Capitolo 1

1. **Definire il concetto di protocollo e poi spiegare come funziona un protocollo particolare a scelta. Perché gli standard sono importanti nei protocolli?**

Un protocollo definisce il formato e l'ordine dei messaggi scambiati tra due o più entità in comunicazione all'interno della rete, come le azioni intraprese in fase di trasmissione/ricezione di un messaggio o di un altro evento.

La standardizzazione di un protocollo garantisce l'interoperabilità tra dispositivi e/o agent software indipendenti

HTTP:
Hypertext transfer protocol è implementato in due programmi, client e server, in esecuzione su sistemi periferici (host) diversi che comunicano tra loro scambiandosi messaggi HTTP. **Definisce la struttura dei messaggi e la modalità con cui i client e server si scambiano messaggi.**

Una **pagina web è costituita da oggetti. Un oggetto è un file indirizzabile tramite un URL.** Ogni **URL** ha due componenti: il **nome dell'host del server che ospita l'oggetto e il percorso dell'oggetto.**

HTTP definisce in che modo i client web richiedono le pagine ai web server e come essi le trasferiscono ai client. **Il browser invia al server messaggi di richiesta HTTP per gli oggetti nella pagina. Il server risponde con messaggi di risposta HTTP contenenti gli oggetti.**

HTTP utilizza TCP come protocollo di trasporto. **Il client inizia una connessione TCP al server e, una volta stabilita, i due processi accedono a TCP attraverso la propria socket. L'interfaccia socket è la porta tra un processo e la sua connessione TCP.**

Il server invia i file richiesti ai client senza memorizzare informazioni di stato a proposito del client. In caso di richiesta l'oggetto da parte dello stesso client, il server procederà nuovamente con l'invio, non avendo tenuto traccia del precedente.

HTTP dunque è un protocollo senza memoria di stato (stateless protocoll). Un web server è sempre attivo, ha un indirizzo IP fisso e risponde potenzialmente alle richieste provenienti da milioni di browser.

HTTP, nelle modalità di default, usa le connessioni non persistenti ma, i server e i client possono essere configurati per usare quelle persistenti.

FTP:
In una sessione FTP, l'utente utilizza un host per trasferire file da o verso un host remoto. Per essere autorizzato a scambiare informazioni con l'host remoto, l'utente deve **fornire un nome identificativo e una password**.

L'utente interagisce con FTP tramite un agente software (FTP user agent): **fornisce nome dell'host remoto**, in modo che il processo FTP client stabilisca una connessione TCP con il processo FTP server .
Ottenuta l'autorizzazione, il client può inviare uno o più file memorizzati nel file system locale verso quello remoto.
HTTP e FTP sono protocolli di trasferimento dei file e usano TCP.

Invia le proprie informazioni di controllo fuori banda, utilizza due connessioni TCP parallele:
connessione di controllo, che serve per inviare le informazioni tra gli host come nome utente, comandi ecc...
connessione dati, per l'invio dei file

Il lato ***client*** **inizializza una connessione di controllo TCP con il lato server sulla porta 21** del server, dove invia l'identificativo dell'utente e password. **Spedisce poi, i comandi per cambiare la directory remota.**

Il *server* inizializza una connessione TCP verso il client in cui invia esattamente un file prima di richiuderla.
Se l'utente volesse trasferire un altro file, FTP aprirebbe un'altra connessione dati. **La connessione di controllo rimane aperta per l'intera durata della sessione utente, ma si crea una nuova connessione dati per ogni file trasferito all'interno della sessione.**

Per tutto l'arco di una sessione, il server FTP deve mantenere lo stato dell'utente e deve associare la connessione di controllo ad uno specifico utente e tenere traccia della directory corrente dell'utente. Ciò limita il numero totale di sessioni che FTP riesce a gestire contemporaneamente.

SMTP:
I componenti della posta elettronica in internet sono 3:
- User agent, consentono agli utenti di leggere, inviare e ricevere i messaggi di posta
- Server di posta, gestisce e contiene i messaggi
- protocollo SMTP

E' un protocollo a livello di applicazione per la posta elettronica.
Utilizza il servizio di trasferimento di dati affidabile di TCP per trasferire le mail dal server mittente a quello destinatario.

Ha un lato client in esecuzione sul mail-server mittente e un lato server in esecuzione su quello destinatario.

1. Il mittente invoca il proprio user agent per la posta elettronica, fornisce l'indirizzo di poste del destinatario, compone il messaggio e da istruzione allo user agent di inviarlo
2. Lo user agent mittente invia il messaggio al suo mail server che lo colloca in una coda di messaggi
3. Il lato client di STMP vede il messaggio nella coda dei messaggi e apre una connessione TCP verso un server SMTP in esecuzione sul server destinatario.
4. Dopo un handshaking SMTP, il client SMTP invia il messaggio sulla connessione TCP.

5. Il lato server di SMTP riceve il messaggio e lo posiziona nella casella dell'utente.
6. Quando l'utente invece il proprio user agent può leggere il messaggio.

Il client SMTP stabilisce una connessione TCP sulla porta 25 verso il server SMTP. Se il server è inattivo, il client riprova più tardi.
Stabilita la connessione, server e client effettuano handshaking a livello applicativo in cui il client indica l'indirizzo email del mittente e del destinatario e solo dopo invia il messaggio.

SMTP fa uso di connessioni *persistenti:* per ogni messaggio il client inizia il processo con un nuovo MAIL FROM: e stabilisce la fine del messaggio con un punto slash n "./n".

SMTP è un protocollo *PUSH*: il mail server di invio spinge i file al mail server in ricezione. la connessione TCP viene iniziata dall'host che vuole spedire il file

POP3 (Post-Office-Protocol V.3)

Entra in azione quando lo user agent apre una connessione TCP verso il mail server sulla porta 110.
Ha 3 fasi:

1. *Autorizzazione*, lo user agent invia nome utente e password in chiaro per autenticare l'utente.
2. *Transazione,* lo user agent recupera i messaggi e può marcarli per la cancellazione e ottenere statistiche sulla posta;
3. *Aggiornamento* dopo che il client ha inviato il comando QUIT, il server rimuove i messaggi marcati per la cancellazione.

Uno user agent può essere configurato per " scaricare e cancellare " o per " scaricare e mantenere"

Nella prima modalità il destinatario potrebbe voler accedere ai propri messaggi di posta da più macchine, se l'utente legge un messaggio su un dispositivo non sarà in grado di leggerlo da un'altro.
Nella seconda lo user agent lascia i messaggio sul server di posta, dopo averli scaricati saranno disponibili su qualsiasi dispositivo.

Durante una sessione tra uno user agent e il mail server, il server POP3 mantiene traccia dei messaggi dell'utente marcati come cancellati. Non trasporta informazioni di stato tra diverse sessioni.

IMAP(Internet Mail Access Protocol)
Un server IMAP associa a una cartella ogni messaggio arrivato al server. I messaggi in arrivo sono associati alla cartella INBOX.
IMAP fornisce comandi per consentire agli utenti di creare cartelle e spostare i messaggi da una cartella a un'altra, consente agli utenti di effettuare ricerche nelle cartelle e permette agli user agent di ottenere singole parti dei messaggi.
I server IMAP conservano informazioni di stato sull'utente da una sessione all'altra.

Lo user agent può anche essere un browser e l'utente comunica con la propria casella remota via HTTP. Il messaggio email viene spedito dal server di posta al browser usando il protocollo HTTP

2.
Quale vantaggio presenta una rete a commutazione di circuito rispetto ad una a commutazione di pacchetto? Quali vantaggi ha TDM rispetto a FDM in una rete a commutazione di circuito?

Le risorse richieste per consentire la comunicazione tra sistemi periferici sono riservate per l'intera durata della sessione di comunicazione.
Prima che possa iniziare l'invio, la rete deve stabilire una connessione tra mittente e destinatario. I commutatori presenti sul percorso mantengono lo stato della connessione per tutta la durata della comunicazione. E' garantita una velocità di trasmissione costante.
Quando due host desiderano comunicare, la rete stabilisce una connessione end-to-end dedicata.

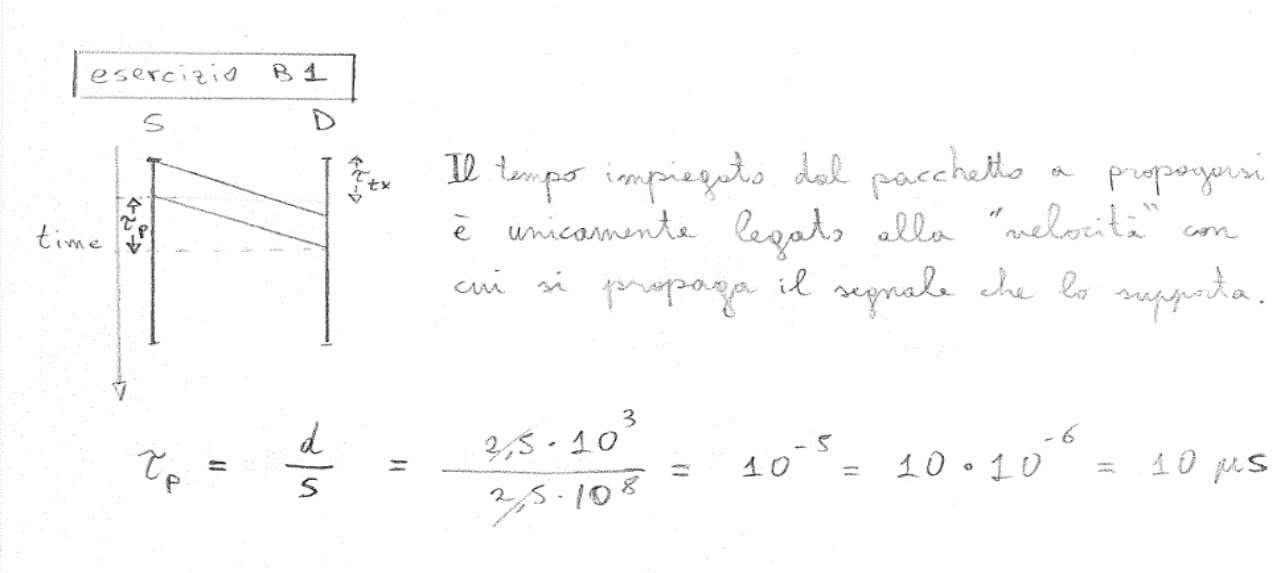
TDM rispetto a FDM in una rete a commutazione di circuito ha il vantaggio che dividendo in porzioni temporali fisse la connessione, la velocità di trasmissione di un circuito è uguale alla frequenza di frame moltiplicata per il numero di bit di uno slot.

