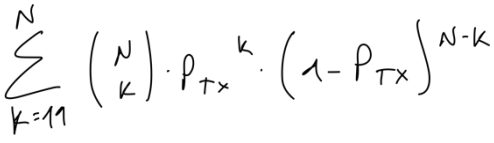
Cos'è internet? È un insieme di dispositivi hardware collegati fra loro tramite una infrastruttura. I router smistano e instradano i pacchetti verso la loro destinazione finale, consentendo agli host di scambiare dati. I punti di accesso permettono di accedere ad internet e possono essere wireless o wired. La rete fornisce un servizio alle applicazioni che può essere **affidabile** (da sorgente a destinazione perfettamente) o non affidabile "**best effort**" (potrei perdere pacchetti). Di base internet è non affidabile ma attraverso precisi protocolli si può renderlo affidabile. I protocolli definiscono regole per le interazioni fra entità, come il formato e l'ordine dei messaggi scambiati. L'architettura di una comunicazione può essere **client-server** o **peer-to-peer**.
Inizialmente si usava la rete telefonica come mezzo fisico (banda 56kbps) successivamente si è passati alla DSL che necessitava di un router e infine si stanno sostituendo i cavi in rame con fibra ottica per avere prestazioni ancora migliori. Una **LAN** è una rete locale che generalmente è anche connessa ad internet. I **mezzi trasmissivi** fisici possono essere guidati (fibra ottica, rame) o a onda libera (atmosfera). I cavi in rame possono essere di diverse categorie con diverse prestazioni, i cavi coassiali sono poco utilizzati in Italia, la fibra ottica ha caratteristiche molto buone (sottile, flessibile, basso tasso di errore, immune a interferenze elettromagnetiche, altissimo bitrate), i canali radio hanno vari problemi (interferenze, ostacoli, attenuazione).
I dati possono essere trasferiti attraverso la rete con:

- **commutazione di circuito** (circuit switching CS): un circuito viene dedicato per l'intera durata della sessione. Poiché le risorse sono riservate abbiamo prestazioni come il delay garantite, è necessaria però una fase di instaurazione della chiamata che potrebbe causare un ritardo. Il problema per la rete è che il canale riservato potrebbe essere inutilizzato a lungo. La banda è divisa in canali assegnati ciascuno ad una comunicazione (multiplazione nel dominio della frequenza FDM o multiplazione nel dominio del tempo TDM). Ogni router trasmette appena inizia a ricevere.
- **commutazione di pacchetto** (packet switching PS): il flusso informativo è suddiviso in pacchetti che utilizzano le risorse su richiesta e quindi potrebbero dover attendere per un collegamento. Le risorse sono usate a seconda della necessità (multiplazione statica). Store and forward: il commutatore riceve l'intero pacchetto prima di cominciare a trasmettere sul collegamento di uscita. Contesa per le risorse: in caso di congestione la richiesta di risorse può eccedere il quantitativo disponibile, in questo caso i pacchetti sono memorizzati in un buffer in attesa dell'utilizzo del collegamento (queueing delay). Nel caso in cui il buffer del router sia pieno si verifica il packet loss. Mi permette di avere molti più utenti contemporaneamente se trasmettono solo per un po' di tempo ciascuno.

Se ho due router di mezzo, con la PS dovrò avere 3 tempi di trasmissione (1 per l'host e 2 per i router), con la CS ho idealmente 1 tempo di trasmissione (solo l'host, perché i due router rilanciano immediatamente).
La **MTU** è la massima dimensione di un pacchetto inviabile, se devo mandare più dati li dovrò dividere in più pezzi (segmentazione). Con la CS ho come tempo end-to-end F/C dove F è la dimensione del file e C la capacità di trasmissione. Con la PS avrò come tempo (Np+Nh-1)L/C dove Np è il numero di pacchetti, Nh il numero di hop e L la grandezza di ogni pacchetto.
Se ho un collegamento da C=1Mbps e voglio fu=100kbps per utente sapendo che trasmetterà il 10% del tempo posso usare CS e avere 10 utenti o PS e avere 35 utenti con probabilità inferiore allo 0,0004 di averne più di 10 attivi. Con CS faccio C/fu. Con PS ho Ptx=0,1 probabilità che un utente trasmetta, 1-ptx=0,9 probabilità che un utente stia zitto, N numero totale di utenti, in figura la probabilità che almeno 11 utenti usino la rete contemporaneamente.

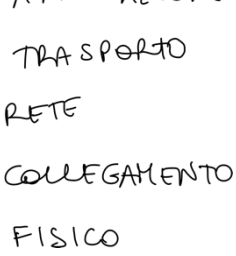


Se in un router il tasso di arrivo dei pacchetti eccede la capacità del collegamento in uscita, allora i pacchetti si **accodano nel buffer** in attesa di essere trasmessi causando un ritardo. Se non ci sono posti liberi nei buffer i pacchetti sono scartati e persi. Un ritardo può essere causato anche dal tempo di processamento in ogni router.
Ritardo dlink = somma di:

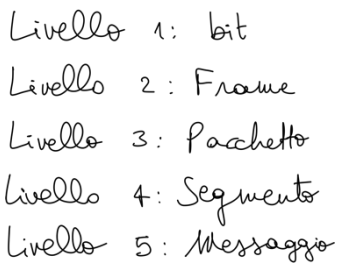
- delab: elaborazione ed instradamento verso l'interfaccia di uscita
- dqueue: accodamento nel buffer
- dtrasm: trasmissione = lunghezza del pacchetto / frequenza di trasmissione del collegamento
- dprop: propagazione = lunghezza collegamento fisico / velocità di propagazione del collegamento

