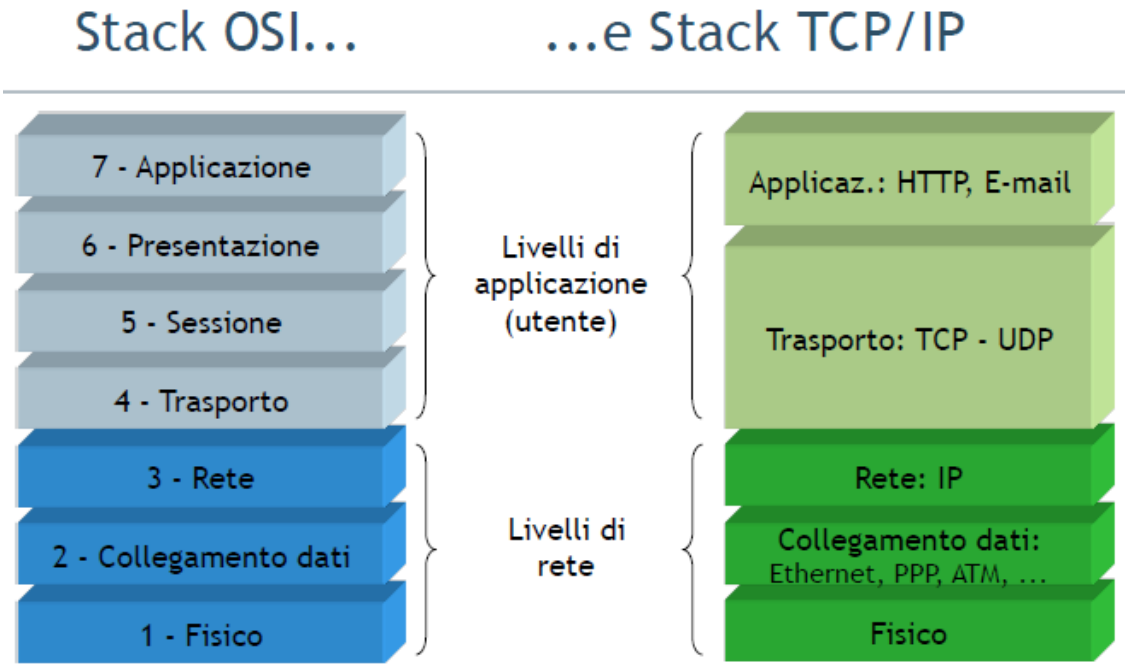
Questa dispensa è scritta da studenti senza alcuna intenzione di sostituire i materiali universitari. Essa costituisce uno strumento utile allo studio della materia ma non garantisce una preparazione altrettanto esaustiva e completa quanto il materiale consigliato dall'Università.
Lo scopo di questo documento è quello di riassumere i concetti fondamentali degli appunti presi durante la lezione, riscritti, corretti e completati facendo riferimento alle slide per poter essere utilizzato come un manuale "pratico e veloce" da consultare. Non sono presenti esempi e spiegazioni dettagliate, per questi si rimanda ai testi citati.

Se trovi errori ti preghiamo di segnalarli qui:
www.stefanoivancich.com ivancich.stefano.1@gmail.com
Il documento verrà aggiornato al più presto.

INDICE

1.MODELLO OSI

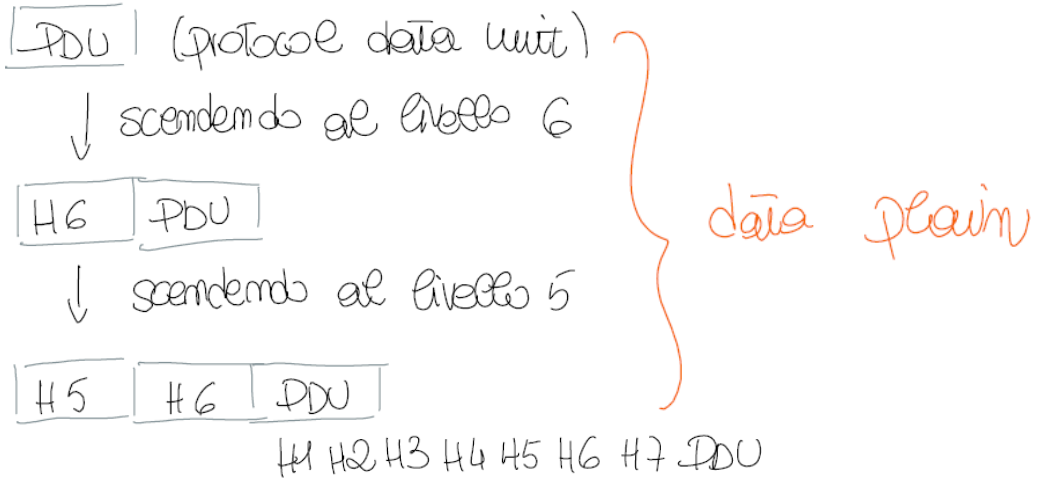
Lo standard delle reti è l'OSI



Physical layer: trasmette i bit in un canale (cavo, aria,)
Data link: trasmissione di un pacchetto, regola l'arbitraggio.
Networking: fa fare la strada giusta nella rete.
…

1.1. Data plain

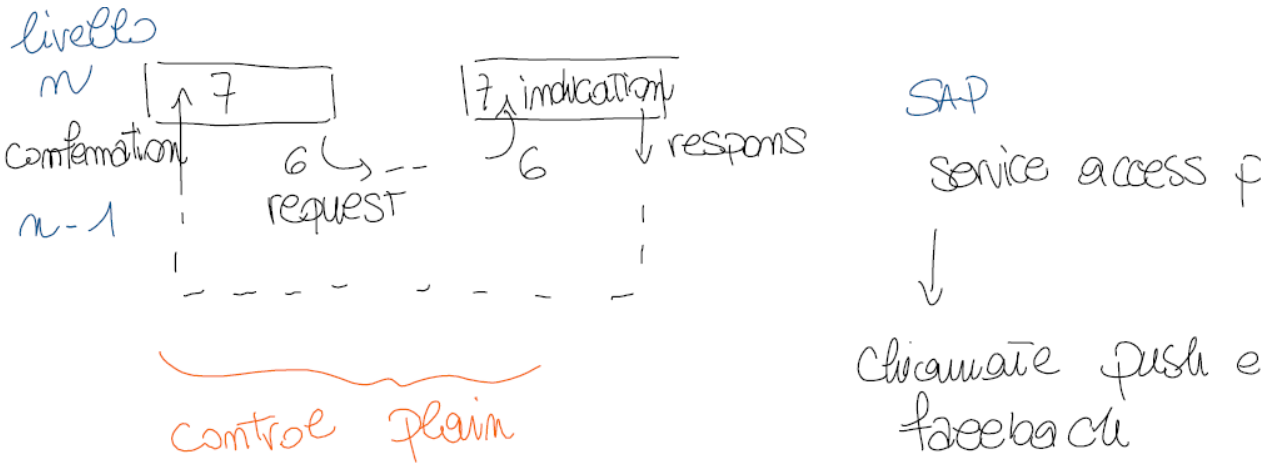
Processo di incapsulamento e decapsulamento.
Due sistemi sono collegati dal Physical layer fisicamente, gli altri livelli interagiscono tramite il livello fisico.
Quindi per trasmettere un dato ad ogni livello si aggiunge ad esso un prefisso (Header) di livello al informazione passata dal livello superiore. Quando l'informazione arriva, ad ogni livello si toglie il prefisso fino a che non si ottiene il dato originale.



Quindi tutti i livelli a parte quello fisico comunicano logicamente.

1.2. Control Plain

Controllo delle chiamate, funzioni e servizi che vengono dati ovvero i Service Acces Point.
Due livelli omologhi: quando uno invia una Request l'altro riceve una Indication che processa ed elabora una Response che causa una Confirmation.



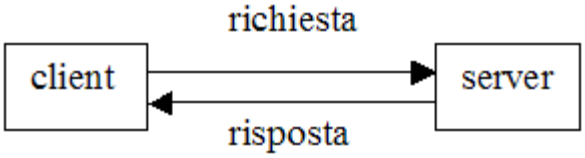
1.3. Management Plain

Attiene a tutto l'insieme di configurazioni, che non fanno parte del normale funzionamento.
Lettura di variabili di sistema che aiutano la gestione dei sistemi.

2. LIVELLI 5-6-7: Applicazione

2.1. Modelli principali

2.1.1. Modello Client-Server

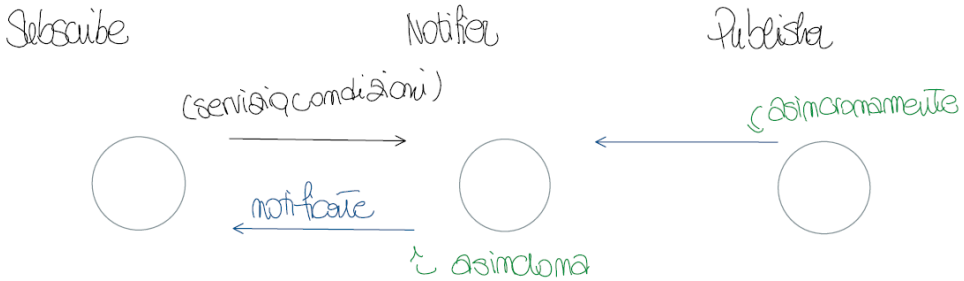


Formato da 2 entità Client e Server, messi di solito su 2 host (macchine) diversi, ognuno ha lo stack di 7 livelli del modello OSI.
Usa in questo ordine le primitive del modello OSI:

- Client invia una richiesta(Request)
- Server invia una Response (quasi immediata)

L'iniziativa di mandare un dato è sempre del Client, il Server aspetta le richieste del Client.
Modello largamente usato. Usato nel WEB.

2.1.2. Modello Publish/Subscribe/Notify



Formato da 3 entità: Subscriber, Publisher, Notifier. (Spesso Publischer e Notifier sono un'unica entità).
Subscribe fa una richiesta e rimane in attesa di una risposta (Notification), non immediata, il tutto avviene in modo asincrono.
Si può emulare il modello S/P/N con il Client-Server facendo il Polling, ovvero chiedere spesso al server finché non risponde, però è un impegno che ha un costo.

2.1.3. Modello peer to peer

Stabilisce dei vincoli su come avviene il reperimento dei dati, che anzi che essere presi da un server unico vengono presi da diversi client che hanno la funzione di server.
Usato per lo scambio di contenuti.

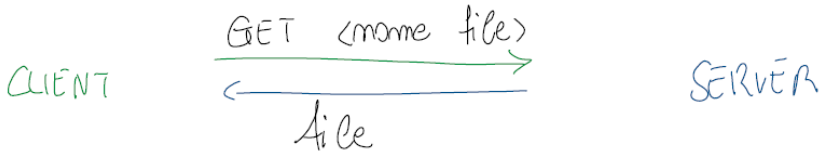
Livello 5-6-7 sono un modulo unico: livello 5 gli header, livello 6 il file, livello 7 lo stream

- Livello 7) Gestione stream
- Livello 5) - Gestione delle sessioni (autenticazione e cookies)
  - Caching
  - Compatibilità (Con che Server sto parlando, con che User-Agent)
  - Referer (Possibilità di un server di conoscere il contenuto da cui è stato ricavato l'accesso)
- Livello 6) Tipo file

2.2. Protocollo HTTP 1.0 (RFC 1945)

0.9-->1.0-->1.1-->2.0

HTTP 0.9: il Client richiede un file e il Server glie lo invia. Non erano considerati i livelli 6 e 5.



URL (Uniform Resource Location): in che macchina e in che cartella si trova il file.
Il livello 4 permette di scambiare flussi di byte in maniera simile alla scrittura di file.

HTTP 1.0: Descritto nel documento RFC1945.
Ogni richiesta è stateless ovvero nasce e muore senza lasciare traccia.
Client: programma che stabilisce le connessioni allo scopo di inviare richieste.
Server: applicazione che accetta connessioni per servire le richieste inviando indietro risposte.