3. Considerate l'invio di un pacchetto da un host ad un altro lungo un percorso fisso. Elencate le componenti di ritardo complessivo, indicando quali sono costanti e quali variabili.

Ritardo di elaborazione, variabile,viene esaminata l'intestazione del pacchetto e determinato dove dirigerlo.
Ritardo di accodamento, variabile,avviene mentre attende la trasmissione sul collegamento
Ritardo di trasmissione,variabile**,**è il tempo richiesto per trasmettere tutti i bit del pacchetto sul collegamento, la quantità di tempo richiesta da parte del router per trasmettere in uscita il pacchetto, è funzione della lunghezza del pacchetto e della velocità di trasmissione
Ritardo di propagazione, fisso,impiegato da ogni bit per propagarsi fino al router di destinazione

4. **Quanto tempo impiega un pacchetto di lunghezza di 1000 byte per propagarsi su un collegamento lungo 2500 km, con una velocità di propagazione di 2.5×10^8 m/s e velocità di trasmissione 2Mbps?**
    a. **Il *tempo di trasmissione* T_{TX} di un pacchetto su un canale è il periodo di tempo in cui il trasmettitore è impegnato ad inviare i bit del pacchetto sul canale. La durata del tempo di trasmissione è il rapporto tra il numero di bit di cui è composto il pacchetto e la velocità di trasmissione sul canale (espressa in bit/s).**
    b. **Il *tempo di propagazione* T_P è il tempo necessario ad ogni bit (o, più precisamente, al segnale che lo rappresenta) per percorrere il canale fino**

al nodo successivo. La durata del tempo di propagazione dipende esclusivamente dalla velocità di propagazione del segnale nel mezzo trasmissivo (espressa in m/s) e dalla lunghezza (espressa in m) del mezzo stesso.



5. **Il tempo di propagazione dipende dalla lunghezza del pacchetto? Dipende dalla velocità di trasmissione?**

Il tempo di propagazione NON dipende dalla lunghezza del pacchetto perché dipende solo dalla distanza tra i due host e la velocità della luce che è una costante. Non dipende nemmeno dalla velocità di trasmissione.

Tempo che intercorre dall'inizio della trasmissione di un pacchetto di dimensione pari a *b* bit fino all'avvenuta completa ricezione al nodo successivo, considerando la trasmissione su un solo canale di lunghezza *l* metri, con una velocità di trasmissione pari a *r* bit/s

$$T_{tot} = T_{TX} + T_P = b/r + l/c$$

c = velocità della luce

6. Si consideri una topologia di rete lineare composta da un singolo canale con velocità di trasmissione pari a 1 Mbit/s, come illustrato nella figura. Il nodo S deve trasmettere un file di dimensione pari a 9500 byte verso il nodo D. Si ipotizzi che:

- **Non vi siano errori di trasmissione**
- **Il tempo di propagazione sul canale sia pari a 5 ms**
- **La dimensione massima dei pacchetti trasmessi sul canale sia pari a 1500 byte (si trascurino le intestazioni)**

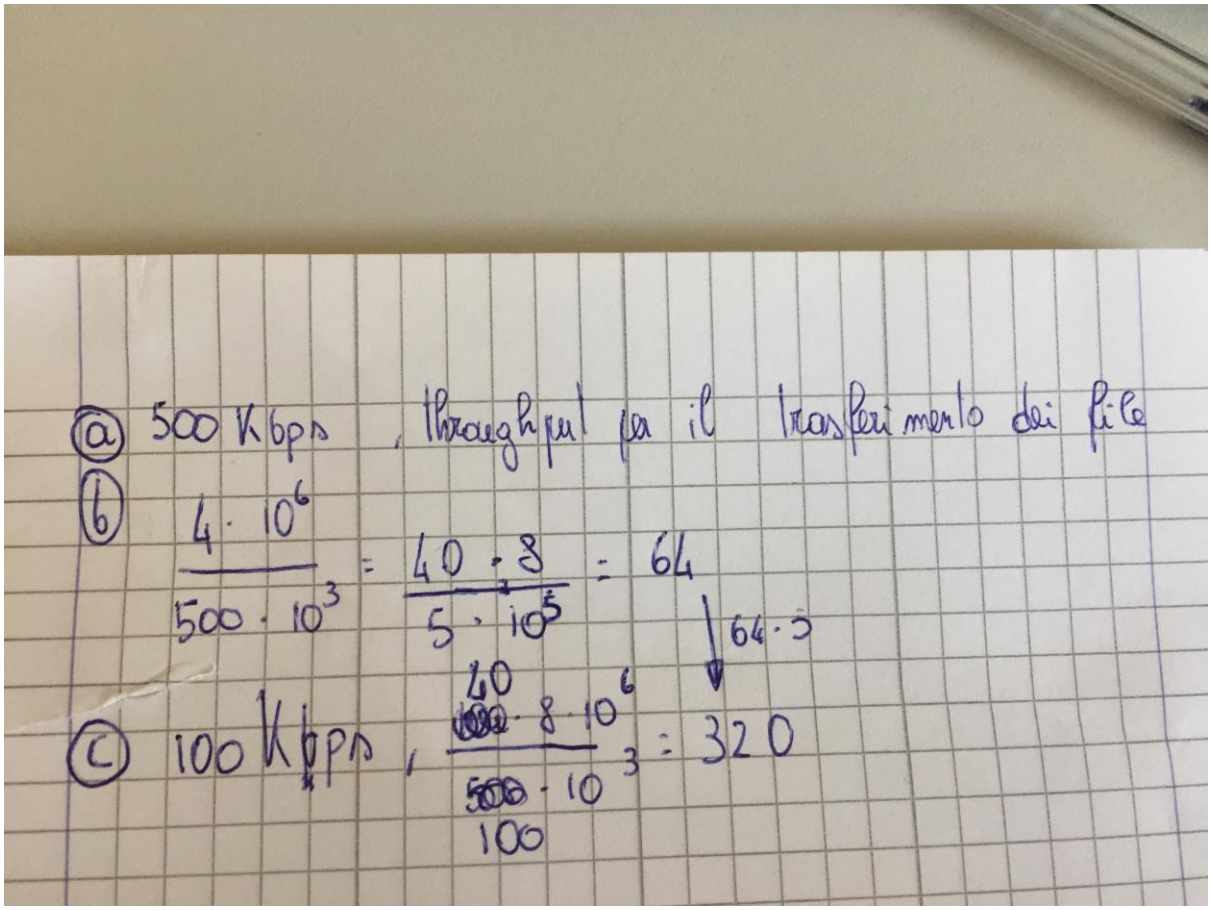
Determinare il tempo necessario affinché il nodo D riceva completamente il file.

11 **Esercizio A-1 - Soluzione**

t. Trasmissione = t. Trasmissione Pacchetti + t. Propagazione
Numero Pacchetti (*) >> 9500 byte /1500 byte = 6 con resto 500 byte
Velocità Trasmissione = 1Mb/s cioè 10-6 s x bit
Pacchetto 1500 byte >> 1500 x 8 x 10-6 = x 10-6 = 12 ms
Pacchetto 500 byte >> 500 x 8 x 10-6 = x 10-6 = 4 ms
Tempo Trasmissione = 6 x 12 ms + 1 x 4 ms + 5 ms = 81 ms
(*) = Data l'ipotesi di trascurare le intestazioni lo stesso risultato si ha eseguendo il calcolo sui 9500 byte

7. L'host A vuole inviare un file voluminoso all'Host B. Il percorso tra l'Host A e l'Host B ha tre collegamenti con frequenze R_1 = 500kbps, R_2 = 2Mbps e R_3 = 1Mbps, rispettivamente.

- **Assumendo che non vi sia altro traffico nella rete, qual è il throughput per il trasferimento del file?**
- **Se il file è di 4MByte, dividendo tale grandezza per il throughput, approssimativamente quanto tempo occorrerà per trasferire il file all'Host B?**
- **Rispondere alle precedenti domande supponendo R_2 = 100kbps**



8. Quali sono i cinque livelli della pila di protocolli Internet? Quali sono i loro principali compiti?

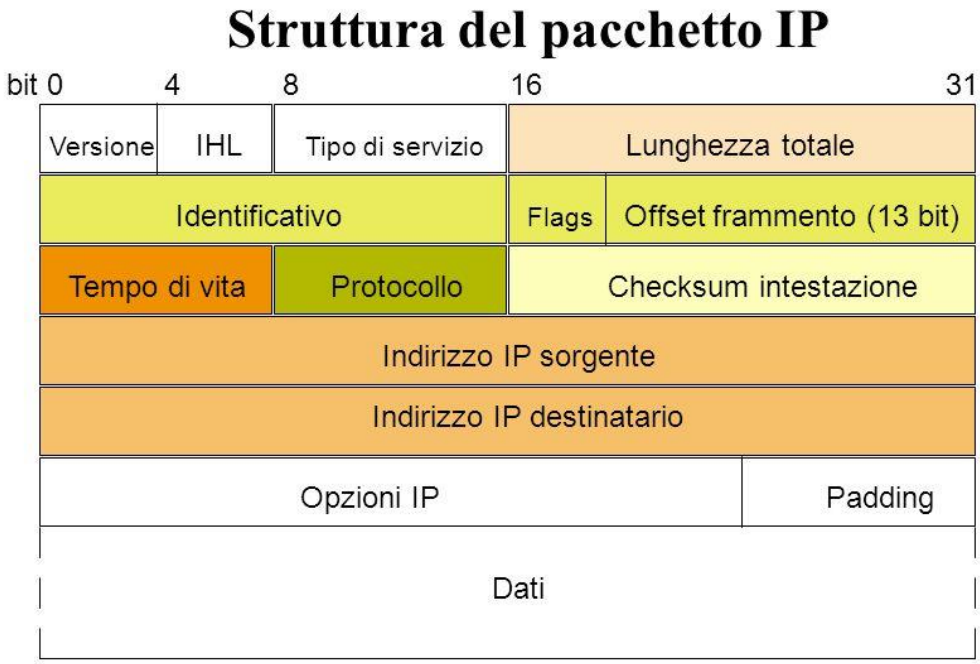
I cinque livelli della pila di protocolli Internet è formata da:

1. **Livello applicativo**: La sede delle applicazioni di rete, utilizza i protocolli HTTP, SMTP, FTP, DNS
2. **Livello di trasporto:** Trasferisce i messaggi del livello di applicazione, in questo livello si chiamano segmento. I principali protocolli sono TCP e UDP
    a. TCP: fornisce un servizio orientato alla connessione che include la garanzia della consegna dei messaggi a livello di applicazione alla destinazione e il controllo di flusso. Fornisce un meccanismo di controllo della congestione, un src regoli la propria velocità trasmissiva quando la rete è congestionata.
    b. UDP: fornisce un servizio senza connessione, senza affidabilità né controllo di flusso e di congestione
3. **Livello di rete:** Trasferisce i pacchetti, in questo livello chiamati datagrammi, a livello di rete da un host ad un altro. Il protocollo a livello di trasporto passa al livello sottostante un segmento e un indirizzo di destinazione. Comprende il protocollo IP che definisce i campi dei datagrammi e come i sistemi periferici e i router agiscono su tali campi.
4. **Livello di collegamento:** Il livello di rete di Internet instrada un datagramma attraverso una serie di router. I servizi forniti dal livello di collegamento dipendono dal protocollo utilizzato. Dato che i datagrammi in genere devono attraversare diversi

collegamenti, un datagramma potrebbe essere gestito da differenti protocolli a livello di collegamento lungo le diverse tratte che costituiscono il suo percorso

5. **Livello fisico:** Trasferisce i singoli bit del frame da un nodo a quello successivo

9. Spiegare i campi del datagram IP.



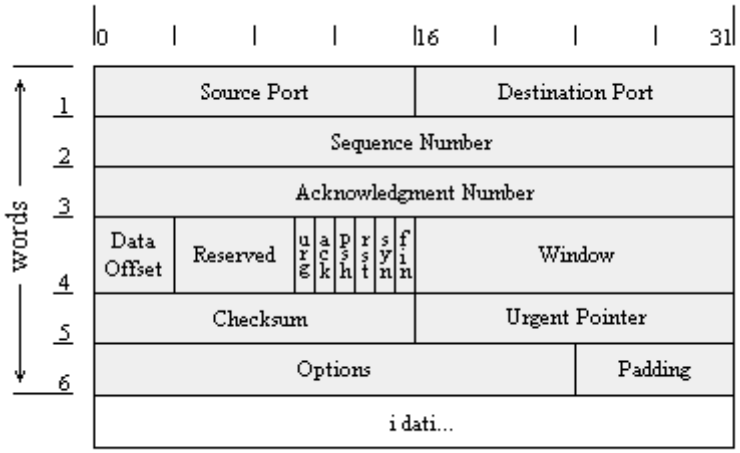
IHL = Internet Header Length
Identificativo/flags/offset di frammento = frammentazione di pacchetto
Tempo di vita = TTL (accoppiato con ICMP e traceroute)
Protocollo = TCP/UDP o altro

- **Versione:** 4 bit che specificano la versione del protocollo IP
- **Internet Header Length:** Dato che IPv4 può contenere un numero variabile di opzioni, questi 4 bit indicano dove iniziano effettivamente i dati del datagramma
- **Tipo di Servizio:** I bit relativi al tipo di servizio sono inclusi nell'intestazione IPv4 per distinguere i pacchetti
- **Lunghezza Totale(del datagramma):** Lunghezza totale del pacchetto IP, intestazione più dati, misurata in byte.
- **Identificativo (16bit, flags e offset fragment):** Servono per la frammentazione. In IPv6 non è consentita la frammentazione sui router
- **Tempo di vita(TTL):** Server per assicurare che i pacchetti non restino in circolazione per sempre nella rete. Questo campo viene decrementato di un'unità ogni volta che il datagramma è elaborato dal router. Quando raggiunge 0 il datagramma viene scartato.
- **(Upper layer)Protocollo:** Viene usato solo quando raggiunge la destinazione e il valore nel campo indica lo specifico protocollo a livello di trasporto al quale vanno passati i dati del pacchetto.
- **Checksum intestazione:** Consente ai router di rilevare gli errori sui bit dei datagrammi ricevuti, E' calcolato trattando ogni coppia di byte dell'intestazione come

numeri che sono poi sommati in complemento a1.

- **Indirizzo IP sorgente:** Contiene l'indirizzo IP dell'host mittente che ha inviato il pacchetto
- **Indirizzo IP destinazione:** Contiene l'indirizzo IP del server/host ricevente a cui deve essere recapitato l'indirizzo
- **Opzioni IP:** Serve per estendere l'intestazione IP. Le opzioni dell'intestazione sono state concepite per un utilizzo sporadico.
- **PADDING...**
- **Dati:** Qui si trova il payload del pacchetto di lunghezza variabile

10. Come è composto l'header di un pacchetto TCP e che compito svolgono i campi?



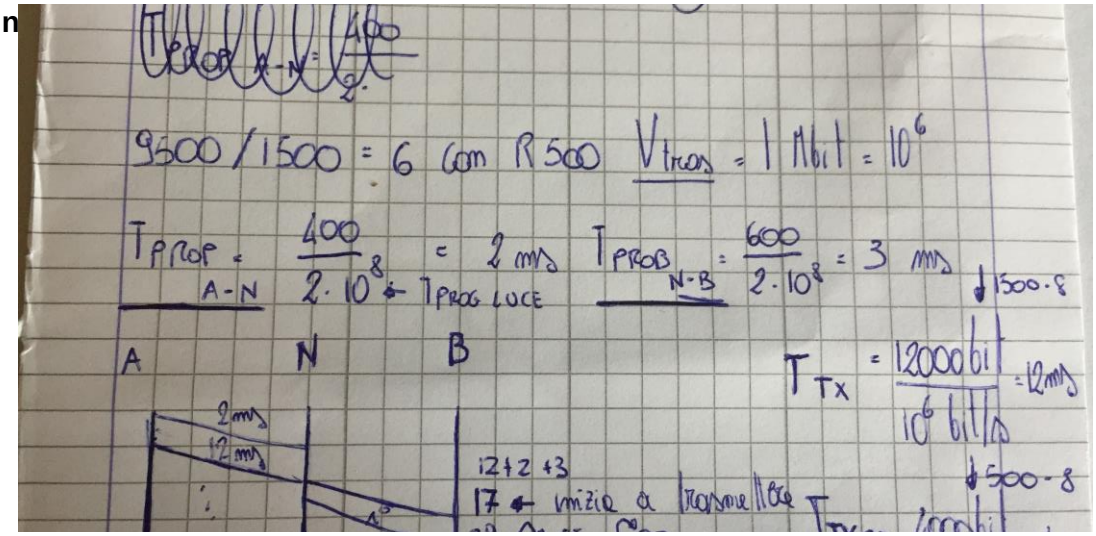
- **Porta sorgente:** Indica la porta sorgente del pacchetto TCP
- **Porta di destinazione:** Indica la porta di destinazione del pacchetto TCP
- **Numero di sequenza:** Serve per identificare il numero del pacchetto.
- **Numero Ack:** Serve per sapere il numero dell'ACK corrispondente al pacchetto
- **Data Offset/Head len:** Sono 4 bit che indicano la lunghezza dell'intestazione in multipli di 32 bit può variare in base al campo option che è variabile
- **Reserved/Not Used:** Serve nel caso di ulteriori implementazioni
- **U (Urgent Data)** Bit generalmente che non è usato
- **A (ACK)** E' un bit che mi serve per identificare se questo datagramma è un ACK
- **P (Push)** Serve per far si che venga messo a livello di applicazione (non usato molto)
- **R (Reset)**
- **S (Syn)**
- **F (Fin)** Sono i bit di controllo per il controllo di connessione(stabilire,l chiudere la connessione e fare un rest)
- **Receive Window:** Indica quanti bit sono disposto a ricevere
- **Checksum:** E' una somma bit a bit in complemento a 1 per determinare se un pacchetto è corrotto oppure no
- **Urgent data pointer:** E' un puntatore che indica dove iniziano i dati importanti e in genere non è usato

- **Options:** E’ di dimensione variabile e può mettere variabili opzionali
- **Padding: ...**
- **Dati:** Anche questo di lunghezza variabile

11 Si consideri una topologia di rete lineare composta da una sequenza di due canali con velocità di trasmissione pari a 1 Mbit/s, come illustrato in figura sotto. Il nodo S deve trasmettere un file di dimensione pari a 9500 byte verso il nodo D attraverso un nodo intermedio N che opera in modalità store-and-forward. Si ipotizzi che:

- **non vi siano errori di trasmissione;**
- **i tempi di elaborazione nel nodo N siano trascurabili, ma non il tempo per lo store-forward;**
- **il nodo N abbia capacità di memorizzazione infinita;**
- **la lunghezza del primo canale sia pari a 400 km, la lunghezza del secondo pari a 600 km;**
- **la dimensione massima dei pacchetti trasmessi sul canale sia pari a 1500 byte (si trascu-**
  rino le intestazioni).