Dati R frequenza di trasmissione, L lunghezza del pacchetto, a tasso medio di arrivo dei pacchetti, l'intensità di traffico è pari a L*a/R. Se è circa 0 il ritardo è limitato, se va verso 1 il ritardo cresce in modo non lineare, se è maggiore di 1 il ritardo medio è infinito. Protocolli come il TCP fanno inviare meno dati se il router è

congestionato. Quando un pacchetto viene scartato, in alcuni protocolli è ritrasmesso dal sistema terminale che lo ha generato, in altri è ritrasmesso dal nodo precedente, in altri non viene ritrasmesso.
Il **Throughput** è la frequenza (bit/unità di tempo) alla quale i bit sono trasferiti tra mittente e ricevente attraverso vari link. Può essere istantaneo o medio. Se si passa fra due link di velocità Rs e Rc, il throughput sarà il minore fra i due (il bottleneck). Generalmente i bottleneck sono nella parte di accesso alla rete ovvero in ingresso e in uscita. Traceroute: programma diagnostico che fornisce una misura del ritardo dalla sorgente ad ogni router fino alla destinazione.
Internet è una rete di reti con **struttura gerarchica**. Abbiamo gli ISP di livello 1 che sono i più importanti e sono in collegamento tra loro. Grazie ad accordi commerciali i router di ISP diversi vengono connessi fisicamente fra di loro. Gli ISP di livello 2 sono connessi a quelli di livello 1 e anche ad altri di livello 2 per risparmiare e non dover attraversare quelli di livello 1. Seguono gli ISP di livello 3 e quelli locali. Nei NAP vengono messi. I router di diversi ISP che sono poi connessi fra di loro.
Ogni livello realizza un servizio effettuando determinate azioni e utilizzando i servizi del livello inferiore.



A **livello di applicazione** abbiamo vari protocolli: http per la navigazione web, SSH per il controllo remoto, FTP per il trasferimento dei file, DNS per l'elaborazione di indirizzi, SMTP per l'invio di mail. Chi progetta un'applicazione non ha bisogno di conoscere i livelli sottostanti. A **livello di trasporto** abbiamo protocollo TCP (transfer control protocol) e UDP (user data protocol). A **livello di rete** abbiamo protocolli IP, RIP, OSPF, BGF. Sotto la rete c'è il **livello di collegamento** con diverse tecnologie come gli switch e standard come ethernet, wifi. Infine abbiamo il **livello fisico** che prepara l'informazione ad essere trasferita. In figura la **pila protocollare** che deriva dall'architettura ISO/OSI a 7 livelli. Partendo dal livello più alto e scendendo, il payload viene man mano incapsulato con aggiunta di header e infine viene trasmesso. A destinazione il messaggio risale pila venendo man mano decapsulato, in ogni passaggio viene levata ed elaborata una header.



Livelli:

- **Applicazione**: ho due tipi di applicazioni, client-server o peer-to-peer.
  Nel primo caso il client invia il primo messaggio richiedendo un servizio al server che può gestire più richieste contemporaneamente. La connessione fra le applicazioni e gli OS avviene attraverso porte (o interfacce) logiche, le socket. Poiché un server può essere distribuito su più dispositivi in un data center esistono dei **load balancer** che girano la richiesta al server fisico più libero, anche con l'ausilio di server DNS che possono restituire ogni volta un indirizzo IP diverso per lo stesso dominio.
  Nel secondo caso ogni macchina può essere sia client che server, viene utilizzato ad esempio nello scambio di file con torrent.
  Ogni processo invia e riceve messaggi sulla sua **socket** (porta), ogni porta ha una sua funzione e serve al SO per sapere a quale processo girare il pacchetto ricevuto. Ad esempio la porta 80 viene utilizzata per la navigazione web http.
  Ogni applicazione può necessitare di trasferimenti con caratteristiche spesso gestite dai livelli sottostanti. Queste richieste possono essere di affidabilità della connessione, ritardi piccoli o poco variabili, throughput minimo garantito, sicurezza.

  Nella **navigazione http** le pagine sono generalmente scritte in html e salvate in un server con indirizzo ip e hostname. Ogni pagina ha un URL che indica dove è salvata nel server. Il protocollo http regola le richieste e le trasmissioni: il client richiede il file e il server risponde inviandolo. Http sfrutta il protocollo TCP con porta 80 ed è stateless, ovvero non mantiene informazioni sulle richieste del client.
  La **connessione** può essere:
  - **Persistente**: più oggetti trasmessi su una singola connessione TCP condivisa che rimane attiva fino allo scadere di un timeout, T = 2*RTT + Nobj*RTT dove RTT è il round trip time e ne ho 1 per la richiesta TCP e 1 per la pagina html. Abbiamo prima una richiesta TCP, poi un ACK e poi le richieste e risposte html una dopo l'altra.
  - **Non persistente**: ogni oggetto è trasmesso su una connessione TCP dedicata che si chiude al termine del trasferimento, T = (2*RTT)(Nobj+1) dove 1 è per la pagina html e ho nobj aggiuntivi. Abbiamo ogni volta richiesta TCP e ACK e poi richiesta e risposta html.

http prevede due tipi di **messaggi**: **richiesta e risposta**. I **cookie** servono al server per tenere memoria e sono dei numeri casuali scelti dal server e salvati sul client. La prima volta che contatto un server mi viene assegnato un cookie che salvo, le successive volte lo invierò insieme alla richiesta al server. Dall'altra parte il server salverà i cookie generati in un database.

La **web cache** è spesso usata nelle aziende per evitare congestioni. Si salvano su un server le pagine relative a richieste frequenti e recenti per far sì che le successive richieste uguali non debbano attraversare il link di accesso alla rete internet ma possano rimanere nella LAN. In questo modo ogni richiesta passa prima per la web cache per controllare se è disponibile la pagina relativa e anche ogni risposta passa per la web cache per essere eventualmente salvata.