Grammatica BNF (insieme di regole)

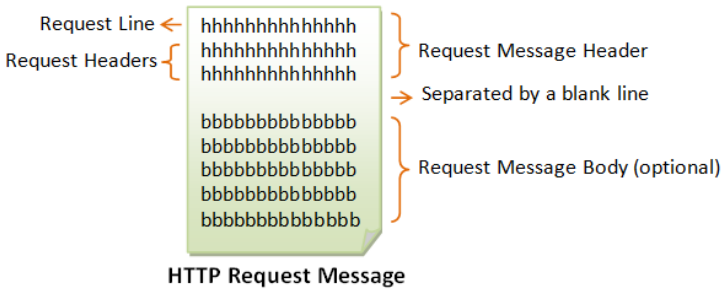
- `name = definition`
- `"literal"`: Quotation marks surround literal text. The text is case-insensitive.
- `rule1 | rule2`: Elements separated by a bar ("I") are alternatives
- `(rule1 rule2)`: Elements enclosed in parentheses are treated as a single element.
- `*rule`: The character "*" preceding an element indicates repetition. Full form is "`<n>*<m>element`" indicating at least <n> and at most <m> occurrences of element.
- `[rule]`: Square brackets enclose optional elements.
- `N rule`: Specific repetition: "<n>(element)" is equivalent to "<n>*<n>(element)" ; that is, exactly <n> occurrences of (element).
- `#rule`: defining lists of elements. `<n>#<m>element` indicating at least <n> and at most <m> elements, each separated by one or more commas (" , ")
- `;` comment: A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of line.

Messaggi HTTP: consistono in richieste dal client al server e risposte dal server al client.

```
HTTP-message   = Simple-Request           ; HTTP/0.9 messages
               | Simple-Response
               | Full-Request             ; HTTP/1.0 messages
               | Full-Response
```

Simple request può rispondere con niente o con un file, ma non si sa il tipo del file.

2.2.1. Richiesta HTTP




```
Full-Request   = Request-Line
                 *( General-Header
                  | Request-Header
                  | Entity-Header )
                 CRLF
                 [ Entity-Body ]
```

Entity: oggetto che viene trasportato dalla richiesta

Request-Line: `Request-Line = Method SP Request-URI SP HTTP-Version CRLF`

```
Method          = "GET"
                | "HEAD"
                | "POST"
                | extension-method
```

- `HEAD`: come GET solo che il server non dovrà restituire l'Entity-Body
- `POST`: per inviare un contenuto (file)???
- `Request-URI`: vedremo più avanti, intanto è un nome di un file"/blabla/…"
- HTTP-Version: "HTTP" "/" 1*DIGIT "." 1*DIGIT (Noi mettiamo `HTTP/1.0` o `HTTP/1.1`)

Headers: `request-header-name: request-header-value1, value2, ... CRLF`

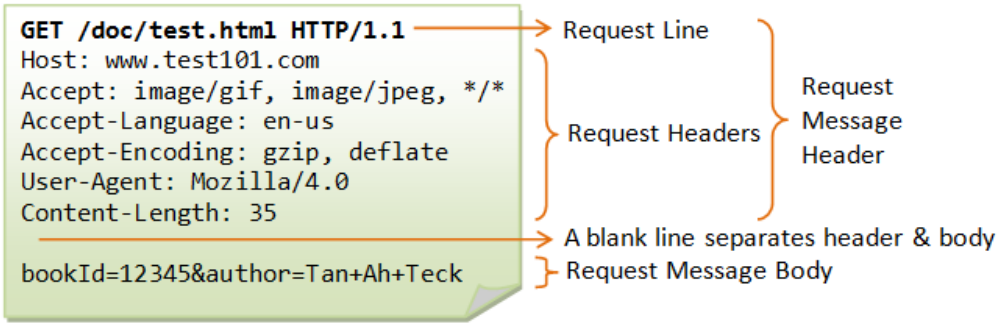
- `Host: www.xyz.com`
- `Referer: http://`… dice al server l'URL precedente

```
Request-Header = Authorization
               | From
               | If-Modified-Since
               | Referer
               | User-Agent
```

- `If-Modified-Since: HTTP-date` usata con GET, se la risorsa richiesta non è stata modificata verrà restituito dal server un codice 304 senza body.
- `Last-Modified: HTTP-date` indica la data che il sender pensa sia dell'ultima modifica al file.
- `Expires: HTTP-date` indica che dopo la data il contenuto del body è considerato vecchio.
- `User-Agent`: indica il browser

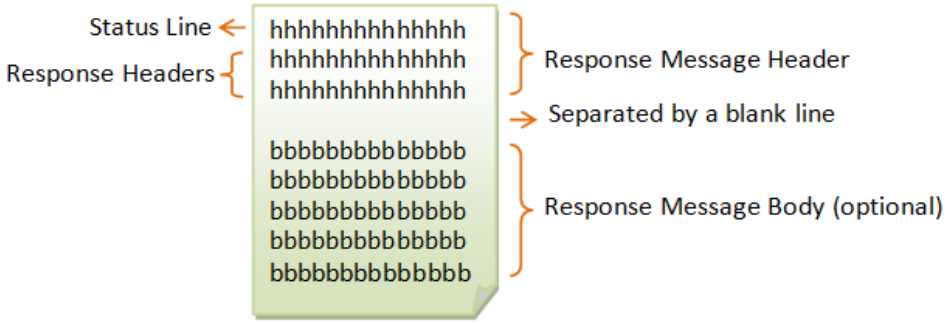
Gli header vengono letti fino a 2 `CRLF` poi c'è il body.

Esempio: `GET /test.html HTTP/1.0\r\nHost:www.google.co.uk\r\n\r\n`



2.2.2. Risposta HTTP

```
Full-Response   = Status-Line
                  *( General-Header
                   | Response-Header
                   | Entity-Header )
                  CRLF
                  [ Entity-Body ]
```



HTTP Response Message

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

- **Status-Code:** codice a 3 cifre, la 1 cifra indica la categoria,

```
1xx: Informational - Not used, but reserved for future use

2xx: Success - The action was successfully received,
     understood, and accepted.

3xx: Redirection - Further action must be taken in order to
     complete the request

4xx: Client Error - The request contains bad syntax or cannot
     be fulfilled

5xx: Server Error - The server failed to fulfill an apparently
     valid request
 Status-Code     = "200"   ; OK
                 | "201"   ; Created
                 | "202"   ; Accepted
                 | "204"   ; No Content
                 | "301"   ; Moved Permanently
                 | "302"   ; Moved Temporarily
                 | "304"   ; Not Modified
                 | "400"   ; Bad Request
                 | "401"   ; Unauthorized
                 | "403"   ; Forbidden
                 | "404"   ; Not Found
                 | "500"   ; Internal Server Error
                 | "501"   ; Not Implemented
                 | "502"   ; Bad Gateway
                 | "503"   ; Service Unavailable
                 | extension-code
```

- **Reason-Phrase:** stringa che descrive l'errore. `*<TEXT, excluding CR, LF>`
- Esempio: `HTTP/1.0 404 Not Found`

Headers: `response-header-name: response-header-value, value2,...`

Es: `Keep-Alive: timeout=15, max=100`

In http1.1 c'è un header Content-Length:269 che dice la dimensione del body.

Il client deve essere in grado di leggere la risposta.

Parsing: trovo il carattere di inizio riga e sostituisco i caratteri separatori ":" con caratere terminatore, così ho nome(puntatore)+terminatore+valore(puntatore).

Transcodifica chunked: (HTTP1.1) permette di non sapere la grandezza del body a priori, quindi lo manda in pezzi con piccoli header che ne dicono la dimensione, l'ultimo pezzo ha dimensione 0.

whatis request

GET è chiamare un programma che produce un output

CGI (Common gateway Interface): i server diventano praticamente terminali. è poco sicuro, in realtà si usano interpreti con forti limitazioni.

URI indica un comando che si vuole eseguire sul server e si vuole che l'output venga mandato al client. (A volte è il nome di un file).
URL è un caso particolare di URI.

```
URI           = ( absoluteURI | relativeURI ) [ "#" fragment ]

absoluteURI   = scheme ":" *( uchar | reserved )

relativeURI   = net_path | abs_path | rel_path

net_path      = "//" net_loc [ abs_path ]
abs_path      = "/" rel_path
rel_path      = [ path ] [ ";" params ] [ "?" query ]

path          = fsegment *( "/" segment )
fsegment      = 1*pchar
segment       = *pchar

params        = param *( ";" param )
param         = *( pchar | "/" )

scheme        = 1*( ALPHA | DIGIT | "+" | "-" | "." )
net_loc       = *( pchar | ";" | "?" )
query         = *( uchar | reserved )
fragment      = *( uchar | reserved )

pchar         = uchar | ":" | "@" | "&" | "=" | "+"
uchar         = unreserved | escape
unreserved    = ALPHA | DIGIT | safe | extra | national

escape        = "%" HEX HEX
reserved      = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+"
extra         = "!" | "*" | "'" | "(" | ")" | ","
safe          = "$" | "-" | "_" | "."
unsafe        = CTL | SP | <"> | "#" | "%" | "<" | ">"
national      = <any OCTET excluding ALPHA, DIGIT,
```

Autenticazione
Client invia `id:password` in base64:

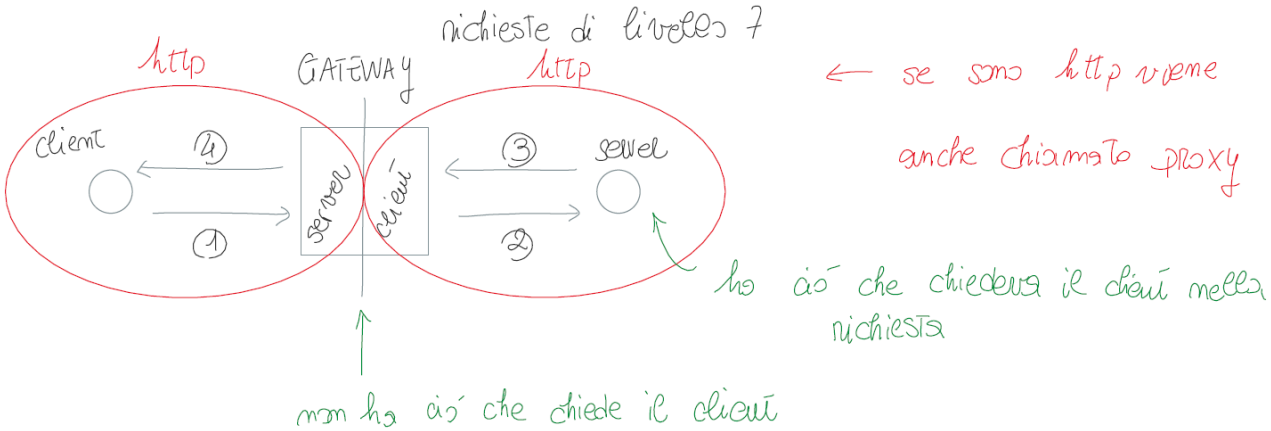
Basic SP id:password

Es: header: `Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`

Il server invia: `WWW-Authenticate: Basic realm="WallyWorld"`

2.3. Gateway di livello 7

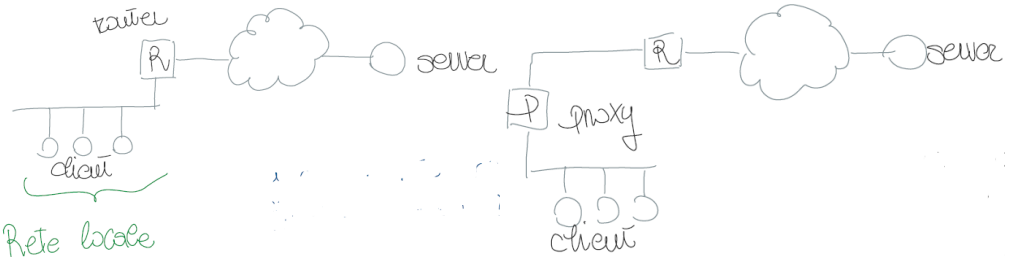
Riceve e inoltra richieste di livello 7



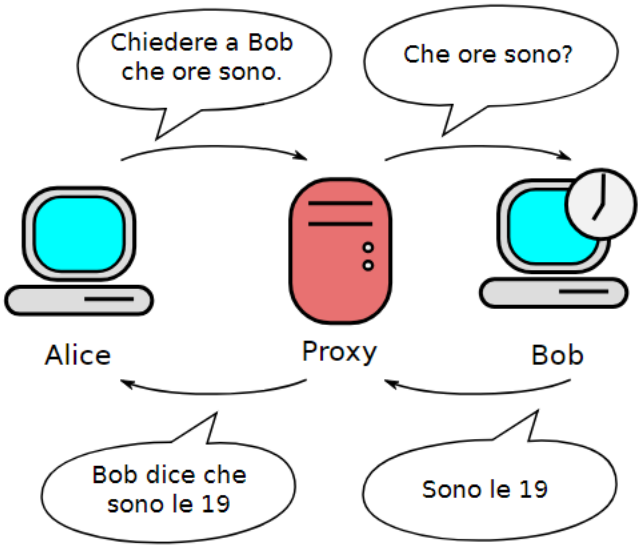
Fa da intermediario o da traduttore del protocollo.
(1) non è la classica richiesta, bisogna dire al gateway quale server vogliamo contattare.
Usare exec nella GET è un gateway http->ssh.