Si determin



Capitolo 2

1. **Discutere, fare esempi e confrontare le architetture client-server e P2P**

- **P2P:** i nodi non sono gerarchizzati sotto forma di client e server ma sotto forma di nodi paritaria, peer, che possono quindi fungere sia da client che da server verso gli altri host della rete. Qualsiasi nodo è in grado di avviare o completare una transazione. Tra i vantaggi dell’utilizzo di questa architettura
  - Non è necessario un server con potenza elevata; **la capacità di servizio è distribuita**
  - **Elevata scalabilità, ovvero la possibilità di aggiungere capacità di servizio al sistema (più peer)**
  - **La velocità media di trasmissione è più elevata dal momento che**

l'informazione richiesta può essere reperita da diversi client connessi (peer) anzichè un unico server

- **Client Server:** In un'architettura **Client-Server** gli host di rete possono essere Client, se fruiscono un determinato servizio dalla rete o Server se forniscono tale servizio. **Il server resta in ascolto delle richieste di servizi da parte del client.** la comunicazione è avviata dunque dal client. Sia client che server dispongono di un indirizzo fisico, detto indirizzo IP che identifica, nella comunicazione, la sorgente e la destinazione. Quando un client e server iniziano a comunicare si possono scambiare pacchetti di controllo prima di spedire i dati effettivi(**handshaking**). Spesso nelle applicazioni Client-Server un singolo host server non è in grado di rispondere a tutte le richieste dei client. Per questo si usano i **data center** che ospitano molti host, che insieme creano un potente server virtuale

2. **Se una transizione tra client e server deve essere la più veloce possibile, cosa è meglio tra TCP e UDP (motivare la risposta)?**

   Se la transazione tra client e server dev'essere più veloce possibile è meglio utilizzare UDP, perché:

   - Senza connessione, non necessita di handshaking
   - servizio di trasferimento non affidabile: possibile perdite errori e ordine non corretto
   - Non esiste un controllo di congestione: un processo può inviare a qualunque velocità

   Anche se sembra che questo protocollo abbia solo difetti, nasce per l'esigenza di avere il massimo throughput possibile, ha così tanti difetti che perché utilizza un datagramma più snello perché non ha controlli e l'invio è più veloce

3. **Che cosa si intende per handshaking e spiegare contesto/motivazioni del suo uso.**

L'handshaking avviene quando un client deve connettersi ad un server, utilizzando il protocollo TCP, la prima azione che deve eseguire per effettuare la connessione è proprio l'handshaking che serve per stabilire una connessione tra di loro.

1. **Il client invia un messaggio al server**, **dove viene impostato il campo sync del pacchetto a 1, ack a 0 e viene messo il numero di Initial sequence number** (per esempio 2000), **il quale marca l'inizio della sequenza di numeri per i dati** che il client andrà a trasmettere
2. **Il server, se riceve il pacchetto, invia il client un pacchetto che contiene a sua volta il campo sync a 1, ack a 1, ISN impostato a un valore** (esempio 5000) e il **ack number = ISN+1** (nel nostro caso 2001)
3. **Il client**, una volta ricevuto il messaggio dal server, **invia a sua volta un altro messaggio, il quale dopo esso potrà iniziare la vera e propria comunicazione tra di loro.** Il messaggio **conterrà il campo SYN a 0, ACK =1, Sequence number = 2001 e il ack number = 5000+1**

Questa tecnica si riferisci a TCP three-way handshake or SYN, SYN-ACK, ACK.

Dopo questo three-way handshake, la connessione è aperta e i partecipanti iniziano a mandare dati usando il numero di seuqenza e l'ack

4. **Si supponga che Alice debba mandare un messaggio di posta elettronica a Bob e che quest'ultimo scarichi la posta con il POP3. Descrivere il "viaggio" di questo messaggio dettagliando i protocolli coinvolti.**
5. **Che differenza c'e' tra POP3 e IMAP?**

E' un protocollo che veniva usato prima dell'avvento del IMAP, quando ancora non erano presenti tutti questi dispositivi connessi al Internet.

POP3 entra in azione quando lo user agent apre una connessione TCP verso il mail server sulla porta 110. Tre fasi:

1. ***Autorizzazione:*** lo user agent invia nome utente e password in chiaro per autenticare l'utente, utilizzando lo user e la password
2. ***Transazione*** lo user agent recupera i messaggi e può marcarli per la cancellazione e ottenere statiche sulla posta
3. ***Aggiornamento*** dopo che il client ha inviato il comando QUIT, il server rimuove i messaggi marcati per la cancellazione.

Uno user agent può essere configurato per " scaricare e cancellare " o per " scaricare e mantenere" , nella prima modalità il destinatario potrebbe voler accedere ai propri messaggi di posta da più macchine, se l'utente legge un messaggio su un dispositivo non sarà in grado di leggerlo da un dispositivo diverso, mentre in modalità " scarica e mantieni" lo user agent lascia i messaggi sul server di posta, dopo averli scaricati saranno disponibili su un qualsiasi dispositivo.

Durante una sessione tra uno user agent e il mail server, il server POP3 mantiene traccia dei messaggi dell'utente marcati come cancellati. Non trasporta informazioni di stato tra le diverse connessioni

IMAP

Un server IMAP associa a una cartella ogni messaggio arrivato sul server, I messaggi in arrivo sono associati alla cartella INBOX. IMAP fornisce comandi per consentire agli utenti di creare cartelle e spostare i messaggi da una cartella a un'altra, consente agli utenti di effettuare ricerche nelle cartelle e permette agli user agent di ottenere singole parti dei messaggi. I server IMAP conversano informazioni di stato sull'utente da un sessione a all'altra.

Lo user agent può anche essere un browser web e l'utente comunica con la propria casella remota via HTTP. Il messaggio e-mail viene spedito dal server di posto al browser usando il

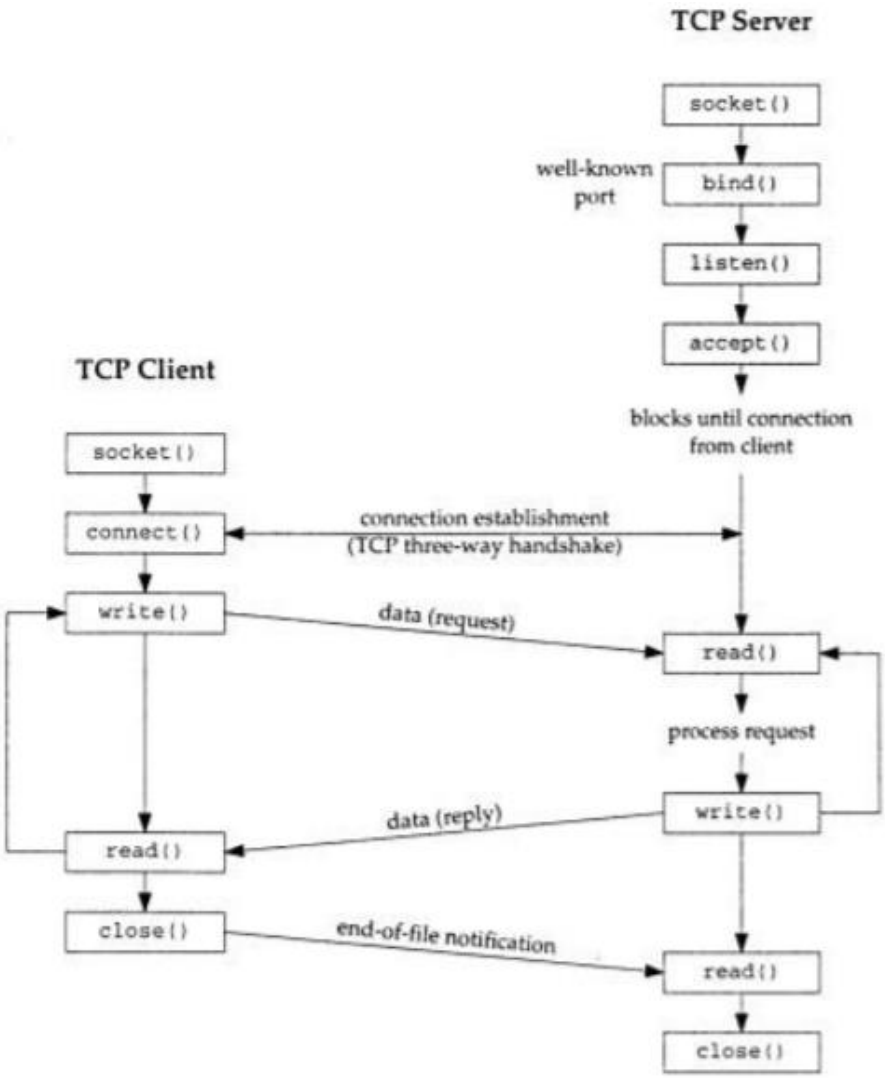
protocollo HTTP

DIFFERENZE FRA POP3 E IMAP

La differenza principale quindi è che nel POP3, era utilizzato quando non c'erano molti dispositivi connessi a Internet e difficilmente qualcuno aveva più dispositivi connessi, infatti una volta che un'email veniva scaricata su un dispositivi non era più possibile scaricarla da altri, e le email inviate non potevano essere visualizzate se non dallo stesso dispositivo da cui venivano inviate e dal ricevente.

All'inizio l'utilizzo di IMAP era a pagamento, solo dopo qualche tempo è stato reso gratis, infatti esso è un'evoluzione del POP3 perchè mantiene tutti i messaggi in un unico posto ovvero il server ed è possibile consultarli anche da diversi dispositivi. Permette anche di organizzare i messaggi in cartelle e di mantenere le cartelle della mailbox sincronizzate

6. **Descrivere la sequenza di system call sia per il client, sia per il server, per il protocollo TCP.**



La prima azione per fare dell'I/O da rete è la chiamata alla funzione **socket()** specificando il tipo di protocollo di comunicazione da utilizzare.