La **posta elettronica** si basa su tre protocolli principali: pop, imap e smtp. Un utente manda tramite il protocollo smtp un messaggio al suo server, che poi provvede a girarlo al server del destinatario. L'altro utente accederà al messaggio sul suo server tramite il protocollo pop o imap.

DNS: **domain name system**. Quando viene effettuata una richiesta con un URL, l'host name deve essere convertito nell'indirizzo ip del destinatario. I server DNS svolgono questa funzione. Si utilizzano dei database distribuiti per evitare singoli point of failure, dividere il livello di traffico e poter effettuare manutenzioni senza interrompere il servizio. C'è una **gerarchia dei server DNS**: root server, top-level domain (com, edu, it), server autoritativi (yahoo.it, amazon.com), DNS server locali. Se cerco l'ip di gaia.cs.umass.edu chiederò al DNS del top-level domain edu che mi risponderà con l'ip del DNS autoritativo umass.edu. Facendo una richiesta al DNS di umass.edu mi verrà restituito l'indirizzo ip cercato. La query può essere iterativa (come nell'esempio sopra) o ricorsiva (vado al root che chiede al TLD che chiede all'autoritativo e poi la risposta risale e mi viene data dal root). Esiste anche in questo caso un meccanismo di caching per cui ogni DNS può ricordare nella propria cache le query fatte e i relativi risultati per poter rispondere più velocemente e senza sovraccaricare altri DNS. Le richieste DNS utilizzano il protocollo UDP. Le tuple nei database dei DNS sono chiamate Resource Records RR e contengono (name, value, type, ttl). Esistono 4 tipi di tuple: A (name è l'host e value è l'IP), NS (name è il dominio e value è l'host server autoritativo), CNAME (servono a gestire gli alias) e MX (per posta elettronica). Quando si crea una nuova piattaforma web si registra il nome attraverso un registrar che inserirà le relative RR nel TLD legato al dominio.

- **Trasporto**: fa da tramite fra rete e applicazione, trasforma i dati in modo da poterli mandare e fornisce un collegamento logico fra i processi su host remoti. Viene eseguito solo nei dispositivi terminali. L'emittente divide i messaggi incapsulandoli in segmenti con intestazione (**multiplexing**) che verranno poi mandati allo strato di rete. Il ricevente li ricostruirà per consegnarli all'applicazione tramite la socket appropriata (**demultiplexing**). Ogni segmento contiene il numero di porta di origine e di destinazione, ciascuno da 16 bit. Abbiamo due protocolli principali TCP e UDP:

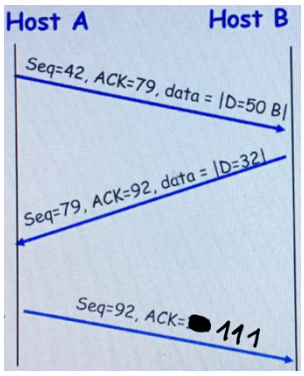
  TCP è orientato alla connessione poiché prevede una fase iniziale di setup fra client e server. Si occupa anche del controllo di flusso risolvendo perdite di pacchetti e rendendo internet affidabile. Inoltre regola l'invio se la rete è sovraccarica praticando quindi un controllo di flusso. Non riesce però a garantire un ritardo massimo, un throughput minimo o sicurezza. Una socket TCP ha 4 parametri: IP e porta di origine e di destinazione. Vengono create più socket per ogni processo, una per ogni host con cui deve comunicare.

  UDP è connectionless, prevede un trasferimento dati inaffidabile e manca di controllo di flusso, della congestione, del ritardo, del throughput e della sicurezza. In compenso non perde tempo e quindi è utile per applicazioni loss tolerant che richiedono alta velocità. Una socket UDP ha 2 parametri: IP e porta di destinazione. Viene creata una socket per processo anche se deve comunicare con più host diversi. Il segmento contiene un campo checksum calcolato alla partenza e all'arrivo sulla base degli altri campi per controllare che non ci siano stati errori di trasmissione.

I **numeri di porta** sono statici e riservati fra 0 e 1023 (es. 80 http, 22 ssh, 53 dns) o dinamici da 1023 in poi, ovvero assegnati dal sistema operativo all'apertura della connessione. Il sistema operativo crea una socket TCP o UDP in base alle richieste dell'applicazione.

Il livello di trasporto si appoggia al livello di rete che è inaffidabile. In alcuni casi serve quindi implementare delle funzionalità per rendere il **trasferimento affidabile**. Questo comporta la risoluzione di errori, di ritardi e la perdita di interi pacchetti. Per rilevare errori si usa il campo **checksum** e per correggerli si richiede un nuovo invio dell'intero segmento alla sorgente. Quando riceviamo un segmento rispondiamo con un **ack** o un **nak**. Quando la sorgente riceve un nak come risposta sa che deve rinviare il segmento. Se ack o nak si perdono o danneggiano viene considerato nak. In questo caso l'applicazione potrebbe ricevere due volte lo stesso segmento corretto e quindi vengono numerati per potersene accorgere. In ogni caso dopo una ricezione doppia si invia l'ack per non ricevere il segmento una terza volta.
Stop and Wait: il mittente attende la risposta prima di inviare il segmento successivo. Il numero di sequenza è 0 o 1.
Nak-Free: al posto di mandare un nak mando un ack con il bit dell'ultimo segmento ricevuto correttamente.
Vengono settati dei **timeout** al momento della spedizione di un segmento per ovviare al problema della perdita della risposta (ack o nak). Al termine del timeout rimando il segmento se non ho ricevuto risposte. Lo stop and wait è inefficiente perché perdo molto tempo, funziona se il PBR (C * RTT) è piccolo rispetto alla lunghezza del messaggio. L'efficienza si calcola 1 / ((C/L * RTT)+1).
In alternativa si possono utilizzare protocolli diversi che prevedono il pipelining come il go-back-n o il selective repeat. In questi casi mando più messaggi senza aspettare l'ack dopo ognuno. Considero una finestra di trasmissione Wt (numero massimo di ack che attendo) e vari bit per memorizzare i numeri di sequenza. Se attendo gli ack 3 4 5 6 e ricevo l'ack 6 allora so che tutti i segmenti prima sono stati anche loro ricevuti correttamente. Si preferisce il selective repeat come prestazioni, il go-back-n però risparmia inviando ack cumulativi.