Il client invia Request Line: `Connect www.google.it` (è un GET che richiede l'encription, noi ignoriamo e trasformiamo in una GET, ma è un path completo e non relativo)
Il gateway è utile per:

- Operazioni di caching collettivo: se tanti client devono accedere alla stessa risorsa, invece di fare tante richieste ne viene fatta una sola e la risorsa messa in cache



- Vede le richieste e può filtrare (può non accettare determinati contenuti)
- Sicurezza: lasciare i client in rete privata senza connessione a internet.



3.LIVELLO 3: Rete

Livello 2 e 3: si occupano del trasporto di pacchetti di dati.
Switch: è un dispositivo di rete che si occupa di commutazione a livello datalink

Commutazione di pacchetto (packet switching)

I nodi lavorano come centri di smistamento:

- Instradamento (**routing**): scegliere la prossima destinazione, i nodi vengono chiamati router.
- Inoltro (**forwarding**)

I nodi hanno tanti link di ingresso e di uscita, Il pacchetto ha un indirizzo di destinazione, il router decide a quale nodo mandare il pacchetto.
Il pacchetto ha una lunghezza massima.
Multiplexing statistico.
Nei livelli 2 e 3 non esistono gli stream e le connessioni, vengono simulate con un servizio datagram.
Internetworking: pacchetto virtuale che può viaggiare su qualsiasi rete o tecnologia.

3.1. Interet Protocol (RFC 791)

Unico protocollo di livello 3 ormai esistente. Pacchetto in grado di viaggiare tra reti eterogenee.
Il router scarta pacchetti quando trova errori o buffer overflow (arrivano più pacchetti di quanti possano essere processati).
IP è tra il livello 3 e 4. Usa le reti a pacchetto. Il livello 4 chiama IP che si appoggia su 3, i router lavorano solo fino al livello 3.

L'obiettivo è di muovere datagrammi in un insieme interconnesso di reti. Questo lo si fa spostando i datagrammi da un modulo all'altro (router) finché non arriva a destinazione.
Il messaggio può passare per reti in cui la dimensione massima di pacchetto è inferiore della dimensione del datagram, per ovviare a ciò si usa la frammentazione.

Implementa 2 funzioni:

- **Indirizzamento**: un name indica cosa cerchiamo, l'address indica dov'è, la route indica come arrivarci. IP lavora con gli indirizzi.
  Indirizzo è a 32 bit, diviso in 2 parti: rete e host (indirizzo locale), i primi bit dicono dove avviene la suddivisione.

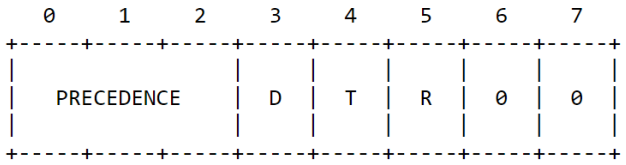
High Order Bits	Format	Class
0	7 bits of net, 24 bits of host	a
10	14 bits of net, 16 bits of host	b
110	21 bits of net, 8 bits of host	c
111	escape to extended addressing mode	

- **Frammentazione**: un datagram può essere marcano come "non frammantare", in tal caso se è troppo grande viene scartato.

+	Bits 0–3	4–7	8–15	16–18	19–31
0	Version	Internet Header length	Type of Service (adesso DiffServ e ECN)	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options (facoltativo)				
160 o 192+	Data				

Versione [4 bit] - Indica la versione del pacchetto IP: per IPv4, ha valore 4 (da qui il nome IPv4);
Internet Header Length (IHL) [4 bit] - Indica la lunghezza (in word da 32 bit) dell'header del pacchetto IP ovvero da dove parte il campo dati; tale lunghezza può variare da 5 word (20 byte) a 15 word (60 byte) a seconda della presenza e della lunghezza del campo facoltativo Options;
Type of Service (TOS) [8 bit] -

- bit 0-2: Precedenza
- bit 3: Latenza (0 = normale, 1 = bassa) (Banda)
- bit 4: Throughput (0 = normale, 1 = alto) (Quanti pacchetti far passare)
- bit 5: Affidabilità (0 = normale, 1 = alta) (Non buttar via pacchetti)
- bit 6-7: Riservati per usi futuri



Total Length [16 bit] - Indica la dimensione (in byte) dell'intero pacchetto, comprendendo header e dati. minimo di 20 byte (header minimo e campo dati vuoto) ad un massimo di 65535 byte.

Identification [16 bit] - identifica il frammento, per quando vanno riassemblati. Il riassemblaggio viene fatto dall'host, non dalla rete. È un campo inutile se non viene fatta la frammentazione ma bisogna metterlo lo stesso.
Flags [3 bit] -

- Reserved - sempre settato a 0
- DF (Don't Fragment) - se settato a 1 indica che il pacchetto non deve essere frammentato.
- MF (More Fragments) - se settato a 0 indica che il pacchetto è l'ultimo frammento.

Fragment Offset [13 bit] - Indica l'offset (misurato in blocchi di 8 byte) di un particolare frammento relativamente all'inizio del pacchetto IP originale: il primo frammento ha offset 0.

Time to live (TTL) [8 bit] - Indica il tempo di vita (time to live) del pacchetto.
Protocol [8 bit] - Indica il codice associato al protocollo utilizzato nel campo dati: TCP=6, ICMP=1.
Header Checksum [16 bit] - È un campo usato per il controllo degli errori dell'header, NON sul contenuto.
Source address [32 bit] - Indica l'indirizzo IP associato all'host del mittente del pacchetto.
Destination address [32 bit] - Indica l'indirizzo IP associato all'host del destinatario.

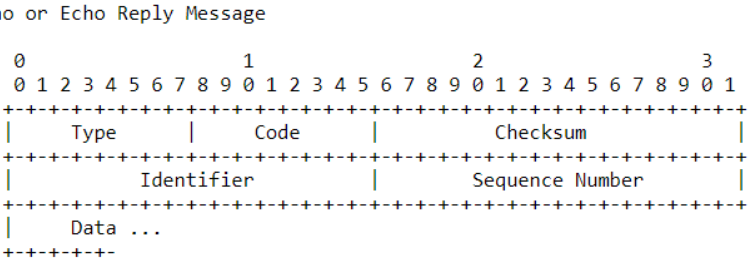
3.2. ICMP (RFC 792)

Serve a controllare che fin al livello 3 funzioni tutto.
È un protocollo di servizio per reti a pacchetto che si occupa di trasmettere messaggi riguardanti malfunzionamenti.
ICMP è **incapsulato direttamente in IP**, aggiunti **8 byte all'inizio** dei payload del IP, (è un protocollo di livello 3 dello stack TCP/IP) e non è quindi garantita la consegna a destinazione di esso.
Viene utilizzato da molti applicativi di rete, tra cui **ping e traceroute**.
Va implementato in ogni modulo IP.
Non sono inviati messaggi ICMP riguardo se stessi.
Mandati solo per gestire il Fragment di offset 0.
Tipi di messaggi: (Tipo/nome)

- `0 Echo Reply`
- `3 Destination Unreachable`
- `4 Source Quench`
- `5 Redirect`
- `8 Echo`
- `11 Time Exceeded`
- `12 Parameter Problem`
- `13 Timestamp`
- `14 Timestamp Reply`
- `15 Information Request`
- `16 Information Reply`

`route -n` per sapere il mio gateway

Echo or Echo Reply Message



3.3. Routing (Instradamento)

La funzione principale di un router è di trasmettere un pacchetto verso la sua rete di destinazione (l'indirizzo IP di destinazione del pacchetto). Per fare questo, un router deve cercare le informazioni di instradamento memorizzato nella tabella di routing.

Rete connessa direttamente: è una rete che è collegata direttamente ad una delle interfacce del router. Quando una interfaccia del router è configurata con l'indirizzo di rete (indirizzo IP e subnet mask), l'interfaccia diventa un host collegato su quella rete. L'indirizzo di rete dell'interfaccia è inserito nella tabella di routing come una rete connessa direttamente. Quando un router inoltra un pacchetto a un host, tale host è sulla stessa rete del router (rete connessa direttamente).
Rete remota: è una rete che non è collegato direttamente al router. In altre parole, una rete può essere raggiunta solo inviando il pacchetto a un altro router.

Ogni volta che un nodo di una rete deve inviare dati ad un altro nodo, deve prima sapere dove inviarli. Se il nodo di invio non è connesso direttamente al nodo di destinazione, invierà il dato al gateway della rete di appartenenza, che poi stabilisce come indirizzare il "pacchetto" dati alla destinazione corretta. Ogni gateway ha quindi bisogno di tenere traccia delle rotte necessarie all'instradamento dei pacchetti di dati e, per questo motivo, si utilizza una **tabella di routing**.
Tabella di routing: è un database, memorizzato in un router, che elenca le rotte di destinazione di una data rete e in molti casi una metrica di tale rotta. Contiene anche informazioni sulla topologia della rete immediatamente circostante.
Contenuto:

- l'ID di rete: cioè la sotto-rete di destinazione
- Costo/metrica: costo di una connessione
- salto successivo: è l'indirizzo della stazione successiva (gateway) in cui il pacchetto deve essere inviato sulla strada verso la sua destinazione finale.

Protocollo di routing: permette ai router di scambiarsi informazioni tra loro al fine di costruire delle tabelle di routing che permettano così il corretto instradamento dei pacchetti verso la giusta destinazione.
Il ricorso ai protocolli di routing per la costruzione automatica e dinamica delle tabelle di routing diventa necessario quando il numero di sottoreti interconnesse è elevato (come nel caso della rete Internet).
Algoritmi per il Routing Statico (non adattivi): Dijkstra e flooding.

3.3.1. Algoritmi per il Routing Dinamico (adattivi)

Routing Gerarchico: in Internet, visto il gran numero di reti interconnesse, dal punto di vista dell'instradamento è comodo intendere la rete come un insieme di sistemi autonomi (AS) ognuno dei quali si occupa di gestire autonomamente e uniformemente il routing interno alla propria interrete con un medesimo protocollo di routing e l'interconnessione solo con gli altri AS direttamente connessi.

Protocolli sono diversi a seconda che si tratti di router all'interno di uno stesso AS (interior gateway protocol, IGP), oppure di router che collegano tra loro più AS (EGP), Exterior Gateway Protocol.