Int socket (int family, int type, int protocol); restituisce un descrittore di socket maggiore o uguale a zero, oppure -1 in caso di errore, e setta errno.

La funzione **connect()** è usata dal client TCP per stabilire la connessione con un server TCP.

La funzione **bind()** collega al socket un indirizzo locale. Per TCP e UDP ciò significa assegnare un indirizzo IP ed una porta a 16-bit.

La funzione **listen()** è **chiamata solo dal TCP server** e esegue due azioni: 1) **ordina al kernel di far passare il socket dallo stato iniziale CLOSED allo stato LISTEN,** e di **accettare**

richieste di inizio connessione per quel socket, accodandole in delle code del kernel. 2) specifica al kernel **quante richieste di inizio connessione può accodare al massimo** per quel socket.

La funzione **accept()** è chiamata **solo dal TCP server e restituisce la prima entry nella coda delle connessioni già completate per quel socket.** Se la coda è vuota la accept resta in attesa.

La funzione **close()** è utilizzata normalmente per chiudere un descrittore di file, è utilizzata per chiudere un socket e terminare una connessione TCP.

Client TCP

La System Call della **socket**, noi dobbiamo decidere se la famiglia di indirizzi è IPv4 o IPv6. IPv6 è il nuovo tipo di indirizzo che permette di indirizzare un numero maggiore di dispositivi rispetto a IPv4. L'altro aspetto che bisogna capire è il tipo di servizio (TCP o UDP) e infine ci sono una serie di opzioni per definire il protocollo.

Ci sono una sequenza di opzioni che possiamo mettere nella System Call per definire le famiglie di indirizzi (useremo AF_INET).

Il socket type potrà essere di diversi tipi ma useremo (SOCK_STREAM - tcp o SOCK_DGRAM-udp)

Una chiamata socket sarà chiamata in questo modo:

```
mySocket = socket(AF_INET,SOCK_STREAM,0)
```

(se mettiamo 0 userà di default IP)

Connect()

Per connetterci al server,

```
connect(simpleSocket, (strcut sockaddr*) &simple-Server, sizeof(simpleServer))
```

Vuole in ingresso il socket la struttura dati che rappresenta la connessione e la sua dimensione. Poi comincia un ciclo **read e write** per scrivere al server o leggere ciò che arriva al server.

```
read(simpleSocket, buffer,sizeof(buffer))
```

```
write(simpleSocket,stringbuff,strlen(stringbuff))
```

Si scrive o si legge dal socket, serve un buffer per leggere e scrivere attraverso il socket e la sua dimensione, se la read o lo write hanno restituito qualcosa <= 0 vuol dire che c'è un errore, in quanto la read e la write restituiscono la quantità di byte letti o scritti. Una volta finito il ciclo si può effettuare una **close().**

Server TCP

All'inizio c'è una socket e poi c'è una bind.

```
bind(simpleSocket, (struct sockaddr *) & simpleServer, sizeof(simpleServer));
```

la bind ha quel tipo di interfaccia. Bisogna controllare cosa restituisce per capire se ci sono degli errori(quell'indirizzo o quella porta erano già stati utilizzati). La funzione per l'assegnazione di un indirizzo ad un socket è *bind()* con la quale se si assegna un indirizzo locale ad un socket si guarda che sia diverso da zero se è diverso da 0 è un errore se no è uguale la bind è andata a buon fine.

```
listen(simpleSocket,5);
```

Quando il client invia un SYN, se tutto va bene viene restituito un SYS e se tutto va bene il client invia l'ACK. Finchè non si completa questa sequenza di 3 scambi si rimane in una situazione incompleta, il SYN_FLOOD server quando si vuole attaccare un server e riempire quella struttura dati "SYN_RCVD".

LISTEN(5) significa che la struttura dati non può supportare più di 5 connessioni pendenti. Cioè non più di 5 connessioni **che non hanno completato** il three way and shake. Se da un risultato = a -1 c'è un errore.

ACCEPT

```
accept(SimpleSocket, (structsockAddr*)&clientName, &clientNameLength);
```

Serve per accettarmi la connect fatta da client. A questo punto il server è pronto per accettarmi delle connessioni dal client. Parte un ciclo while, fa un po di inizializzazioni e poi un accept. **Rimane bloccante finchè qualche client non fa una connect.** I parametri sono sempre il socket e gli altri parametri sono con la & quindi sono dei parametri che vengono inizializzati quando scatta l'accept.

Posso fare anche un accept control, se io so che un indirizzo IP è uno spammer, posso già rifiutare qui di andare avanti chiedendogli la connessione.

Questo è un blocco a livello applicativo che può essere fatto anche a livello di firewall. Se siamo i gestori di una rete locale possiamo decidere che non si arrivi nemmeno a fare SYN, se ci fosse un firewall si potrebbe bloccare prima che faccia la SYN. Abbiamo poi il controllo se tutto è andato bene e si verifica guardando se ci viene restituito un valore diverso da -1.

Una cosa importante è che il SimpleSocket è un socket per accettare nuove richieste, una

volta che un client è connesso, verrà restituito un SimpleChildSocket e questo ci verrà utile quando faremo una fork.

WRITE READ E CLOSE

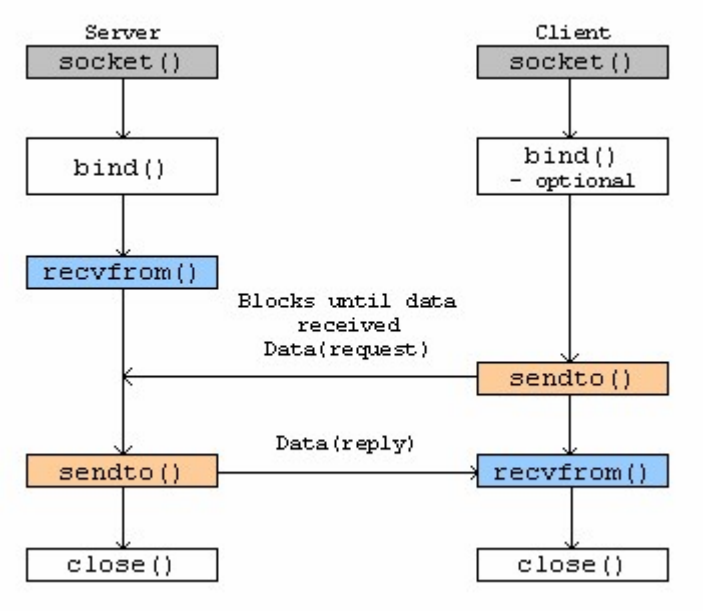
Abbiamo settato tutto e client è stato accettato, faremo quindi una serie di write e read e infine una close. Una volta fatta la close il server ritorna a rimanere in attesa di altre connessioni. (Il server continua a rimanere in ascolto)

La read restituisce il n. di byte letti

La write restituisce il n. di byte scritti

E' molto comodo quando si definisce un protocollo, una lunghezza di stringhe standard, se ci aspettiamo 50 byte è molto più comodo perchè nella read posso controllare se ha letto 50 byte. Se non ha letto 50 byte o qualcosa non è andata bene o il nostro server non è riuscito a spedire tutto.

7. Descrivere la sequenza di system call sia per il client, sia per il server, per il protocollo UDP.



La prima azione per fare dell'I/O da rete è la chiamata alla funziona **socket()** specificando il tipo di protocollo di comunicazione da utilizzare.

Int socket (int family, int type, int protocol); restituisce un descrittore di socket maggiore o uguale a zero, oppure -1 in caso di errore, e setta errno.

La funzione **connect()** è usata dal client TCP per stabilire la connessione con un server TCP.

La funzione **bind()** collega al socket un indirizzo locale. Per TCP e UDP ciò significa assegnare un indirizzo IP ed una porta a 16-bit.

La funzione **listen()** è chiamata solo dal TCP server e esegue due azioni: 1) ordina al kernel di far passare il socket dallo stato iniziale CLOSED allo stato LISTEN, e di accettare richieste di inizio connessione per quel socket, accodandole in delle code del kernel. 2) specifica al kernel quante richieste di inizio connessione può accodare al massimo per quel socket.

La funzione **close()** è utilizzata normalmente per chiudere un descrittore di file, è utilizzata per chiudere un socket e terminare una connessione TCP.

CLIENT UDP

La System Call della **socket**, noi dobbiamo decidere se la famiglia di indirizzi è IPv4 o IPv6. IPv6 è il nuovo tipo di indirizzo che permette di indirizzare un numero maggiore di dispositivi rispetto a IPv4. L'altro aspetto che bisogna capire è il tipo di servizio (TCP o UDP) e infine ci sono una serie di opzioni per definire il protocollo.

Ci sono una sequenza di opzioni che possiamo mettere nella System Call per definire le famiglie di indirizzi (useremo AF_NET).

Il socket type potrà essere di diversi tipi ma useremo (SOCK_STREAM - tcp o SOCK_DGRAM-udp)

Una chiamata socket sarà chiamata in questo modo:

```
mySocket = socket(AF_INET,SOCK_STREAM,0)
```

(se mettiamo 0 userà di default IP)

E' più semplice rispetto a un CLIENT TCP perchè non si fa la connect, si fa solo una socket e poi una serie di sendTo() e rcvFrom() e infine una close per chiudere una connessione

SERVER UDP

Non fa più listen e accept. Socket bind recvFrom e sendTo() e infine close.

Differenza tra send, write e read recvfrom:

Dato che facendo una connect nel protocollo TCP so già dove scrivere o leggere faccio semplicemente una write o una read, nel protocollo UDP dato che non faccio una connect devo specificare oltre a cose scrivere dove lo voglio scrivere/o leggere.

```
recvfrom(simpleSocket, buffer, sizeof(buffer),0,(struct sockAddr *)
& senderAddr, & senderSize);
```

```
sendTo(simpleSocket, string, strlen(string),0,(struct
sockAddr*)&destAddr, destSize);
```

La sendTo ha il buffer, la lunghezza del buffer, lo 0 ha una serie di opzioni che noi non vedremo poi abbiamo la destinazione e la struttura dati che contiene la destinazioni. **Quindi la sendTo() non ha solo socket e buffer ma specifica anche indirizzo e porta di quello a cui vuole inviare ciò che è nel buffer.**

recVfrom() possiamo salvare chi ci ha contattato perché in UDP non lo possiamo sapere a priori. Diciamo che la struttura di systemCall presa in considerazione è più leggera rispetto a TCP.