Go-Back-N: il mittente può avere fino a N pacchetti senza ack in pipeline. Il ricevente invia solo ack cumulativi, accetta solo pacchetti in ordine e non dà l'ack di un pacchetto se c'è un buco. Il mittente ha un timer per il più vecchio pacchetto senza ack, se il timer scade allora ritrasmette tutti i pacchetti a partire da quello. Per individuare la finestra minima di trasmissione per evitare interruzioni, devo far sì che sia maggiore di (Ttrasm + Tack + 2Tprop) / Ttrasm poiché metto Ttrasm + Tack + RTT <= W*Ttrasm, considero RTT = 2*tprop, Ttrasm = L/C e Tack = Lack/C.



Selective Repeat (ripetizione selettiva): il mittente può avere fino a N pacchetti senza ack in pipeline. Il ricevente invia l'ack solo per singoli pacchetti, accetta anche pacchetti fuori sequenza per cui necessita di un buffer in ricezione. Il mittente ha un timer per ogni pacchetto che non ha ancora ricevuto ack, alla scadenza ritrasmette i pacchetti senza ack.
Il protocollo TCP funziona più o meno come il go-back-n: il numero di sequenza è il numero del primo byte del segmento nel flusso di byte (se n=42 allora il primo byte del campo dati di questo segmento è il 42esimo dei dati che sto trasferendo), l'ack è il numero di sequenza del prossimo byte atteso dall'host. |D| è la dimensione del campo dati.

Nel **TCP** i riscontri sono cumulativi e possono essere mandati in segmenti senza dati o aggiunti in segmenti che portano altri dati (piggybacking). Si utilizza un campo per il checksum e il timeout è assegnato solo all'ultimo pacchetto non riscontrato (e disattivato quando arriva l'ack). Il **timeout** deve dipendere dal RTT (che è però variabile) ed essere ben calibrato. Il RTT viene stimato con una media dei recenti RTT (sample RTT). EstimatedRTT = (1-α) * EstimatedRTT precedente + α * sampleRTT. È una media mobile esponenziale ponderata. Il valore di default α = 0,125. L'influenza dei vecchi campioni decresce esponenzialmente. Per il timeout si usa l'EstimatedRTT più un margine di sicurezza calcolato sulla base della deviazione standard dei precedenti EstimatedRTT. Retransmission TimeOut RTO =

EstimatedRTT + 4*DevRTT. Se un timeout scade e rimandiamo il pacchetto useremo un RTO maggiore perché immaginiamo che la rete sia congestionata RTOi+1 = q*RTOi dove i è il numero di ritrasmissione e q è generalmente uguale a 2.
C'è una eccezione nel caso in cui il mittente riceva degli ack duplicati mandati dal destinatario che ha ricevuto pacchetti successivi ad uno perso. In questo caso, dopo il terzo ack duplicato, il mittente rimanderà il segmento perso anche se non è scaduto il timeout. Non viene rinviata la finestra intera ma solo il segmento mancante. Dopo la ricezione del segmento perso il destinatario manderà direttamente l'ack richiedendo il nuovo segmento (non serve mandare gli ack dei segmenti a cui ha risposto con ack duplicati per segnalare la perdita di uno).

Il TCP prevede una fase iniziale di **instaurazione della connessione** con dei messaggi di controllo. **Handshaking a 3 passi**: A chiede a B di iniziare la connessione mandando un messaggio syn con un primo numero di sequenza casuale x, B risponde con un synack che contiene un altro numero di sequenza y e l'ack x+1, A riceve e manda un ack y+1. A questo punto la connessione è instaurata e si possono scambiare segmenti contenenti i dati. In questa fase si condivide anche la MSS **Maximum Segment Size** scelta da ogni host. Per chiudere una connessione si usa il messaggio FIN.

Controllo di flusso: TCP limita l'emissione dei dati dall'host per non riempire il buffer della destinazione. TCP usa un controllo di flusso basato su una finestra scorrevole di larghezza variabile. La destinazione notifica lo stato del suo buffer quando riscontra i segmenti, così che il mittente sappia quanto spazio libero rimane e gestisca il tasso invio. Oltre al sequence number e all'acknowledgement number c'è il RecWindow, ovvero il numero massimo di byte che il mittente può ancora emettere prima del prossimo riscontro. AckN = x e RecWindow = w indica che sono riscontrati tutti i byte ricevuti fino al x-1, inoltre l'emittente può trasmettere altri w byte (fino al x+w-1 byte).