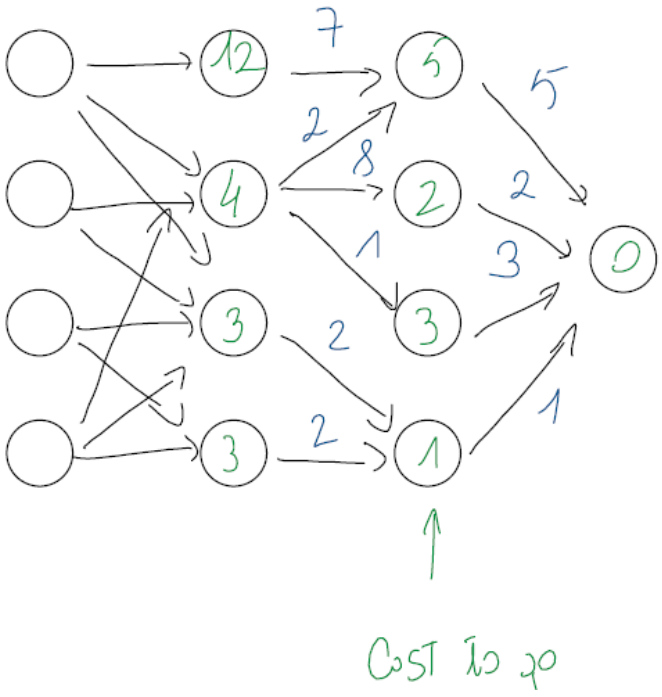
Protocolli interni al sistema autonomo (IGP):

- **Protocolli distance vector** (Dijkstra distribuito): ricevono e mandano informazioni riguardo ai collegamenti solamente ai router adiacenti, ovvero ogni nodo manda ai suoi adiacenti il proprio cost to go, e il nodo sceglie il più piccolo su cui instradare l'informazione. RIP (Routing Information Protocol) e IGRP (Interior Gateway Routing Protocol).
- **Protocolli link state** (Dijkstra centralizzato): mandano informazioni a tutti i router del proprio sistema autonomo. Ogni nodo dice ad i suoi adiacenti quali sono i suoi adiacenti, l'informazione si propaga in modo che ogni nodo può ricostruire il grafo della rete. Viene ricostruito ogni 30 sec.
  IS-IS (Intermediate System to Intermediate System) e OSPF (Open Shortest Path First).
- **Protocolli ibridi:** EIGRP (Enhanced Interior Gateway Routing Protocol).

Protocollo esterno al sistema autonomo (EGP): BGP (Border Gateway Protocol).

Strada migliore per arrivare a destinazione:

- Metrica: costo di una connessione
- Cost to go: minimo delle somme dei costi a ritroso
- Si usa l'algoritmo Dijkstra con Matrice delle adiacenze: che dice chi è connesso a chi con che pesi. (Algoritmo Lez29)



3.4. Funzioni aggiuntive: NAT, Firewalling e Tunneling

NAT (Network address translation): è una tecnica che consiste nel modificare gli indirizzi IP contenuti negli header dei pacchetti in transito su un sistema che agisce da router all'interno di una comunicazione tra due o più host.
Si può distinguere tra source NAT (SNAT) e destination NAT (DNAT), a seconda che venga modificato l'indirizzo sorgente o l'indirizzo destinazione del pacchetto che inizia una nuova connessione.
Nel **source NAT**, le connessioni effettuate da uno o più computer vengono alterate in modo da presentare verso l'esterno uno o più indirizzi IP diversi da quelli originali. Quindi chi riceve le connessioni le vede provenire da un indirizzo diverso da quello utilizzato da chi effettivamente le genera. NAPT (Network Address and Port Translation), vengono modificati non solo gli indirizzi IP ma anche le porte TCP e UDP delle connessioni in transito.
Nel **destination NAT**, le connessioni effettuate da uno o più computer vengono alterate in modo da venire redirette verso indirizzi IP diversi da quelli originali. Quindi chi effettua le connessioni si collega in realtà ad un indirizzo diverso da quello che seleziona.
Il router può: mappare ad ogni indirizzo privato un indirizzo pubblico (NAT Puro) o guardare il contenuto dei pacchetti per creare una mappa dinamica.
Meccanismo NAT ha salvato lo spazio degli indirizzi. Non è una funzione di basa ma addizionale al livello 3.

Firewalling: router analizza il pacchetto IP che gli arriva confrontandolo con delle regole, e decide se instradarlo o eliminarlo (drop). Possiede anche uno stato delle connessioni attive

Tunneling: trasportare die pacchetti IP dentro pacchetti IP

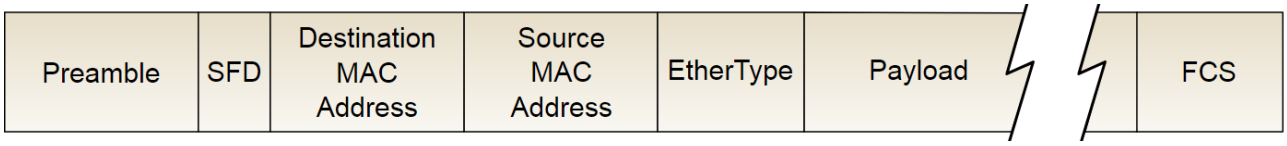
4. LIVELLO 2: Collegamento dati

4.1. Protocollo Ethernet

Protocollo di Livello 2, il più utilizzato. Viaggia su reti locali, router connessi punto-punto.
LAN: Collegare calcolatori in una stanza o edificio.
Politica di arbitraggio CSMA/CD, carrier sense: sente se qualcuno sta trasmettendo, quindi aspetta che gli altri finiscano, tempo massimo per comunicare: lunghezza pacchetto/banda.
Se mentre sto parlando c'è un altro che inizia a parlare devo invalidare quello che ho trasmesso.
Tempo minimo di trasmissione di un pacchetto deve essere 2 volte il tempo di propagazione di un segnale.
Lunghezza pacchetto: L>2Bd/v (B: Banda, d: distanza, v: velocità propagazione)
Esponential backoff: i 2 host che collidono estraggono un numero a caso, 0 o 1. Se collidono ancora, l'intervallo diventa 0-2, e via dicendo raddoppiando fino a 16. Poi viene considerato errore.
Cablaggio strutturato: per evitare problemi di scollegamento, il bus viene messo in un HUB che è un dispositivo che inoltra i dati in arrivo da una qualsiasi delle sue porte su tutte le altre, cioè in maniera diffusiva (broadcasting). Per questa ragione può essere definito anche come ripetitore multiporta.
Se l'HUB ha un microprocessore è uno **switch di livello 2**, ovvero riceve in ingresso i singoli bit dei singoli host, invia i dati solo alle porte che lo meritano.

Siccome in una rete LAN tutti possono vedere quello che passa, allora un host può trasmettere un pacchetto ad un altro host tramite un **indirizzo MAC** (di rete locale, non è un indirizzo internet).
Cioè tutti vedono passare il pacchetto ma solo chi ha l'indirizzo giusto lo può leggere.

Lo switch memorizza una tabella di indirizzi ethernet degli host collegati alle sue porte. Ad una porta possono essere collegati altri switch, quindi la tabella può diventare onerosa.
Pacchetti broadcast: pacchetti destinati a tutti (indirizzo tutti bit=1).
Banda aggregata: commutare n volte la banda dei singoli n canali che usano il massimo della banda sul loro canale.
Insieme di broadcast: insieme di nodi raggiungibili dal broadcast, collegati al canale
Pacchetto ethernet:



Non nel buffer (solo per sincronizzare):

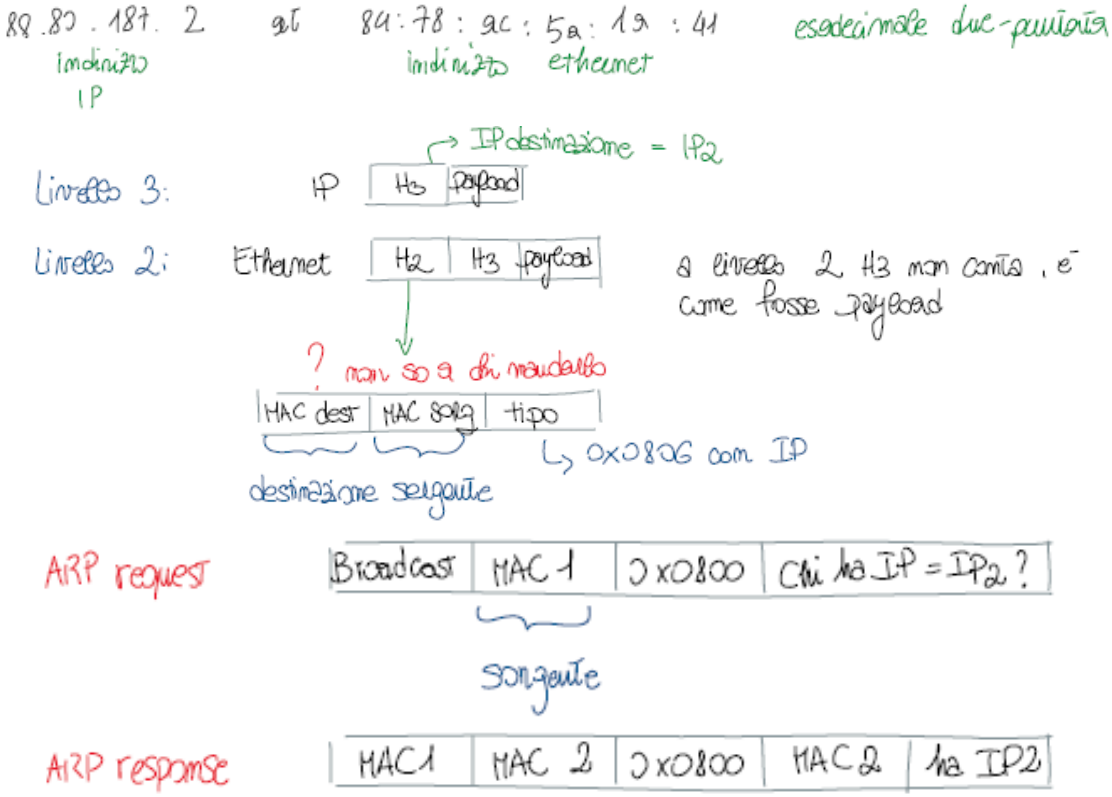
- **Preambolo:** 56-bit (7-byte) di 1 e 0 alternati per la sincronizzazione a livello di bit tra i device.
- **SFD:** 8-bit (1-byte) per la sincronizzazione a livello di byte. Finisce con 11 che indica l'inizio del pacchetto ethernet.

Pacchetto vero e proprio:

- **MAC address destinazione:** 48-bit (6-byte)
  6 byte di indirizzo ethernet perché la scheda deve essere unica al mondo
- **MAC address sorgente:** 48-bit (6-byte)
- **Ethertype:** 16-bit (2-byte): identifica il protocollo di livello superiore incapsulato nel campo dati. 0x0800=IP, 0x0806=ARP
- **Payload:** da 46 a 1500 byte
- **CRC:** 32-bit (4-byte)

4.2. Protocollo ARP (RFC 826)

Ricava da un indirizzo IP il corrispondente indirizzo Ethernet. Fa un broadcast chiedendo a tutti chi ha l'indirizzo IP, uno solo risponde dandogli l'indirizzo ethernet, e lo mette in cache. Cache visualizzabile con: `arp -a`



Ogni nodo prende una piccola decisione di routing, sa se il destinatario è nella sua rete o altrove. Lo fa chiedendosi `net(mioIP)==net(IPdestinatario)`?