8. Spiegare come funziona la select per il multiplexing

Quando devo gestire più client attraverso un multiplexing (un'alternativa al multiplexing è la

fork) si usa la select, la select mi indica quale socket descriptor ha avuto attività.

9. **Spiegare i parametri e come funzionano le system call read/write**

```
read(simpleSocket, buffer, sizeof(buffer))
```

Leggo dal socket e lo mette sul buffer indicato

```
write(simpleSocket,stringbuff,strlen(stringbuff));
```

Inserisce ciò che ho scritto sul buffer nel socket.

10. **Discutere vantaggi e svantaggi dell'approccio concorrente (fork) rispetto a quello multiplexing (select)**

Fork è concorrente appunto perchè vengono usati dei processi figli mentri il multiplexing no perchè c'è un unico processo.

Multiplexing ha un lungo tempo di attesa perché devo:

Fare la **copia dell'array, fare la select, gestire il client e tornare su e riiniziare da capo** per gestirli tutti.

Mentre **la fork non viene fatta in parallelo con la creazione dei processi figli quindi breve tempo di attesa.**

La select però sulla singola locazione ha un tempo di esecuzione più breve perché con la fork devo aspettare lo scheduler.

Multiplexing ha job ordinati mentre con la fork no.

Multiplexing non ha nessuna copia di dati mentre la fork si.

Non ci sono errori sui processi per il multiplexing

11. **Quali tecniche si possono usare per rendere un'applicazione di rete tollerante ai guasti?**

La scelta di queste opzioni **si sceglie in base all'affidabilità** (per più affidabilità si va verso la directory server anche se sarò più lento = se invece voglio velocità nel servizio allora si usa anche solo un log sul server. **La copia ha senso farla in base all'entità che sta facendo più lavoro- Tenendo contro che le copie devono essere sincronizzate e devo capire dove devo andare se da quello principale o dalla copia.**

Log del server:

Server di backup:

Database duplicato:

Directory server: Single point of failure. C' una directory server poi ci sono più server che possono anche essere duplicate, e più client c'è un database che può essere anche duplicato. A decidere chi deve fare cose e dove lo sceglie il directory server.

Attivo: Il directory server chiede lui al server o al database

Passivo: Il directory server lavora passivamente, se mi arriva la richiesta dal client di parlare con il server di directory server gli risponde con un indirizzo ip e porta di quel server ma poi sarà il client ad interfacciarsi con il server e se ne accorgerà lui e è morto.

Il directory server dice solo se un server c'è oppure no.

Se poco dopo mi arriva la richiesta da un client per lo stesso server (che era morto) posso fare una sorte di differenza e dirgli che il server è morto.

12. Come funziona l'applicativo traceroute? Che valori restituisce?

Traceroute è un programma che mi consente di tracciare il percorso fatto dai pacchetti su una rete. Traceroute sfrutta il TTL, Time-To-Live.

Un'applicazione traceroute, quindi, invia un pacchetto al destinatario di cui si vuole ricavare il percorso di traceroute con il campo TTL impostato ad 1. Il primo router che lo riceverà, constatando che il campo TTL ha raggiunto lo 0, invierà un errore al mittente (ICMP Time Exceeded). L'applicazione memorizzerà l'indirizzo IP del primo router, quindi invierà un nuovo pacchetto con TTL impostato a 2. L'operazione verrà ripetuta finché il pacchetto non sarà arrivato al destinatario, che invierà un ICMP Echo Reply.

Alla fine l'applicazione avrà ottenuto la lista degli indirizzi IP dei router su cui hanno transitato i pacchetti.

13. Commentare il seguente codice:

1: rset=allset; //rset: array di bit

//la select si usa perchè stabilisce quale socket è attivo.

2: if(ready=**select**(maxd+1,&rset,NULL,NULL,NULL)<0) {perror("select problem")};
/*

Se c'è qualcosa di pronto in lettura nel socket del client vuol dire che il client ci ha parlato. Per esempio, ci ha chiesto di calcolare il massimo. Gli rset che vengono passati alla select vengono passati per riferimento (infatti c'è la &) il che vuol dire che se la passo alla select e la select scatta può succedere che l'array può passare a una situazione del genere [0,0,0,1] perché viene passata da riferimento e la select l'ha modificato, se l'ha modificato così vuol dire che io ero in ascolto su 2 socket descriptor ma solo uno è diventato pronto in modalità ascolto.

```
*/
3:                  if(FD_ISSET(sockfd,&rset))    {

//tramite la FD-ISSET guardo quali di questi bit ha avuto attività


4:                                  sin_size = sizeof(their_addr);
5:                                  new_fd = accept(sockfd, (struct sockaddr
*)&their_addr, &sin_size);
6:                                  if(write(new_fd,"Riprovare più tardi\n",strlen("Riprovare
più tardi\n")) == -1) perror("write");
7:                              }
```

14. Di seguito è riportato il contenuto (in codifica testuale ASCII) di una richiesta HTTP. Rispondere alle domande seguenti indicando dove trovate la risposta nella richiesta HTTP.

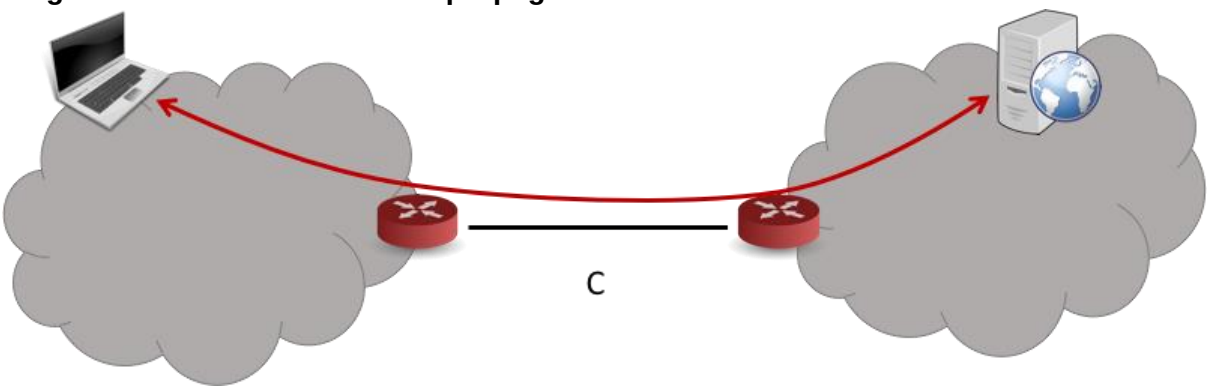
```
GET /cesana/index.html HTTP/1.1
Host: home.deib.polimi.it
Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/124 (KHTML,
like Gecko) Safari/125
Accept: ext/xml, application/xml, application/xhtml+xml,
text/html;q=0.9, text/plain;q=0.8, image/png,*,*;q=0.5
Accept-Language: ita
Keep-Alive: 300
Connection: keep-alive
```

a)Quale è la URL richiesta? *home.deib.polimi.it*`/cesana/index.html`
b)Quale versione di HTTP è usata? `1.1`
c)Il browser richiede una connessione non persistente o persistente? `keep-alive`
d)A cosa serve l'indicazione del tipo di browser nel messaggio di richiesta

L'indicazione del tipo di browser nel messaggio di richiesta indica da quale tipo viene visualizzata la pagina web, questo è necessario per capire come far visualizzare la pagina.

15. Un client HTTP richiede ad un server HTTP una pagina web costituita da un oggetto base (file HTML) e 10 altri oggetti. Ogni oggetto ha una dimensione L=200 kbit. Il collegamento tra client e server HTTP è in grado di trasferire informazione ad una velocità di C=100 kb/s in entrambe le direzioni. I messaggi di controllo usati per

aprire una connessione TCP tra client e server ed il messaggio di GET HTTP hanno lunghezza l=100bit. Il ritardo di propagazione sia trascurabile.



Calcolare in tempo totale per ricevere interamente la pagina web richiesta nei due casi seguenti:
a)il client HTTP apre in parallelo in modalità non persistente tutte le connessioni TCP necessarie per scaricare la pagina web
(si assuma che il data rate della singola connessione sia r=C/N, con C data rate del collegamento e N numero di connessione aperte in parallelo)
b)il client HTTP apre un'unica connessione TCP persistente per scaricare tutti gli oggetti della pagina web.
c)Il client HTTP apre in serie 11 connessioni TCP in modalità non persistente
Nel caso a) il client funziona in modo non persistente. Il client apre una connessione TCP non persistente per richiedere il file HTML (oggetto base) della pagina web. Ricevuto il file HTML chiude la connessione, "legge" il file HTML, scopre che la pagina web è costituita da 10 altri oggetti ed apre quindi 10 connessioni TCP non persistenti per scaricare ciascuno dei 10 oggetti. Il tempo richiesto per ottenere il file HTML (oggetto base) è:

dove il primo termine rappresenta il tempo per aprire la connessione TCP, il secondo termine il tempo per inviare al server il messaggio di GET HTTP ed il terzo termine il tempo per scaricare il file HTML. Una volta ottenuto il file HTML, il client apre in parallelo 10 connessioni TCP che condividono lo stesso collegamento. Il tempo per ottenere ciascuno dei 10 oggetti attraverso le connessioni TCP, ciascuna delle quali necessita di una fase di apertura, è
Il tempo complessivo sarà quindi dato da:
Sostituendo i valori di l, L, C e r nelle formule, si ottiene:

e quindi

Nel caso b) il client apre una connessione persistente per scaricare in serie tutti gli 11 oggetti che comprendono la pagina web:

dove il primo termine rappresenta il tempo per aprire la connessione TCP, ed il secondo termine rappresenta il tempo per inviare la richiesta GET HTTP e ricevere il singolo oggetto per tutti gli 11 oggetti della pagina web. Sostituendo i valori nella formula si ottiene:

Nel caso c) il client apre 11 connessioni TCP in serie in modalità non persistente. Si ha quindi:

da cui si ottiene:

Capitolo 3

Supponete che un server web sia in esecuzione sull'host C sulla porta 80. Supponete che questo web server usi le connessioni persistenti stia al momento ricevendo richieste da diversi host, A e B. Tutte le richieste vengono inviate attraverso la stessa socket? Se vengono fatte passare attraverso socket diverse, entrambe hanno la porta 80? Discutete e spiegate questa soluzione

Anche se vengono fatte passare tramite socket diverso la porta a lato server è sempre la 80, i client creano una socket con il loro indirizzo IP su un porta sopra la 1024, dopo di che inviano una richiesta sulla porta 80 del server, il quale effettuerà una fork per creare un processo figlio che gestisca le richieste di quel client.
La connessione viene identificata dall'end-point, ovvero IP+PortaClient e IP+PortaServer. Quando arriva la richiesta dal client, il server riesce a riconoscere se è una richiesta nuova o vecchia in quanto ha salvato la corrispondenza tra il server figlio e il client da cui riceve le richieste quindi riesce a capire se deve creare un nuovo processo o inoltrare la richiesta al processo figlio.

Esempio:
Se un client apre più finestre nel browser sullo stesso sito, il server corrispondente creerà una entries nella tabella dei SD (socket descriptor) a cui associa quel end-point. Con lo stesso schema può gestire richieste da client distinti.

Protocollo rtd
Perchè in questi protocolli abbiamo introdotto

- **Il numero di sequenza?**
- **Il timer?**

Motivare le risposte.

Nel protocollo RTD sono stati introdotti **il numero di sequenza perché nel caso in cui il pacchetto viaggi su un canale non sicuro dove è possibile che un insieme di pacchetti arrivi a destinazione in ordine diverso rispetto quello a cui sono stati inviati** o il client/ server riceve ACK/pacchetti duplicati, è stato aggiunto il numero di sequenza per far in modo che ogni pacchetto inviati venga identificato con esso, così che il server possa mandare un ACK identificatore per quel pacchetto tramite il numero di sequenza, ottenendo così la ritrasmissione di solo quel pacchetto.
Il timer invece è stato introdotto nel caso in cui il pacchetto non giunga mai a destinazione o anche per il controllo di flusso. Quando un client invia un pacchetto, fa iniziare il timer per esso, nel caso in cui superi l'EstimetedTime, il client invia un altro pacchetto uguale. Da qui si può intuire che la rete è congestionata perchè il pacchetto non arriva a destinazione e il client non riceve ACK

Checksum
Supponendo che il ricevente UDP calcoli il checksum per il segmento UDP ricevuto e trovi che corrisponda al valore trasportato nel campo checksum. Può il ricevente essere assolutamente certo che non vi siano stati errori sui bit?

Motivare la risposta.

Il ricevente non può essere sicuro che non vi siano stati errori sul bit perchè il protocollo UDP non offre nessun controllo degli errori, ma si limita solamente a scartare il pacchetto nel caso a cui facendo il calcolo del checksum succede che non tutti i bit sono a 1. Una volta scartato il client non è a conoscenza di ciò e non avviene nessuna ritrasmissione.

ESEMPIO:
1010+
1111

0001+
1

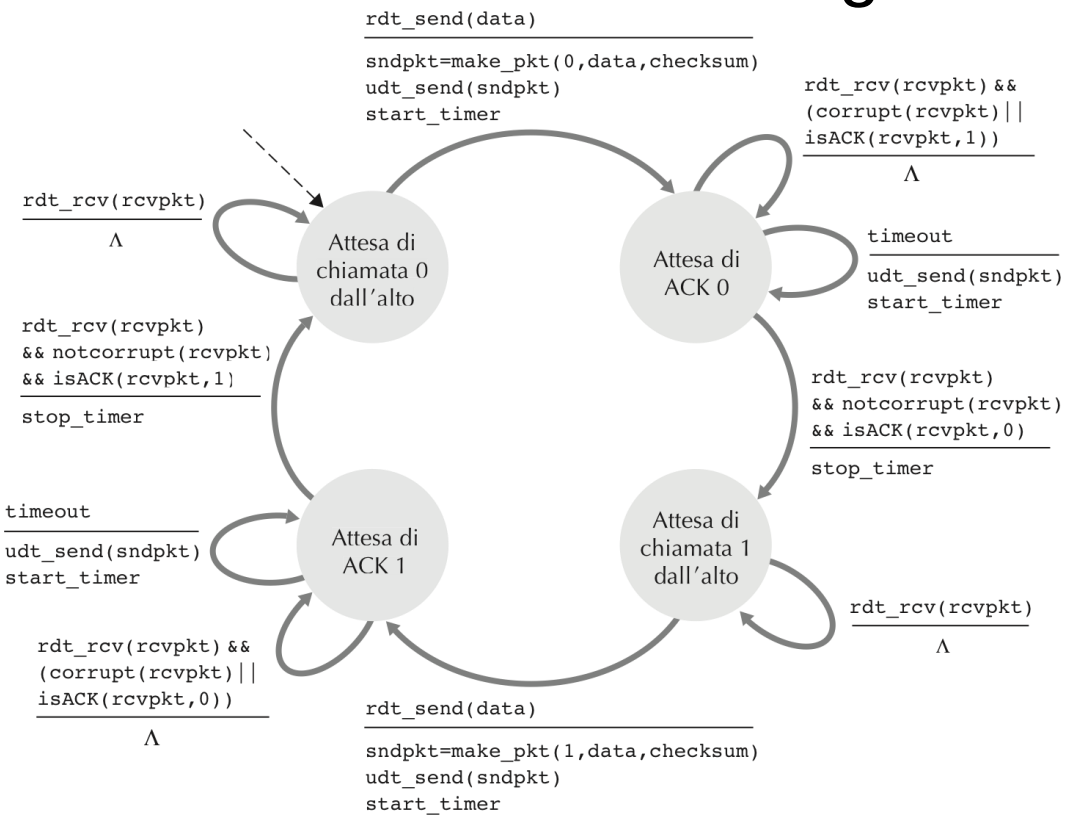
0010 ←RISULTATO DELLA SOMMA
1101 ← COMPLEMENTO A 1 DELLA SOMMA

Se io avessi fatto:
1111+
1010

0001+
1

0010 ←RISULTATO DELLA SOMMA
1101 ← COMPLEMENTO A 1 DELLA SOMMA
I due checksum sono uguali ma I dati sono diversi.

Rtd 3.0: commentare la figura



Questo caso si applica quando il canale di trasmissione, potrebbe danneggiare i bit e smarrire i pacchetti. (RTD 2.2?)

Qui viene rappresentato un client che vuole inviare un pacchetto ad un server, il tutto inizia con la chiamata **rtd_send(data) che si occupa di prelevare i pacchetti a livello di applicazione, impacchettarli e inviarli sul canale** per trasportarli verso il server.
Quindi ora si mette in attesa di un ACK, il quale potrà essere positivo , quindi qualcosa è andato storto e i bit sono stati corrotti, quindi avviene una ritrasmissione del pacchetto oppure negativo quindi non dev'essere ritrasmesso, anche se non si ha garanzia che a sua volta il pacchetto ACK sia stato alterato durante la ritrasmissione.
L'innovazione principale di questo protocollo però è la tolleranza alla perdita di pacchetti, aggiungendo un **timer: ogni volta che viene inviato un pacchetto viene iniziato un timer e dopo un tot di tempo se non è stato ricevuto un ACK questi viene ritrasmesso**. Questo però comporta che ci siano più pacchetti duplicati in circolazione e potrebbero arrivare tutti e destinazione provocando così ACK duplicati.

UDP e correttezza

Come è possibile garantire la correttezza nell'invio dei dati, pur usando un protocollo UDP?
Perchè alcune volte si opta per questa scelta, pur avendo TCP come protocollo affidabile?
La correttezza a livello UDP viene garantita a livello applicativo, ad esempio si può fare un controllo a livello di indirizzi IP per fare controlli sulle sorgenti o eventualmente si può implementare il checksum che non è parte integrante del pacchetto UDP.
Nelle applicazioni di streaming, come Netflix, viene usato UDP perchè non è necessario che tutti i pacchetti arrivino a destinazione ma che la connessione sia utilizzabile e il più leggera possibile, TCP garantisce la connessione ma è molto più pesante rispetto a UDP.

SampleRTT e EstimatedRTT

Supponete che i 5 valori misurati di SampleRTT siano 106 ms, 120ms, 140ms, 90ms e 115ms. Calcolate EstimatedRTT dopo l'acquisizione di ogni valore di SampleRTT, usando un valore α=0,125 e assumendo che il valore EstimatedRTT appena prima dell'acquisizione del primo di questi cinque campioni fosse 100ms. Calcolate anche DevRTT dopo l'acquisizione di ogni campione, assumendo β=0,25 e che il valore di DevRTT appena prima dell'acquisizione del primo di questi 5 campioni fosse 5 ms. Infine calcolate il valore di TCP TimeoutInterval dopo l'acquisizione di ogni campione.