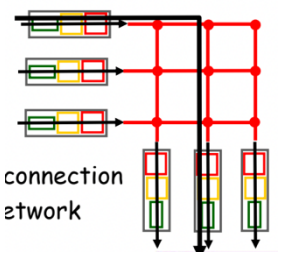
Controllo di congestione: bisogna scegliere quanti byte inviare anche a seconda dello stato della rete. Se si perdono molti pacchetti (scadenze dei timeout) suppongo che la rete sia congestionata e TCP reagisce diminuendo la finestra di trasmissione, così che si mandino meno dati. Così facendo la rete si decongestiona. Esiste anche un altro tipo di controllo di congestione che implica notifiche da parte dei router sul loro stato (non usato dal TCP).
Il TCP inizia inviando un solo segmento appena stabilisce la connessione, se va tutto liscio raddoppio il numero di segmenti inviati (ovvero raddoppio CongWin, ovvero incremento il valore della CongWin di un MSS per ogni ack ricevuto). Questo tipo di meccanismo è slow start, parte piano ma cresce esponenzialmente. Arrivati ad una soglia massima si passa ad una seconda fase (congestion avoidance) che implica un incremento lineare. Appena ho una perdita ritorno a CongWin = 1 e calcolo un nuovo valore di soglia massima che è metà del valore a cui abbiamo avuto la perdita. A questo punto ricomincio con lo slow start. La prima soglia viene determinata dal sistema operativo casualmente. In ogni istante scelgo come finestra di trasmissione la minore fra quella di ricezione e di congestione (valore di Adwin). Dopo 3 ack duplicati torno alla soglia.

- **Rete**: si occupa di gestire i messaggi generati da una macchina perché arrivino all'altra (consegna end to end). Il percorso di rete seguito dal pacchetto viene scelto grazie a **protocolli di routing** o instradamento. Questi consentono ad ogni router di sapere su quale interfaccia reindirizzare un pacchetto.
  I router lavorano sui livelli da 1 a 3. Hanno due funzioni principali:
  - **routing** (instradamento): trovare un percorso fra origine e destinazione. C'è una routing table che mi dice su quale interfaccia inoltrare il pacchetto in base all'indirizzo di destinazione. È una funzione del piano di controllo (funzioni necessarie al funzionamento della rete ed eseguite dalla rete nel complesso).
  - **forwarding** (inoltro): inoltro del pacchetto al router successivo. È una funzione del piano dati (si applica sui pacchetti) per cui deve essere rapida e spesso è implementata a livello hardware.

  Le **tabelle di routing** sono scritte dal protocollo di routing. Questo può essere basato su una architettura distribuita o centralizzata. Nel primo caso ogni router deve comunicare con gli altri, nel

secondo caso esisterà un controller che conosce tutta la rete e genera le tabelle per ogni router. Se è centralizzato può conoscere le congestioni e adattare le tabelle di routing in tempo reale.
Internet è orientato al connection-less ma alcuni protocolli sono orientati invece alla connessione. Lo strato di rete non distinguerebbe questi due modelli e potrebbe mandare pacchetti relativi alla stessa comunicazione su diversi percorsi. Per ovviare a questo problema si aggiunge nella header un campo che identifica la connessione (VCI **virtual circuit identifier**). All'instaurazione della connessione, in ogni tabella di instradamento si creerà una entry con INPUT VCI, Output Port e OUTPUT VCI. In questo modo il router potrà leggere il VCI in input, conoscere la relativa porta di uscita e modificarlo con il VCI di output (in questo modo il VCI cambia su ogni link).
Nel caso di comunicazioni connection-less non possiamo inserire regole singole per ogni possibile indirizzo per cui dobbiamo raggrupparli in qualche modo e rendere sempre più piccoli i gruppi man mano che mi avvicino alla destinazione. Nella tabella potrei quindi avere per ogni porta di uscita un insieme di indirizzi a cui porta. Se nella tabella ho due entry corrispondenti alla mia destinazione sceglierò quella che rappresenta più precisamente la mia destinazione (**longest prefix matching**). Generalmente viene anche inserita una regola di default applicata nel caso in cui nessuna delle altre sia valida.
Una CAM (**Content Addressable Memory**) è una memoria specializzata nell'eseguire operazioni di matching in parallelo per cui il tempo di accesso non dipende dalla grandezza della tabella. Le prestazioni di queste memorie sono elevate ma anche il consumo è molto alto. La CAM confronta l'intero input con l'intero contenuto, per cui è più efficiente una TCAM (CAM di tipo ternario) che permette di cercare direttamente il longest prefix in quando fa memorizzare 0, 1 e x (don't care). Le TCAM vengono utilizzate nei grandi router e restituiscono tutte le entry che fanno match (successivamente verrà scelta quella con il longest prefix matching).

Un **router** ha come **componenti** principali: input line card (ricevono i pacchetti con un eventuale buffer, processano le header e si occupano del routing), controller (funzioni di controllo e allocazione di risorse), switching fabric (forwarding fra porte di ingresso e uscita) e output line card (rinviano i pacchetti e si occupano di scheduling e priorità con un eventuale buffer). Generalmente la tabella di routing è in una memoria del router ma nelle line card vengono conservate le destinazioni più comuni in una cache. Il switching fabric esegue il trasferimento dei pacchetti dalla porta di ingresso a quella di uscita. Ne esistono tre tipi: memory (pacchetto scritto in memoria e letto dalla output line card), bus (bus condiviso fra le interfacce ma può essere utilizzato da un trasferimento alla volta), interconnection network (più usata, le line card sono connesse a griglia e ogni nodo è collegato a tutti gli altri).