4.2.1. Pacchetto ARP

Internet Protocol (IPv4) over Ethernet ARP packet		
octet offset	0	1
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	
8	Sender hardware address (SHA) (first 2 bytes)	
10	(next 2 bytes)	
12	(last 2 bytes)	
14	Sender protocol address (SPA) (first 2 bytes)	
16	(last 2 bytes)	
18	Target hardware address (THA) (first 2 bytes)	
20	(next 2 bytes)	
22	(last 2 bytes)	
24	Target protocol address (TPA) (first 2 bytes)	
26	(last 2 bytes)	

Hardware type (HTYPE): tipo protocollo di rete. Ethernet =1
Protocol type (PTYPE): nternetwork protocol. IPv4=0x0800
Hardware length (HLEN): lunghezza (in octets) degli indirizzi hardware. Ethernet=6.
Protocol length (PLEN): lunghezza (in octets) degli indirizzi usai dal protocollo superiore. IPv4=4.
Operation: Specifica l'operazione che sta eseguendo il mittente: 1 per richiesta, 2 per risposta.
Sender hardware address (SHA): 6-byte

- In una richiesta ARP questo campo viene utilizzato per indicare l'indirizzo dell'host che invia la richiesta.
- In una risposta ARP questo campo viene utilizzato per indicare l'indirizzo dell'host che la richiesta cercava.

Sender protocol address (SPA): Indirizzo di rete (IP) del mittente.
Target hardware address (THA):

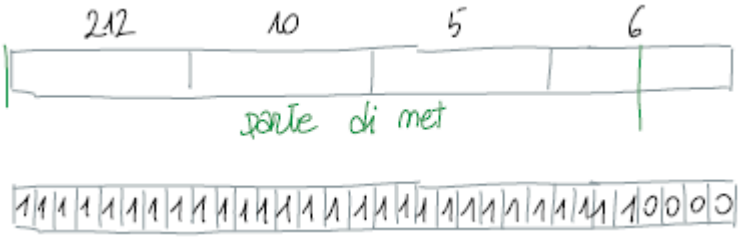
- In una richiesta ARP questo campo viene ignorato.
- In una risposta ARP questo campo viene utilizzato per indicare l'indirizzo dell'host che ha originato la richiesta ARP.

Target protocol address (TPA): Indirizzo di rete (IP) del destinatario previsto.

4.2.2. SubnetMask

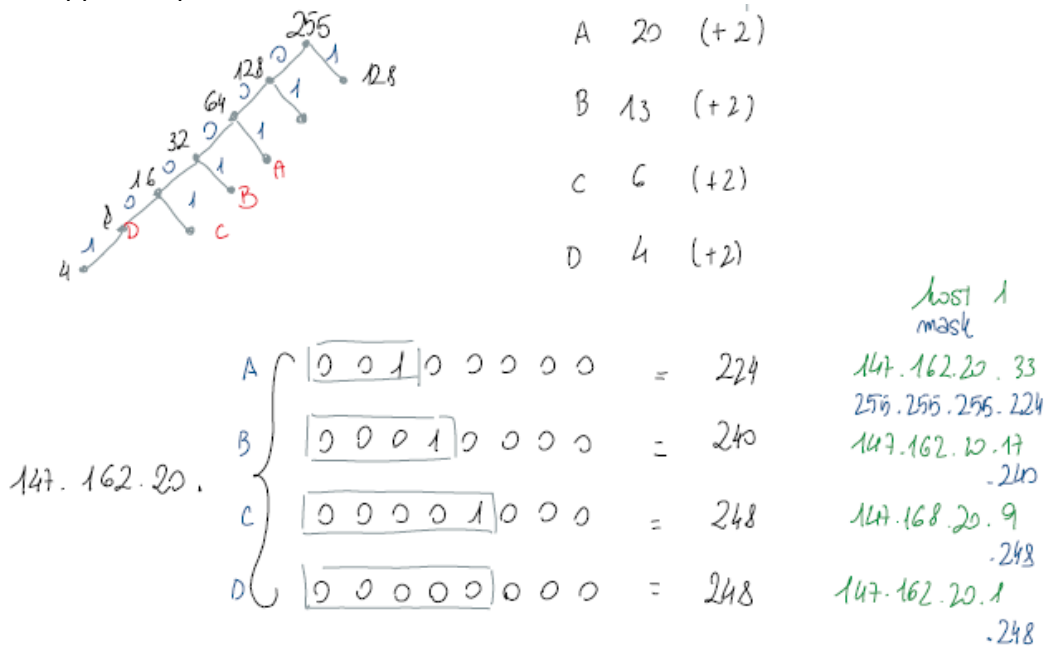
Dato di 4 byte che dice quali bit dell'IP sono la parte di rete. Si mettono degli 1 in corrispondenza dei bit dell'IP che devono essere visti come parte di rete.

Es: 28bit di NET e 4 bit di HOST.



Quindi `if(net(IP1)==net(IP2))` diventa `if(IP1&mask==IP2&mask)`

Può essere riapplicato più volte nelle sottoreti:

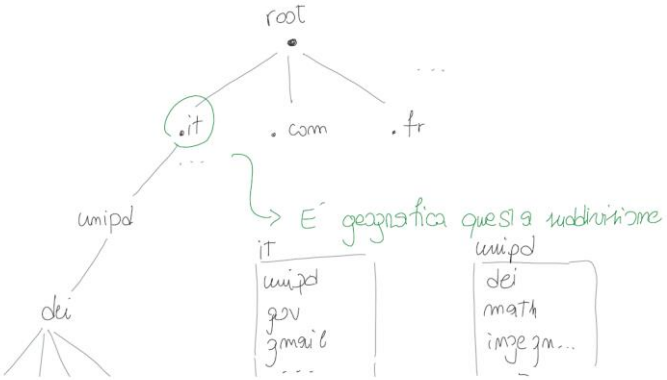


4.3. DNS

Tabella di conversione, corrispondenza IP<-->nomi

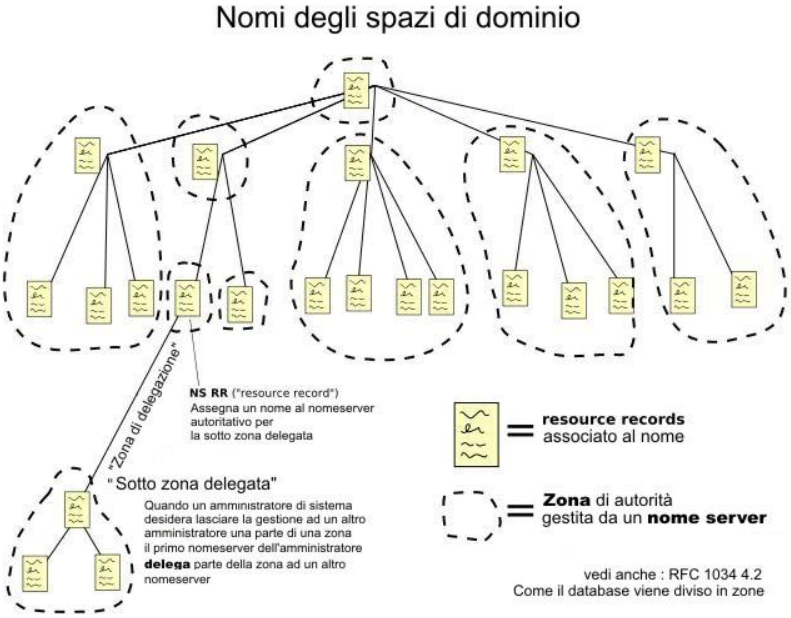
Il servizio è realizzato tramite un database distribuito, costituito dai server DNS. Il DNS ha una struttura gerarchica ad albero rovesciato ed è diviso in domini (com, org, it, ecc.). Ad ogni dominio o nodo corrisponde un nameserver, che conserva un database con le informazioni di alcuni domini di cui è responsabile e si rivolge ai nodi successivi quando deve trovare informazioni che appartengono ad altri domini. Ogni nome di dominio termina con un "." (punto). Ad esempio l'indirizzo wikipedia.org termina con il punto. La stringa vuota che segue il punto finale è chiamata dominio radice (DNS root zone). I server responsabili del dominio radice sono i cosiddetti root nameservers. Essi possiedono l'elenco dei server autoritativi di tutti i domini di primo livello (TLD) riconosciuti e lo forniscono in risposta a ciascuna richiesta. I root nameserver sono 13 in tutto il mondo. Il resolver fa richiesta ad un nameserver che gli dice che percorso fare nell'albero per trovare l'ip.

Per ottenere la risoluzione di un nome è necessario partire dalla radice, interrogare uno dei root server nel dominio di primo livello, ottenere il server che lo gestisce, interrogarlo nel dominio di secondo livello, fino a raggiungere il server autorevole per il nome desiderato.



Zona: pezzo di sotto albero

Server ricorsivo: server che viene configurato in una popolazione di client, che si occupa di risolvere le query che riceve interrogando i server originali, e mantenendo una cache delle risposte ricevute.

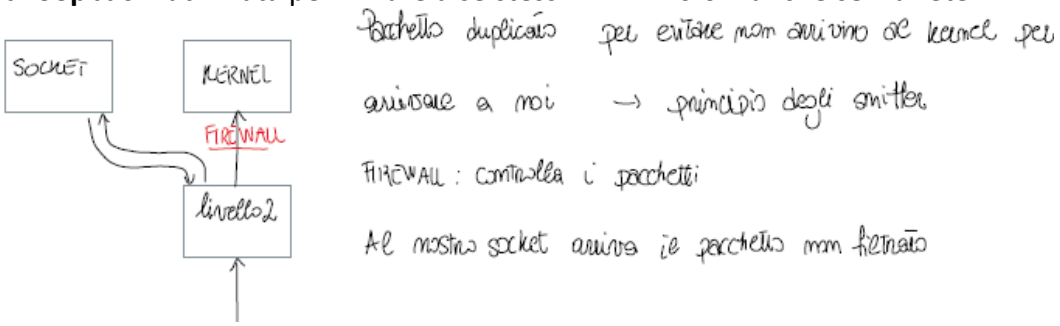


Di solito non si fa richiesta al root ma al NS del dominio a cui sono connesso.

Protocollo point to point

Protocollo di livello 2, meno utilizzato
Avvia una negoziazione sulle proprietà del canale.
Viene fatto del framing sui pacchetti.
Si usa la famiglia `AF_PACKET`

Interfaccia loopback: utilizzata per inviare a sé stessi. IP: 127.0.0.1 anche senza rete.



Supernetting: è un blocco contiguo di sottoreti accorpate insieme e indirizzate come fossero un'unica sottorete.
È una maschera per fare l'opposto del subnetting, cioè invece di suddividere la rete, si aggregano reti più piccole.
CIDR (Classless Inter-Domain Routing): a.b.c.d/x, dove x è il numero di bit (contati partendo dal più significativo a sinistra) che compongono la parte di indirizzo della rete. I rimanenti y=(32-x) bit consentono di calcolare il numero di host della sottorete pari a 2^y-2.
L'accesso è il punto più costoso

Address Allocation for Private Internets (RFC1918)

```
Hosts within enterprises that use IP can be partitioned into three
categories:

   Category 1: hosts that do not require access to hosts in other
               enterprises or the Internet at large; hosts within
               this category may use IP addresses that are
               unambiguous within an enterprise, but may be
               ambiguous between enterprises.

   Category 2: hosts that need access to a limited set of outside
               services (e.g., E-mail, FTP, netnews, remote login)
               which can be handled by mediating gateways (e.g.,
               application layer gateways). For many hosts in this
               category an unrestricted external access (provided
               via IP connectivity) may be unnecessary and even
               undesirable for privacy/security reasons. Just like
               hosts within the first category, such hosts may use
               IP addresses that are unambiguous within an
               enterprise, but may be ambiguous between
               enterprises.

    Category 3: hosts that need network layer access outside the
               enterprise (provided via IP connectivity); hosts in
               the last category require IP addresses that are
               globally unambiguous.
```

Host della prima e seconda categoria sono "privati", quelli della terza pubblici.
Indirizzi privati…???? Lez31

5.LIVELLO 4: Trasporto (TCP RFC 793)

Deve capire cosa sta succedendo nella rete sottostante nonostante no abbia informazioni da essa. Implementato dal protocollo TCP **RFC 793**

5.1. Descrizione

È presente solo sui terminali di rete (host) e non sui nodi interni di commutazione della rete di trasporto, implementato come strato software di rete all'interno del rispettivo sistema operativo ed il sistema terminale in trasmissione vi accede attraverso l'uso di opportune chiamate di sistema definite nelle API di sistema.