Velocità trasferimento

Si calcoli, in modo approssimato, la massima velocità di trasferimento, in bit/s, di un file di dimensione pari 1 Gbyte tra un nodo localizzato in Italia e un nodo localizzato in California. Si ipotizzi che:

- **la finestra di trasmissione sia pari a 10 kbyte;**

- **i ritardi di accodamento, di elaborazione e di store-and-forward negli apparati di commutazione intermedi siano trascurabili;**
- **la distanza tra i due nodi sia pari a 10000 km.**

Tempo trasferimento file

Si consideri una topologia di rete lineare composta da un singolo canale con velocità di trasmissione pari a 1 Mbit/s. Il nodo sorgente deve trasmettere un file di 9500 byte verso il nodo destinazione. Si ipotizzi che:
- la rete sia scarica e non vi siano errori di trasmissione;
- il tempo di propagazione sul canale sia pari a 5 ms;
- la dimensione massima dei pacchetti trasmessi sul canale, comprendenti 40 byte di intestazione, sia di 1500 byte;
- si utilizzi un protocollo Stop & Wait;
- i pacchetti di riscontro (ACK) abbiano dimensione trascurabile.
Si determini il tempo necessario affinché la destinazione riceva completamente il file corretto.

138 **Esercizio A-18 – Soluzione-1**

File da trasmettere byte
Frame = byte (40 di intestazione) >> dati 1460 byte
Numero di frame da trasmettere 9500 / 1460 = 6,5 circa
6 frame-A da 1500 (per un totale di 8760 byte di dato)
1 frame-B da 780 (per un totale di 740 byte di dato)
Tempo tx frame-A [bit] / 106 [bit/s] = s = 12 ms
Tempo tx frame-B [bit] / 106 [bit/s] = = 6.2 ms
Tempo di Propagazione = 5 ms

139 **Esercizio A-18 – Soluzione-2**

5 ms
17ms = 5ms +12ms
22ms=17ms +5ms

140 **Esercizio A-18 – Soluzione-3**

Ciclo trasmissione Frame
5 ms + 12 ms + 5ms = 22ms
5 ms tempo Propagazione
12 ms tempo Trasmissione
5 ms tempo propagazione ACK
6 Frame = 6 * 22 = 132ms
Frame residuo = 5ms + 6.2ms = 11.2 ms
Tempo Totale di Ricezione = ms

Trasmissione file (Go-Back-N)

Si consideri una topologia di rete lineare composta da un singolo canale con velocità di trasmissione pari a 1 Mbit/s. Il nodo sorgente S deve trasmettere un file di dimensione pari a 12000 byte verso il nodo destinazione D. Si supponga che:

- **la rete sia scarica e non vi siano errori di trasmissione;**
- **il tempo di propagazione sul canale sia pari a 20 ms;**
- **la dimensione massima dei pacchetti trasmessi sul canale, compresi 40 byte di intestazione, sia di 1500 byte;**
- **si utilizzi un protocollo Go-Back-N con finestra di trasmissione $W_T = 3$ pacchetti di dimensione massima;**
- **la dimensione dei pacchetti di ACK sia trascurabile.**

Trasmissione file (Go-Back-N) - II

Si determini:

- **il tempo T_R necessario affinché il nodo D riceva completamente il file;**
- **il valore minimo di W_T che permette di ridurre T_R al minimo;**

Capitolo 4

Il server DHCP serve per ottenere la maschera di rete, il router di default (gateway), un indirizzo IP e quello del DNS. Spiegare il formato e significato di ogni valore.

Il server **DHCP (DInamic Host Configuration Protocol) è un protocollo di rete a livello applicativo che permette ai dispositivi o terminali di una certa rete locale di ricevere automaticamente ad ogni richiesta di accesso a una rete IP la configurazione IP necessaria per stabilire una connessione e operare su una rete più ampia** basata su Internet Protocol, cioè **interpretare con tutte le altre sottoreti scambiandosi dati, purchè anch'esse integrate lo stesso modo con il protocollo IP.** Il protocollo è implementato con servizio di rete.

DHCP:

1. Host invia in broadcast un messaggio, DHCP DISCOVER
2. DHCP server risponde con DHCP OFFER
3. Host richiede l'indirizzo ip: DHCP REQUEST
4. DHCP server invia l'indirizzo: DHCP ACK

Per ottenere la maschera di rete, mi basterà sapere quanti host possono esserci all'interno della rete stessa. (Ci saranno tutti 1 a parte dove ci sono gli host, generalmente nelle reti locali ci sono 356 host quindi la maschera è 255.255.255.0)

Il router di default è il gateway, ovvero il nostro punto di uscita, nelle reti domestiche è il primo indirizzo ip disponibile (se la rete è 192.168.0.0 il gateway sarà 192.168.0.1)

Un indirizzo IP è una sequenza di 32 è un'etichetta numerica che identifica univocamente un dispositivo detto host collegato a una rete informatica che utilizza l'Internet Protocol come protocollo di rete.

Il DNS (Domain Name Server) è l'indirizzo del DNS server a cui fa riferimento per la traduzione da Ip a Hostname

Come fa il livello di rete, quando riceve un pacchetto, a conoscere quale protocollo di livello di trasporto e' destinato il pacchetto stesso?
Nel datagramma IP c'è un campo UPPER LAYER ed in base al numero identifica quale protocollo si riferisce, 6 per TCP, 16 per UDP, 1 per ICMP ed è il collante tra il livello di rete e quello di trasporto, come il port number è il collante tra il livello di trasporto e quello di applicazione.

Confrontare ed evidenziare similitudini e differenze nei campi di intestazione IPv4 e IPv6.

Nel IPv6 non c'è il checksum, viene fatto così da alleggerire il trasporto
Non ci sono i campi relativi alla frammentazione (identificatore, flags, offset fragmentation)
Non c'è più il campo option che era variabile.
Ora utilizza indirizzi 128bit invece di 32bit
Cambia anche l'intestazione che in IPv4 è di 20 byte fissi mentre in IPv6 ne ha 40

Le similitudini sono i restanti campi

È necessario che ogni AS utilizzi lo stesso algoritmo di instradamento interno? Motivare la risposta.

Un AS è un sistema autonomo, il quale contiene un insieme di routing di una certa regione. I vari AS comunicano tra di loro tramite i router di gateway.
Ogni AS può utilizzare al suo interno l'algoritmo che preferisce e generalmente si fa una scelta attraverso il protocollo OSPF

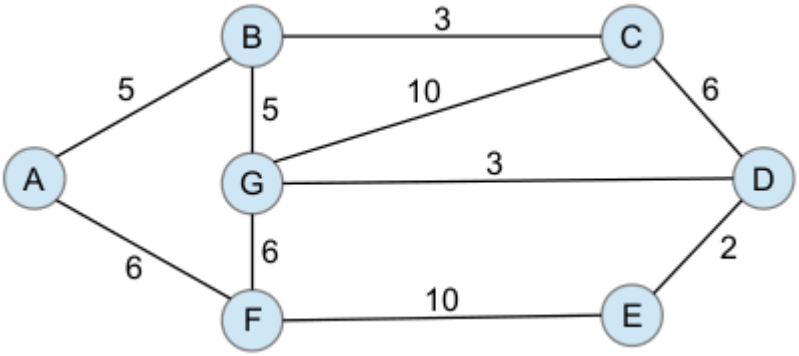
Come utilizza BGP gli attributi NEXT-HOP e AS-PATH?
AS-PATH: elenco degli AS attraversati. Server ad evitare annunci reiterati. Se un AS riceve un messaggio il cui compare il suo ASN, rifiuterà l'annuncio perchè è evidente che c'è un loop o qualche instradamento sbagliato. Un annuncio deve passare solamente una volta per AS.

NEXT-HOP: è l'esatta interfaccia da cui parte un AS-PATH. Questa distinzione è fondamentale quando si può raggiungere lo stesso AS da due router gateway diversi. Quando un router riceve una rotta ha vari modi di decidere cosa fare: ignorarla perchè ha già un percorso migliore, cancellarla ec… questi modi di gestire le rotte si chiamano politiche di impostazione e possono anche determinare se notificare il percorso agli altri AS vicini

Assegnare IP

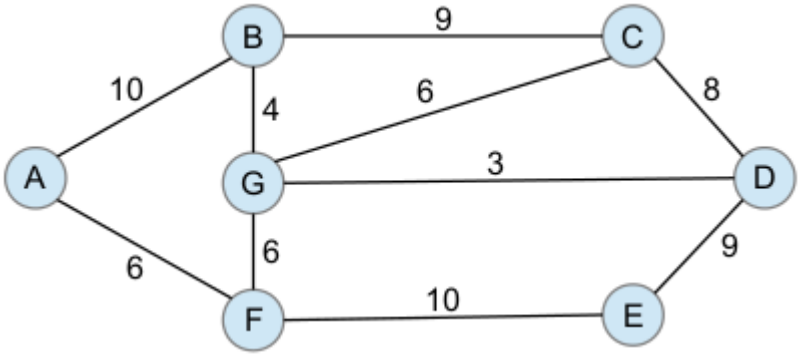
Per una Intranet si ha a disposizione la rete in classe B 129.174.0.0. Nella Intranet occorre installare almeno 15 reti locali collegate mediante dei router; descrivere come possono essere ricavati gli indirizzi per le sotto-reti e dire quanti host al massimo possono contenere le sotto-reti.
Ho bisogno di 4 bit per 15 reti perchè 2^4 fa 16
Per queste sotto reti ho al massimo 2^12-2 indirizzi disponibili, uno è per il gateway e l'altro per il broadcast che invia a tutti un messaggio per identificarli

Cammino minimo



a) Mostrare il funzionamento dell'algoritmo di Dijkstra per trovare i cammini minimi dal nodo A verso tutti gli altri presenti nella rete.
b) Disegnare l'albero dei cammini minimi dal nodo A verso tutti gli altri presenti nella rete.
c) Disegnare l'albero dei cammini minimi nel caso in cui il collegamento G-D venga eliminato.

Cammino minimo II



a) Mostrare il funzionamento dell’algoritmo di Dijkstra per trovare i cammini minimi dal nodo A verso tutti gli altri presenti nella rete.
b) Disegnare l’albero dei cammini minimi dal nodo G verso tutti gli altri presenti nella rete.
c) Disegnare l’albero dei cammini minimi nel caso in cui il collegamento F-E abbia peso 1.