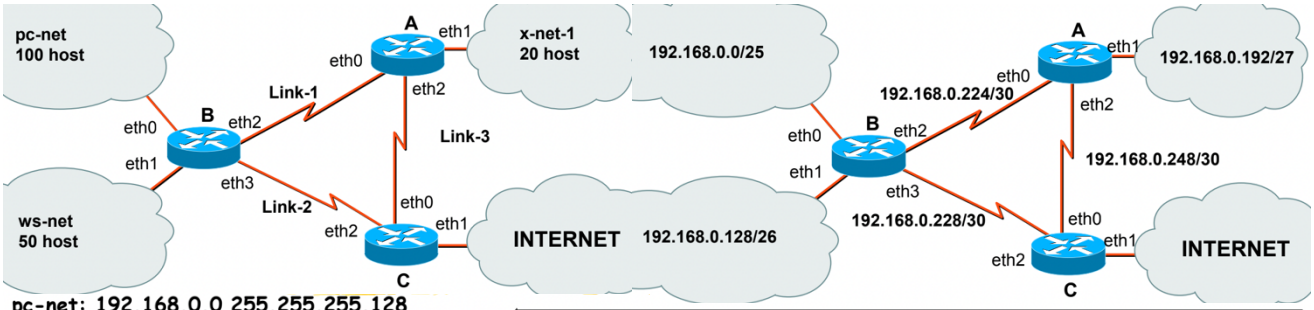
L'utilizzo dei **buffer** causa dei **problemi**: in input se ho due pacchetti su due line card destinati allo stesso output allora uno dovrà attendere (output port contention), se ho una coda di pacchetti sulla stessa line card il primo potrebbe bloccare quelli in coda se la sua line card di output è occupata (HOL blocking). La dimensione di un buffer deve essere pari almeno al PBR/√N dove N è il numero di flussi. Per gestire una coda posso usare come politica di scheduling la FIFO. In caso di saturazione del buffer si deve scegliere chi scartare: tail drop (si scarta l'ultimo), priority (si scartano pacchetti con priorità minore, per farlo necessito di code diverse per ciascuna priorità e dovrò svuotare sempre prima la coda di più alta priorità), random.

Il **protocollo IP** è connectionless e best effort. Definisce il formato dei pacchetti (header e dimensione massima 2^16 byte ovvero 65536 byte), uno schema di indirizzamento, le modalità di instradamento dei pacchetti, eventuali frammentazioni e riassemblaggi delle unità dati (se un pacchetto è troppo grande per entrare in un frame di livello 2).
Pacchetto IPv4 contiene una header di 20 byte. La lunghezza media di un pacchetto è 600-650 byte. Ogni rete fisica ha un valore massimo di lunghezza della propria unità informativa (Maximum Transmission Unit MTU), se questo valore è minore della dimensione del mio pacchetto IP questo verrà frammentato dal router o dall'host e poi ricomposto a destinazione. Nell'header ho dei campi dedicati alla frammentazione: identification (assegnato dal sorgente ed identificativo del pacchetto intero),

flags (un bit serve a indicare se la frammentazione è permessa o vietata, un altro serve ad indicare se quel frammento è l'ultimo o ne seguiranno altri), fragment offset (indica la posizione del frammento all'interno del pacchetto ed è espressa in unità di 8 byte). Un pacchetto di 4000 (3980 + 20 di header) byte con MTU 1500 dovrà essere diviso in tre pezzi di 1480 1480 1020 ciascuno con aggiunta di 20 byte di header.
Il protocollo ICMP (**Internet Control Message Protocol**) consente ai router di inviare all'host sorgente eventuali anomalie nell'elaborazione di un pacchetto (errori di instradamento, TTL scaduto, congestione eccessiva), è integrato in ogni implementazione IP. Viene usato per il ping e il traceroute.
Indirizzamento in IPv4 considerando solo indirizzi IP pubblici e quindi globalmente unici. Ogni indirizzo identifica una interfaccia di rete, singola in un host e numerose in un router. La lunghezza di un indirizzo è 32 bit che possono essere rappresentati anche con notazione decimale puntata e valori massimi 255.255.255.255 (sono 4 gruppi da un byte). Una **sottorete** è una rete isolata in cui i terminali sono collegati ad interfacce di host o di router e possono comunicare senza attraversare un router (ad esempio in una LAN). Un link diretto fra due router è una sottorete con due interfacce. Per calcolare il numero di sottoreti si cancellano i router dal grafico e rimangono tratte isolate che sono le sottoreti. I router sono quindi idealmente dei ponti fra sottoreti. Ogni indirizzo IP è formato da un prefisso **NET_ID** (identificativo di sottorete) e un suffisso **HOST_ID** (identificativo dell'host nella sottorete). Questi due elementi non hanno lunghezza fissa ma sommati devono dare 32 byte. È importante che non ci siano due HOST_ID uguali all'interno di una sottorete. Con /n si indica una **maschera di sottorete**, ad esempio 223.1.3.0/24 indica che il NET_ID è composto dai primi 24 bit dell'indirizzo indicato. In origine le sottoreti erano divise in classi individuate dai bit individuali dell'indirizzo (schema di indirizzamento **classful**). I prefissi avevano lunghezza definita. Il problema principale era lo spreco di indirizzi in quanto ad esempio per avere 257 host necessitavo di una rete di livello B e sprecavo più di 65000 indirizzi. Il multicast è un tipo di trasmissione in cui più ricevitori hanno lo stesso indirizzo e viene utilizzato ad esempio in dirette streaming. Se un host si muove deve cambiare indirizzo IP.

Classe	Bit iniziali	Net_Id	Host_Id	"Reti" disponibili	"Host" disponibili
A	0	7 bit	24 bit	128	16.777.216
B	10	14 bit	16 bit	16384	65.536
C	110	21 bit	8 bit	2.097.152	256
D	1110	Indirizzo multicast: 28 bit Indirizzi possibili: 268.435.456			
E	11110	Riservata per usi futuri: 27 bit Indirizzi possibili: 134.217.728			