Funzioni:

- Stream over packet
- Connection over packet switching
- Flow control
- Congestion control
- Error check sui dati

5.1.1. Segmento TCP

La PDU di TCP è detta segmento. Ciascun segmento viene normalmente imbustato in un pacchetto IP, ed è costituito dal header TCP e da un payload, ovvero dati di livello applicativo. I dati contenuti nell'header costituiscono un canale di comunicazione tra le due entità TCP, che viene utilizzato per realizzare le funzionalità dello strato di trasporto e non è accessibile agli strati dei livelli superiori.

TCP Header

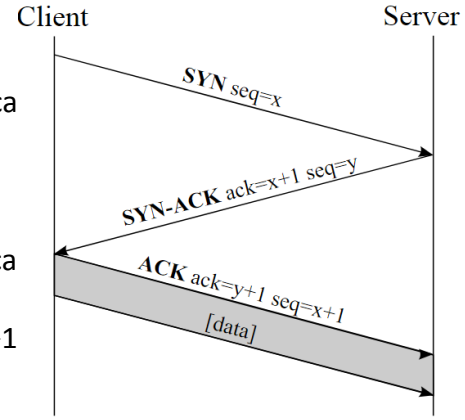
Bit offset	Bits 0–3	4–7	8–15								16–31
0	Source port										Destination port
32	Sequence number										
64	Acknowledgment number										
96	Data offset	Reserved	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size
128	Checksum										Urgent pointer
160	Options (optional)										
160/192+	Data										

- **Source port [16 bit]** - Identifica il numero di porta sull'host mittente associato alla connessione TCP.
- **Destination port [16 bit]** - Identifica il numero di porta sull'host destinatario associato alla connessione TCP.
- **Sequence number [32 bit]** - Numero di sequenza, indica lo scostamento (espresso in byte) dell'inizio del segmento TCP all'interno del flusso completo, a partire dall'Initial Sequence Number (ISN), negoziato all'apertura della connessione.
- **Acknowledgment number [32 bit] -** Numero di riscontro, ha significato solo se il flag ACK è impostato a 1, e conferma la ricezione di una parte del flusso di dati nella direzione opposta, indicando il valore del prossimo Sequence number che l'host mittente del segmento TCP si aspetta di ricevere.

- **Data offset [4 bit] -** Indica la lunghezza (in dword da 32 bit) dell'header del segmento TCP; tale lunghezza può variare da 5 dword (20 byte) a 15 dword (60 byte) a seconda della presenza e della lunghezza del campo facoltativo Options.
- **Reserved [4 bit] -** Bit non utilizzati e predisposti per sviluppi futuri del protocollo; dovrebbero essere impostati a zero.
- **Flags [8 bit] -** Bit utilizzati per il controllo del protocollo:
    - **CWR (Congestion Window Reduced)** - se impostato a 1 indica che l'host sorgente ha ricevuto un segmento TCP con il flag ECE impostato a 1 (aggiunto all'header in RFC 3168).
    - **ECE [ECN (Explicit Congestion Notification) -Echo] -** se impostato a 1 indica che l'host supporta ECN durante il 3-way handshake (aggiunto all'header in RFC 3168).
    - **URG** - se impostato a 1 indica che nel flusso sono presenti dati urgenti alla posizione (offset) indicata dal campo Urgent pointer. Urgent Pointer punta alla fine dei dati urgenti;
    - **ACK -** se impostato a 1 indica che il campo Acknowledgment number è valido;
    - **PSH** - se impostato a 1 indica che i dati in arrivo non devono essere bufferizzati ma passati subito ai livelli superiori dell'applicazione;
    - **RST** - se impostato a 1 indica che la connessione non è valida; viene utilizzato in caso di grave errore; a volte utilizzato insieme al flag ACK per la chiusura di una connessione.
    - **SYN -** se impostato a 1 indica che l'host mittente del segmento vuole aprire una connessione TCP con l'host destinatario e specifica nel campo Sequence number il valore dell'Initial Sequence Number (ISN); ha lo scopo di sincronizzare i numeri di sequenza dei due host. L'host che ha inviato il SYN deve attendere dall'host remoto un pacchetto SYN/ACK.
    - **FIN -** se impostato a 1 indica che l'host mittente del segmento vuole chiudere la connessione TCP aperta con l'host destinatario. Il mittente attende la conferma dal ricevente (con un FIN-ACK). A questo punto la connessione è ritenuta chiusa per metà: l'host che ha inviato FIN non potrà più inviare dati, mentre l'altro host ha il canale di comunicazione ancora disponibile. Quando anche l'altro host invierà il pacchetto con FIN impostato la connessione, dopo il relativo FIN-ACK, sarà considerata completamente chiusa.
- **Window size [16 bit] -** Indica la dimensione della finestra di ricezione dell'host mittente, cioè il numero di byte che il mittente è in grado di accettare a partire da quello specificato dall'acknowledgment number.
- **Checksum [16 bit] -** Campo di controllo utilizzato per la verifica della validità del segmento. È ottenuto facendo il complemento a 1 della somma complemento a uno a 16 bit dell'intero header TCP (con il campo checksum messo a zero),dell'intero payload, con l'aggiunta di uno pseudo header composto da: indirizzo IP sorgente(32bit),indirizzo IP destinazione(32bit), un byte di zeri, un byte che indica il protocollo e due byte che indicano la lunghezza del pacchetto TCP (header + dati).
- **Urgent pointer [16 bit]** - Puntatore a dato urgente, ha significato solo se il flag URG è impostato a 1 ed indica lo scostamento in byte a partire dal Sequence number del byte di dati urgenti all'interno del flusso.
- **Options -** Opzioni (facoltative) per usi del protocollo avanzati.
- **Data -** rappresenta il carico utile o payload da trasmettere cioè la PDU proveniente dal livello superiore.

5.2. Connessione

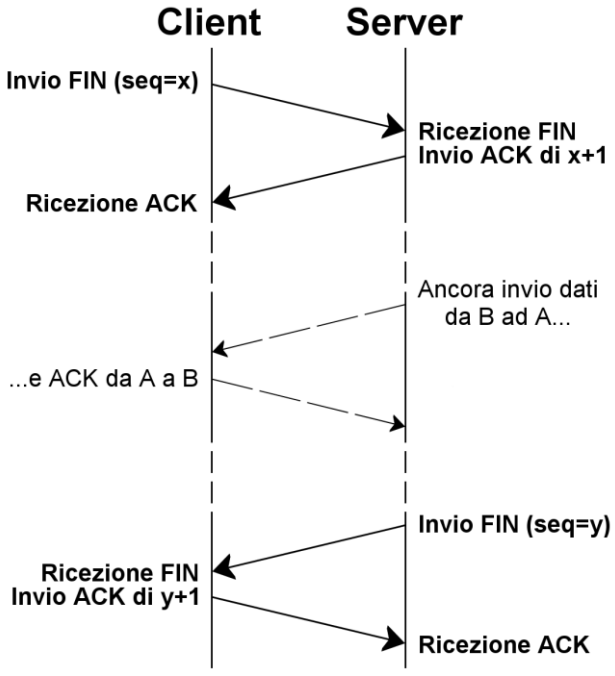
5.2.1. Apertura di una connessione - Three-way handshake



- **A invia un segmento SYN a B**:
  - flag SYN impostato a 1
  - campo Sequence number contiene il valore x che specifica l'Initial Sequence Number di A;
- **B invia un segmento SYN/ACK ad A**:
  - flag SYN e ACK impostati a 1
  - campo Sequence number contiene il valore y che specifica l'Initial Sequence Number di B
  - campo Acknowledgment number contiene il valore x+1 confermando la ricezione del ISN di A;
- **A invia un segmento ACK a B:**
  - flag ACK impostato a 1
  - campo Acknowledgment number (32bit) contiene il valore y+1 confermando la ricezione del ISN di B

5.2.2. Chiusura di una connessione - Four-way handshake

Dopo che è stata stabilita, una connessione TCP non è considerata una singola connessione bidirezionale, ma piuttosto come l'interazione di due connessioni monodirezionali. Pertanto, ognuna delle parti deve terminare la sua connessione, e possono esistere anche connessioni aperte a metà, in cui solo uno dei due terminali ha chiuso la connessione e non può più trasmettere, ma può (e deve) ricevere i dati dall'altro terminale.



Connessione tra A e B chiusa

5.3. Consegna ordinata ed eliminazione di duplicati

Il **Sequence number**, o numero di sequenza, serve a identificare e posizionare in maniera ordinata il carico utile del segmento TCP all'interno del flusso di dati. Con la trasmissione tipica a commutazione di pacchetto della rete dati infatti ciascun pacchetto può seguire percorsi diversi in rete e giungere fuori sequenza in ricezione.

In ricezione TCP si aspetta di ricevere il segmento successivo all'ultimo segmento ricevuto in ordine ovvero quello il cui numero di sequenza è pari al numero di sequenza dell'ultimo pervenuto in ordine più la dimensione del carico utile del segmento in attesa (cioè del suo campo Data).

In relazione al numero di sequenza TCP il ricevente attua le seguenti procedure:

- se il numero di sequenza ricevuto è quello atteso invia direttamente il carico utile del segmento al processo di livello applicativo e libera i propri buffer.
- se il numero di sequenza ricevuto è maggiore di quello atteso deduce che uno o più segmenti ad esso precedenti sono andati persi, ritardati dal livello di rete sottostante o ancora in transito sulla rete. Pertanto, memorizza temporaneamente in un buffer il carico utile del segmento ricevuto fuori sequenza per poterlo consegnare al processo applicativo solo dopo aver ricevuto e consegnato anche tutti i segmenti precedenti non ancora pervenuti passanti anch'essi per il buffer, aspettandone l'arrivo fino ad un tempo limite prefissato (time-out). All'istante di consegna del blocco ordinato di segmenti tutto il contenuto del buffer viene liberato. Dal punto di vista del processo applicativo, quindi, i dati arriveranno in ordine anche se la rete ha per qualsiasi motivo alterato questo ordine realizzando così il requisito della **consegna ordinata dei dati**.
- se il numero di sequenza ricevuto è inferiore a quello atteso, il segmento viene considerato un duplicato di uno già ricevuto e già inviato allo strato applicativo e dunque scartato. Questo permette di realizzare **l'eliminazione dei duplicati di rete.**

5.4. Riscontro dei pacchetti e ritrasmissione

Per ogni segmento ricevuto in sequenza il TCP ricevente invia un Acknowledgment Number o numero di riscontro dell'avvenuta ricezione. Il numero di riscontro presente in un segmento riguarda il flusso di dati nella direzione opposta. In particolare, il numero di riscontro inviato da A (Ricevente) a B è pari al numero di sequenza atteso da A e, quindi, riguarda il flusso di dati da B ad A, una sorta di report su ciò che è stato ricevuto.

In particolare, il protocollo TCP adotta la politica di Conferma cumulativa, ovvero l'arrivo di numero di riscontro indica al TCP trasmittente che il ricevente ha ricevuto e correttamente inoltrato al proprio processo applicativo il segmento avente numero di sequenza pari al numero di riscontro indicato (-1) ed anche tutti i segmenti ad esso precedenti. Per tale motivo TCP lato trasmittente mantiene temporaneamente in un buffer una copia di tutti i dati inviati, ma non ancora riscontrati: quando questi riceve un numero di riscontro per un certo segmento, deduce che tutti i segmenti precedenti a quel numero sono stati ricevuti correttamente liberando il proprio buffer dai dati. La dimensione massima dei pacchetti riscontrabili in maniera cumulativa è specificata dalle dimensioni della cosiddetta finestra scorrevole.

Per evitare tempi di attesa troppo lunghi o troppo corti per ciascun segmento inviato TCP lato trasmittente avvia un timer, detto timer di ritrasmissione RTO (Retransmission Time Out): se questi

non riceve un ACK di riscontro per il segmento inviato prima che il timer scada, TCP assume che tutti i segmenti trasmessi a partire da questo siano andati persi e quindi procede alla ritrasmissione.