Convenzioni: tutti 0 più host_id indica l'host nella rete locale, tutti 1 indica broadcast nella rete locale, net_id + tutti 1 indica broadcast sulla rete net_id e non è assegnabile, net_id più tutti 0 indica il prefisso di rete e non è assegnabile. Posso quindi assegnare in totale 2^(32-x) -2 indirizzi in una sottorete dove x è il numero di bit del net_id.
Successivamente è stato inserito un terzo livello gerarchico: il **subnet_id**, che prende alcuni bit dell'host_id. Serve per suddividere ulteriormente una rete con net_id prefissato. I k bit del subnet_id andranno sommati agli n del net_id nella maschera di sottorete x.x.x.x/n+k. Bisognerà sottrarre dalla disponibilità due indirizzi per ciascuna sottorete (broadcast e prefisso di rete). Per ricavare il net_id di k bit di un indirizzo IP si utilizza una maschera di sottorete formata da k 1 e poi tutti zeri, questa maschera viene poi messa in and con l'indirizzo IP e restituisce infine il net_id (scartando l'host_id).
Subnetting a lunghezza variabile: suddivido la mia rete di net_id in diverse sottoreti da gruppi di indirizzi di grandezza richiesta.
Routing in reti IP: se invio un pacchetto a una destinazione sulla stessa rete verrà reindirizzato direttamente, altrimenti andrà di default al router. Il router esaminerà l'IP di destinazione ed emetterà il pacchetto su una interfaccia di rete, consultando eventualmente la **routing table** per determinare il next-hop a cui mandare il pacchetto. Ogni riga di una routing table contiene il destination IP address, IP address del next-hop relativo, identificatore della porta di uscita, informazioni statistiche.
I **problemi** principali del protocollo IP sono la scarsità di indirizzi e la grandezza delle tabelle di routing. Sono state proposte varie **soluzione** a breve termine come l'utilizzo di indirizzi in modo **classless** o il **NAT**. A lungo termine si passerà al **protocollo IPv6** (indirizzi a 128 bit) ma per il momento coesistono.
Classless Inter Domain Routing (senza classi A, B, C, …). Ho un grande blocco con una maschera e lo suddivido in sottoreti (partendo da una /16 posso creare 4 /18). È più comodo delle classi perché non avendo maschere di lunghezze obbligate spreco meno indirizzi. È utile anche per l'instradamento perché partendo da lontano avrò un unico prefisso per tutte queste sottoreti (supernetting).



pc-net: 192.168.0.0 255.255.255.128
eth0 of router B: 192.168.0.1
Hosts (100): 192.168.0.2 → 192.168.0.101

ws-net: 192.168.0.128 255.255.255.192
eth1 of router B: 192.168.0.129
Hosts (50): 192.168.0.130 → 192.168.0.179

x-net-1: 192.168.0.192 255.255.255.224
eth1 of router A: 192.168.0.193
Hosts (20): 192.168.0.194 → 192.168.0.213

Link-1: 192.168.0.224 255.255.255.252
eth2 of router B: 192.168.0.225; eth0 of router A: 192.168.0.226

Link-2: 192.168.0.228 255.255.255.252
eth3 of router B: 192.168.0.229; eth2 of router C: 192.168.0.230

Link-3: 192.168.0.248 255.255.255.252
eth2 of router A: 192.168.0.249; eth0 of router C: 192.168.0.250

Destinazione	Netmask	Next hop	Interfaccia
192.168.0.0	255.255.255.128	d.c.	eth0
192.168.0.128	255.255.255.192	d.c.	eth1
192.168.0.224	255.255.255.252	d.c	eth2
192.168.0.228	255.255.255.252	d.c.	eth3
192.168.0.192	255.255.255.224	192.168.0.226	eth2
192.168.0.248	255.255.255.252	192.168.0.230	eth3
0.0.0.0	0.0.0.0	192.168.0.230	eth3

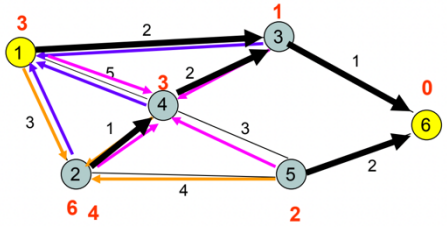
Esercizio: Parto con un blocco di indirizzi 192.168.0.0/24 e ho 6 sottoreti che necessitano rispettivamente di 101 (/25), 51 (/26), 21 (/27), 2 (/30), 2 (/30), 2 (/30) indirizzi.

Ogni host deve avere: indirizzo IP, subnet mask, default router, server DNS. Il DHCP (**dinamyc host configuration protocol**) è un protocollo di **autoconfigurazione** che permette un uso efficiente degli indirizzi IP. Setta tutti i parametri indicati sopra necessari all'host. È un protocollo plug-and-play. L'host invia un messaggio broadcast DHCP discover inserendo come source 0.0.0.0, il server DHCP risponde in broadcast con un DHCP offer proponendo un indirizzo disponibile, l'host richiede la configurazione con quell'indirizzo attraverso una DHCP request inviata sempre in broadcast (per avvertire eventuali altri DHCP server) e infine il server risponde con un DHCP ack inviando la configurazione. Gli indirizzi IP assegnati possono avere una scadenza alla quale l'host dovrà richiedere un nuovo indirizzo IP. L'utilizzo di server DHCP semplifica e rende più efficiente l'assegnazione degli indirizzi IP, richiede però che sia presente il server fisico in ogni rete, inoltre gli host devono essere configurati per usare il DHCP.
Il NAT (**Network Address Translator**) è utilizzato per ridurre l'utilizzo di indirizzi IP pubblici. Viene usato in reti gestite dagli ISP locali. Ad ogni rete viene assegnato un indirizzo IP pubblico visibile dalle reti esterne. All'interno vengono invece utilizzati indirizzi IP privati che non sono unici nell'internet. Ci sono dei blocchi assegnati agli indirizzi privati e sono da 10.0.0.0 a 10.255.255.255 (classe A), da 172.16.0.0 a 172.31.0.0 (classe B) e 192.168.0.0 a 192.168.255.0 (classe C). Gli indirizzi IP privati non sono raggiungibili dall'esterno ma solo dalla rete privata. Il traffico da e verso la rete privata avverrà con un solo indirizzo IP sfruttando varie porte. Verranno assegnati indirizzi IP pubblici unicamente agli host della sottorete che ne hanno necessità. Il router che connette la rete privata ad internet si occuperà di convertire l'IP sorgente privato in pubblico o l'IP destinatario pubblico in privato grazie alla NAT table. Ogni indirizzo IP pubblico potrà essere usato per gestire al massimo 60000 connessioni contemporaneamente, ovvero il numero massimo di porte. Esempio:
Pacchetto parte con S:10.0.0.1:3345 e D:128.119.40.186:80, nel router viene convertito S:138.76.29.7:5001 ed è creata una entry nella NAT table. La risposta avrà S:128.119.40.186:80 e D:138.76.29.7:5001, nel router viene convertito D:10.0.0.1:3345 e mandato al destinatario. Teoricamente viola la pila dei protocolli perché decapsula troppo rispetto a quel che dovrebbe, modifica IP e porta per cui viene anche ricalcolato il checksum.
Calcolando i percorsi di rete si creano le **tabelle di routing**. Ci sono vari percorsi da sorgente a destinazione e bisogna spesso scegliere il migliore secondo vari criteri: minimi ritardo, numero di hop, costo, massima affidabilità. Per calcolare questi parametri bisogna assegnare varie informazioni ad ogni link chiamate **metriche** (stato congestione, delay, up/down in un istante). Ogni **algoritmo di instradamento** si basa su metriche singole o multiple che vengono scambiate fra i router, inoltre definisce la frequenza e i destinatari (vicini o broadcast) dei messaggi contenenti queste metriche. Ogni protocollo di routing deve essere rapido per rispondere a variazioni di topologia o banda dei link, congestione in un instante, assenza di loop. Deve usare ottimamente le risorse e minimizzare i