Si noti che, in TCP, il meccanismo dei numeri di riscontro non permette al ricevitore di comunicare al trasmettitore che un segmento è stato perso, ma alcuni dei successivi sono stati ricevuti (meccanismo ad Acknowledgment Number negativi), quindi è possibile che per un solo pacchetto perso ne debbano essere ritrasmessi molti. Questo comportamento non ottimale è compensato dalla semplicità del protocollo. Questa tecnica è detta Go-Back-N (vai indietro di N segmenti); l'alternativa, ovvero progettare il protocollo in modo tale che solo i pacchetti effettivamente persi vengano ritrasmessi, è detta Selective Repeat (ripetizione selettiva); l'utilizzo però di alcuni campi opzionali appositi permette l'utilizzo della ripetizione selettiva.

I numeri di riscontro e i relativi timer di ritrasmissione permettono quindi di realizzare la consegna affidabile, ovvero di garantire che tutti i dati inviati siano comunque consegnati nel caso in cui qualche pacchetto possa essere perso nel transito attraverso la rete (controllo di errore in termini di riscontro di trasmissione).

5.5. Controllo della congestione

5.5.1. Descrizione

Vuol dire che vengono buttati via pacchetti.
La rilevazione della congestione sia effettuata agli estremi della comunicazione/connessione, e non richiede nessun supporto da parte dei router intermedi per realizzare questa funzione.
Si basa sul fatto che le due entità messe in comunicazione dal TCP si scambiano pacchetti contenenti dati e pacchetti di riscontro (ACK), che viaggiano nel verso opposto rispetto ai dati.
RTT (Round Trip Time): tempo che intercorre tra l'invio di un segmento e la ricezione del relativo riscontro.
In presenza di congestione, le code nei router diventano più lunghe, e questo fa aumentare il ritardo subito dai pacchetti e dai relativi riscontri, e quindi l'RTT.
TCP impone a ciascun mittente un limite alla velocità di invio dei pacchetti in funzione della congestione di rete percepita.
CongWin (Congestion Window): variabile il cui valore impone un limite superiore alla quantità di dati trasmessi e non ancora riscontrati dal mittente.
Se il mittente rileva che sul percorso di invio dei pacchetti c'è assenza di congestione, incrementa il valore di CongWin, viceversa se rileva segnali di congestione, lo riduce.
Un mittente TCP può rilevare la presenza di congestione della rete quando si verificano eventi che sono sintomi della perdita di pacchetti, ad esempio:

- scadenza di un time-out a causa della mancata ricezione di ACK;
- ricezione di pacchetti di ACK con lo stesso numero di sequenza (ACK duplicati), che manifestano la presenza di un "buco" nella sequenza di pacchetti ricevuti.

5.5.2. Algoritmo

2 fasi:

- Partenza Lenta (Slow-Start)
- AIMD (Additive Increase Multiplicative Decrease)

SSTHRESH: variabile che distingue le due fasi. Quando CongWin > SSTHRESH si avvia AIMD.
MSS (Maximum Segment Size): è la massima dimensione del Payload del segmento TCP.

Slow-Start: il mittente TCP inizia trasmettendo il primo segmento dati ed attende un riscontro. Ogni volta che un segmento trasmesso viene riscontrato (si riceve un ACK) il mittente incrementa la CongWin di una quantità pari al MSS. Quindi ad ogni RTT la CongWin raddoppia di dimensioni fino a raggiungere la metà della dimensione massima. Da questo punto in avanti l'aumento procede in maniera lineare fino a raggiungere il massimo. Questo valore viene mantenuto finché non si incorre in un evento di congestione o quando CongWin > SSTHRESH, quindi si avvia AIMD.

AIMD: si ha un incremento additivo lineare del valore di CongWin di 1 MSS ogni RTT.
TCP Reno:

- Alla ricezione di tre ACK duplicati consecutivi (con lo stesso numero di riscontro) imposta il valore di SSTRESH a CongWin/2 e assegna a CongWin tale valore incrementato di 3 MSS.
- Allo scadere di un timeout di ritrasmissione, reagisce dimezzando il valore di SSTHRESH e reimpostando CongWin alla dimensione di un segmento, tornando quindi nella fase di Slow-Start.

Impostazione del timeout RTO= SRTT +4*RTTVAR
Si vuole RTO>RTT, ma RTT cambia continuamente quindi si usano i parametri:

- SRTT= media dei valori di RTT misurati
- RTTVAR=stima della deviazione standard della variabile RTT

5.6. Controllo di flusso

L'obiettivo di tale controllo è evitare che il mittente invii una quantità eccessiva di dati che potrebbero, in alcune situazioni, mandare in overflow il buffer di ricezione del destinatario generando una perdita di pacchetti e la necessità di ritrasmissione con perdita in efficienza.
Il controllo di flusso è a livello logico e fisico agli estremi della connessione dipendendo esclusivamente dalle capacità riceventi del ricevitore.

LastByteRead: Numero dell'ultimo byte nel flusso di dati letto dal processo applicativo
LastByteRcvd: Numero dell'ultimo byte copiato nel buffer di ricezione di B.
LastByteRcvd - LastByteRead <= RcvBuffer
RcvWindow (Finestra di ricezione)=RcvBuffer - (LastByteRcvd - LastByteRead)
Variabile che fornisce al mittente un'indicazione sullo spazio libero disponibile nel buffer del destinatario. Uguale alla quantità di spazio disponibile nel buffer.
Questo valore è comunicato al mittente direttamente dal ricevitore, in un apposito campo dell'intestazione TCP.

LastByteSent-LastByteAcked <=min(CongWin,RcvWindow)

6.LINGUAGGIO C

6.1. Tipi elementari

Fixed point

type	size (x86_32)
char	8bit
short	16bit
int	32bit
long	32bit
long long	64bit

Floating point

type	size (x86_32)
float	32bit
double	64bit
long double	80bit

Modificatori: `unsigned, signed`
stdint.h: int8_t, int16_t, int32_t, uint8_t, uint16_t, uint32_t

Non esiste il boolean, è considerato VERO qualsiasi tipo che ha almeno un un bit =1, Falso s tutti i bit=0.
Non esiste un dato indirizzabile più piccolo di 1 byte.

Costanti:

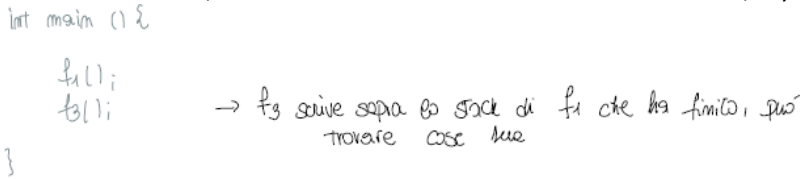
```
const [<modifiers>] <type> <const name> = <const value>;
```

Oppure: `#define <const name> = <const value>;`
Es: const unsigned short int MAX = 128;

Big endian: inizia con i bit meno significativi
Little endian: inizia con i bit più significativi, comodo perché il passaggio tra un tipo più grande avviene per troncamento. (Processori Intel) Es: 0x1234: (34 | 12)

Cast: La conversione non modifica i bit del dato, ma il tipo di indirizzo

Variabili globali: vengono inizializzate a 0, finiscono in una memoria statica.
Variabili locali: NON sono inizializzate a 0. Vengono inserite in uno stack, l'**activation record**, ovvero una struttura dati generata quando viene chiamata una funzione, contiene i parametri passati dalla riga chiamante, indirizzo di ritorno, variabili create all'interno della funzione, …).



Blocco: attraverso le graffe (non solo per funzioni o FOR/WHILE), crea un activation record per memorizzare variabili temporanee.

Operatori logici:

- !
- &&
- ||

Operatori aritmetici

Operation Name	Arithmetic Operator	Algebric Expression	C Expression
Addition	+	f + 7	f + 7
Subtraction	-	p – c	p - c
Multiplication	x	bm	b * m
Division	:	x / y	x / y
Remainder (mod)	mod	r mod s	r % s

Operators	Operations	Precedence
()	Parentheses	**Evaluated first.** If parentheses are nested, the expression in the innermost pair is evaluated first. In case of several pairs of parentheses "on the same level" they are evaluated left to right.
*, /, %	Multiplication, Division, Remainder	**Evaluated second.** If there are several, they are evaluated left to right.
+, -	Addition, Subtraction	**Evaluated last.** If there are several, they are evaluated left to right.

Operatori sui bit

Operation Name	C Operator	Algebric example	C Expression
AND	&	01b AND 10b = 00b	a & b
OR	|	01b OR 10b = 11b	a | b
XOR	^	01b XOR 10b = 11b	a ^ b
NOT	~	NOT(01b) = 10b	~ a
SHIFT Left	<<	10b SHIFTl(2) = 1000b	a << 2
SHIFT Right	>>	1000b SHIFTr(2) = 10b	a >> 2

Assegnazioni composte

Operation name	C operator	C example	C equivalent
Increment by RHS	+=	a += b;	a = a + b;
Decrement by RHS	-=	a -= b;	a = a - b;
Multiply by RHS	*=	a *= b;	a = a * b;
Divide by RHS	/=	a /= b;	a = a / b;
Mod by RHS	%=	a %= b;	a = a % b;
Bitwise AND by RHS	&=	a &= b;	a = a & b;
Bitwise OR by RHS	|=	a |= b;	a = a | b;
Bitwise XOR by RHS	^=	a ^= b;	a = a ^ b;
Bitwise left shift by RHS	<<=	a <<= b;	a = a << b;
Bitwise right shift by RHS	>>=	a >>= b;	a = a >> b;

(RHS = Right Hand Side)

Operatori relazionali

Operator name	C Operator	Return **0** Example	Return **1** Example
Equal	==	55 == 23	55 == 55
Not Equal	!=	55 != 55	55 != 23
Greater Than	>	23 > 55	55 > 23
Less Then	<	55 < 23	23 < 55
Greater or Equal	>=	23 >= 55	23 >= 23
Less or Equal	<=	55 <= 23	55 <= 55

6.2. Array

```
[<modifiers>] <type> <array name>[element count];
```

Accesso: `i = scores[17];`

Inizializzazione:

```
unsigned int scores[123] = { 1, 2, 3, 4, 5};
unsigned int scores[] = { 1, 2, 3, 4, 5};
char hello[123] = "hello\n";
char hello[] = "hello\n";
```

Multidimensionale:

```
[<modifiers>] <type> <array name>[count1][count2] … [countN];
```

Inizializzazione multidimensionale:

```
unsigned int scores[123][45]= {{1, 2, 3}, {4, 5, 6}};
unsigned int scores[][3] = {{1, 2, 3}, {4, 5, 6}};
```

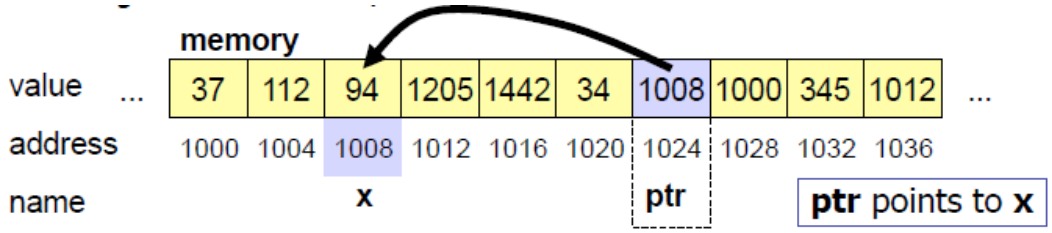
Array di struct e union:…….

Carattere terminatore di stringa: \0

6.3. Puntatori

La memoria può essere considerata un array di elementi (locazioni), in cui ogni elemento ha un indirizzo e un valore.

Un puntatore è una variabile che contiene un indirizzo di memoria.



Dichiarazione:

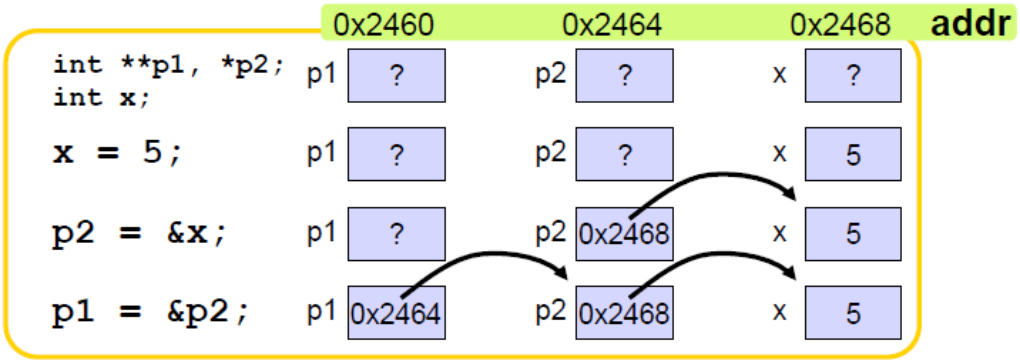
```
[<modifiers>] <type> * <pointer name>;
```

All'inizio il puntatore punta ad un indirizzo a caso.

Es. `int *ptr;`

Operatori:

- &: ottiene l'indirizzo della variabile
- *: ottiene il valore a contenuto nell'indirizzo a cui sta puntando



**p1 == *p2 == x

6.4. printf

```
int printf(const char* format_string, ...)
```

Stampa una stringa in output, la stringa è il primo argomento, gli altri argomenti sono i valori che combaciano con la direttiva %.
Precisione: %.3f

Es:

```
printf("%s %c%d", "great", 'A', 3);
OUTPUT: great A3
printf("%.3f\n", 0.123456789f)
OUTPUT number: 0.123
```

Type	shorthand
char	%c %hhd %hhu
short	%hd
unsigned short	%ho %hu %hx %hX
int	%d
unsigned int	%o %u %x %X
long	%ld
unsigned long	%lo %lu %lx %lX
float	%f
double	%f
long double	%lf

6.5. Strutture

Definire un nuovo tipo:

```
struct <struct name>{
    <type> <variable name>;
    ...
    <type> <variable name>;
};
```

Dichiarazione: `[<modifiers>] struct <struct name> <var struct name>;`
Accesso ai dati: `<var struct name>.<variable name>`

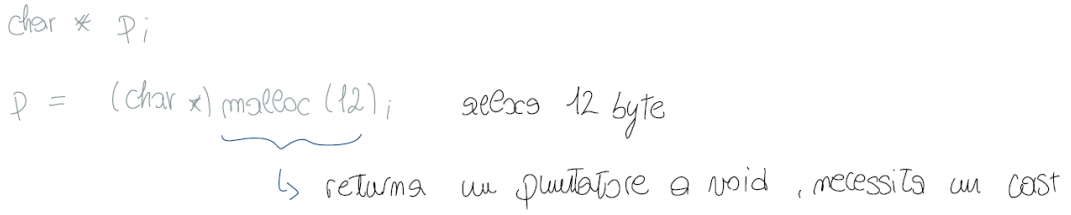
Problema dell'allineamento: non posso mettere un tipo di 2byte in una cella dispari, il compilatore deve allineare (aggiunge un padding).
Ogni dato viene gestito nella memoria (in righe grandi) a multipli della sua dimensione, in una struttura ogni dato viene gestito a multipli del dato massimo della struttura, quindi ai dati più piccoli viene aggiunto un padding.

6.6. Allocazione dinamica:

malloc(N): funzione che alloca dello spazio in memoria. Ritorna un puntatore a carattere che corrisponde al punto di inizio in memoria della porzione riservata di dimensione N. Se la memoria richiesta non può essere allocata, ritorna un puntatore nullo(indirizzo 0).

```
p=(char*)malloc(N);
```

free(p): dis-alloca la memoria.



realloc(void *ptr, size_t size): permette di cambiare la grandezza di un area di memoria già allocata. Con il puntatore NULL si comporta come malloc.

`ptr`: is the pointer to a memory block previously allocated with malloc
`size`: This is the new size for the memory block, in bytes. If it is 0 and ptr points to an existing block of memory, the memory block pointed by ptr is deallocated and a NULL pointer is returned.

6.7. Chiamate a sistema e funzioni

open(file): chiamata a sistema che restituisce un INT, che è un File Descriptor che viene usato per leggere/scrivere il file.

atoi(): converte una stringa in int

fopen(file,modo): funzione C che al suo interno ha anche una open, restituisce un puntatore a struttura.

```c
#include <stdio.h>

int main (){

    FILE * f;
    f = fopen("prova.txt","wt");
    if(f == NULL){
        printf("Errore apertura file\n");
        return 0;

    }

    fprintf(f,"Ciao\n");
    fclose(f);
}
```

fflush(): forza la scrittura dei dati bufferizzati sullo stream.

socket(2): è una chiamata a sistema che crea un endpoint (apre una comunicazione) e restituisce un INT, che è un File Descriptor ovvero l'indice della tabella con tutto ciò che serve per gestire la comunicazione.

```
int socket(int domain, int type, int protocol);
```

- `domain`: `AF_INET`: ipv4
- `type`:
  - `SOCK_STREAM`: apre una comunicazione e invia una sequenza di byte illimitata, che si ferma solo quando viene chiusa la comunicazione. Usata per l'invio dei file.
  - `SOCK_DGRAM`: messaggio di lunghezza finita, quando è stato inviato la comunicazione è chiusa. Usato per comunicazioni real time, messaggi, videochiamate.
  - `SOCK_SEQPACKET`: pacchetti con numero progressivo
  - `SOCK_RAW`: accesso al protocollo di rete non elaborato. (Ethernet)
- `protocol`: di solito 0, `htons(ETH_P_ALL)` per ethernet.
  ETH_P_IP, IPPROTO_RAW, IPPROTO_ICMP

```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>

int main() {

    int s;
    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s == -1){

        perror("SOCKET FALLITA"); //mette i : in automatico e una stringa di errore

    }

}
```

connect(2): è una chiamata a sistema per connettersi al socket riferito dal file descriptor `sockfd` all'indirizzo specificato da `addr`. `addrlen` specifica la lunghezza dell'indirizzo.

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Funziona per tutte le strutture dati dei protocolli, prende l'indirizzo di memoria di un blocco, la sua lunghezza e per capire il tipo di struttura, si usa un identificatore come primo indirizzo della struttura.

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    uint16_t       sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

`sockaddr` è generica (le socket unix non sono solo per il TCP/IP , ma supportano anche altri protocolli), mentre la `sockaddr_in` è dedicata all' IPv4 cioè per essere usata con l'Internet Protocol (TCP o UDP per il trasporto).

Il server in realtà è un programma, quindi servono 2 indirizzi, uno per la macchina (addr) e uno per il programma (port).
Port non è casuale ma è definito da una tabella dei servizi standard (vi /etc/services).
Il livello 4 Transport sa a che applicazione mandare i dati attraverso Port.
Su port deve essere tutto scritto in network order per evitare inconsistenze tra little e big endian.
I port superiori a 1024 sono "usa e getta".
inet_addr(): funzione che converte gli indirizzi IPv4 con punti (a.b.c.d) in valori binari in network byte order.
Domain name system: struttura nomi-->indirizzi
nslookup indirizzoweb: restituisce ip
netstat: restituisce connessioni aperte.

write(2): su un file descriptor

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

Su `count` di solito si utilizza `strlen(`array di char`)`.

bind(2): serve per collegare il socket al port. Assegna un indirizzo al socket.

```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

listen(2): mette in attesa il programma su un socket, marca il socket come passivo, ovvero che dovrà ricevere richieste di conessione.

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

`sockfd`: file descriptor del socket.
`backlog`: definisce la massima lunghezza della coda che gestisce le connessioni in attesa (n° di connessioni in attesa massimo).

accept(2): restituisce il file descriptor di un nuovo socket che eredita tutto da quello precedente, ma è già connesso, pronto per read e write. Questo perché l'altro socket è impegnato con listen.

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

`sockfd`: file descriptor del socket.

read(2): da un file descriptor

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

gethostbyname(3): ritorna una struct IPv4 dato il nome dell'host

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);
```

fork(2): crea un nuovo processo duplicando il processo chiamante, il nuovo processo è identico al padre accetto che ha un suo processID (PID). Al padre restituisce il PID del figlio, al figlio il valore 0. Ritorna -1 se c'è un fallimento.
Usata perché così una connessione da un client è già accettata e possono essere eseguiti più processi senza accettarli uno per uno.

```
#include <unistd.h>
pid_t fork(void);
```

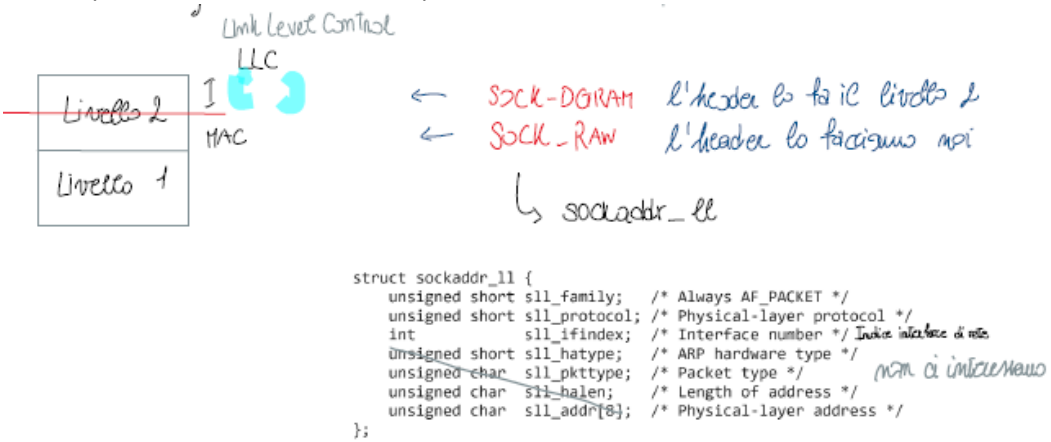
telnet www.google.com: apre un socket e posso scrivere da tastiera sullo stream

packet(7): usato per ricevere/inviare pacchetti RAW al livello 2.

```
#include <sys/socket.h>
#include <netpacket/packet.h>
#include <net/ethernet.h> /* the L2 protocols */

socket(AF_PACKET, int socket_type, int protocol);
```

`socket_type`: può essere `SOCK_RAW` (per pacchetti che includono il link level header) o `SOCK_DGRAM` (link level header rimosso)



if_nametoindex(3): ritorna l'indice dell'interfaccia corrispondente al parametro `ifname`

```
#include <net/if.h>
unsigned int if_nametoindex(const char *ifname);
```

sendto(2): per trasmettere un pacchetto ethernet

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t sendto(int sockfd, const void *buf, size_t len, int
flags, const struct sockaddr *dest_addr, socklen_t addrlen);
```

recvfrom(2): serve per ricevere pacchetti ethernet

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
        struct sockaddr *src_addr, socklen_t *addrlen);
```

memcmp(): confronta i primi n byte di area di memoria str1 e area di memoria str2.

```
int memcmp(const void *str1, const void *str2, size_t n))
```

memcpy(): copia n caratteri dall'area di memoria str2 all'area di memoria str1.

```
void *memcpy(void *str1, const void *str2, size_t n)
```

gettimeofday()….

Comandi Linux

Connessione con Putty: `ssh 88.80.187.84`
Connessione linux: `ssh nomeutente@88.80.187.84`

Cartella con esercizi: `cd /reti18/`
Mia Cartella: `cd /home/nomeutente/`
Compilare: `gcc nomefile.c -o nomefile`
Eseguire: `./nomefile`
Copiare file: `cp` *`sorgente destinazione`*

Modifica file: `vim nomefile.c`
Opzioni VIM:

- Esci: `:q`
- Salva: `:w`
- Indentazione: `:set tabstop=2`
- Numeri righe: `:set nu` `:set nonu`
- Annulla: `Ctrl+u`

Tabella nomi/indirizzi necessaria per il debug: `gcc nomefile.c -o nomefile -g`
Debug: `gdb nomeEseguibile`