cammini, funzionare in condizioni anomale ed avere basso carico di elaborazione. Generalmente vengono salvati anche dei percorsi di backup utilizzati nel caso ci siano problemi. Assegnando indirizzi IP con una gerarchia avrò indirizzi con stesso prefisso vicini fisicamente e le tabelle di routing si potranno accorciare. Le tabelle di routing sono aggiornate dinamicamente e spesso tramite i protocolli di instradamento, che calcolano i percorsi sulla base del database topologico.

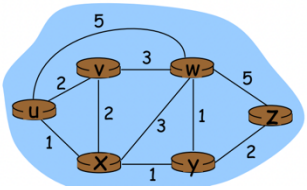
Un **sistema autonomo** (AS) è un insieme di host e router controllati da un ISP. I Core AS (centrali) consentono agli Stub AS (periferici) di connettersi fra di loro e al resto del mondo. Ogni AS può scegliere un suo protocollo di routing interno (IGP **interior gateway protocol**). Esiste invece un solo protocollo di routing esterno (EGP **exterior gateway protocol**) chiamato BGP.
Consideriamo la rete come un grafo pesato con nodi (router) e archi (link) ciascuno con un costo c relativo alla metrica utilizzata (∞ se non c'è il collegamento). Il costo di un cammino (o lunghezza o distanza) è la somma dei costi associati agli archi componenti il cammino. Il protocollo di instradamento deve determinare il cammino a costo minimo tra due nodi della rete. Possibili metriche per stabilire il costo sono: numero di hop, affidabilità (BER), ritardo, bandwidth, carico. Abbiamo due tipi di protocolli:
Distance Vector Protocol: router adiacenti scambiano la lista delle distanze verso le destinazioni, non conoscono la totale topologia della rete, base dell'algoritmo di **Bellman-Ford**. Nella tabella di routing sono memorizzati in ogni entry: destinazione, next-hop e distanza (costo del cammino minimo), i router adiacenti si scambiano dei Distance Vector contenenti (destinazione, distanza) periodicamente o dopo un cambio di stato. Ogni nodo determinerà poi per ogni destinazione il next-hop migliore.
Funzionamento: se un nodo j conosce la distanza minima per arrivare al nodo k, il nodo i connesso con costo c a j sa che per arrivare a k avrà come costo D(j,k)+c. A questo punto controlla se nella sua tabella di routing c'è già un percorso verso k e sostituisce quello nuovo se il costo nuovo è minore del vecchio. È un protocollo iterativo e asincrono, ogni nodo quando riceve un messaggio del cambio del costo da parte del suo vicino effettua il calcolo del costo dei percorsi e, se il distance vector cambia, invia una notifica ai suoi adiacenti. È un protocollo a lunga convergenza e se si disconnette un link rischio che si crei un loop, per evitarlo non trasmetto un DV aggiornato verso il router da cui ho ricevuto aggiornamento (**split horizon**). Con il **poisoned reverse** si trasmette il DV verso chi ci ha mandato l'aggiornamento ma con distanza ∞. Nell'esempio in figura una costruzione della tabella di routing per raggiungere il nodo 6. In ogni casella si usa il formato (Next hop, costo).

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1,∞)	(-1,∞)	(-1,∞)	(-1,∞)	(-1,∞)
1	(-1,∞)	(-1,∞)	(6,1)	(-1,∞)	(6,2)
2	(3,3)	(5,6)	(6,1)	(3,3)	(6,2)
3	(3,3)	(4,4)	(6,1)	(3,3)	(6,2)



Link State Protocol: i costi dei link sono diffusi in rete (flooding), i router conoscono quindi l'intera topologia della rete e calcolano lo shortest path verso ogni destinazione e il next hop relativo, la base è l'algoritmo di **Dijkstra**. Viene individuato il cammino minimo fra un nodo s e tutti gli altri nodi del grafo G, ovvero si crea un albero minimo ricoprente (MST). Come ipotesi c'è la conoscenza da parte di ogni router di tutta la rete in tempo reale (ovvero considerando eventuali modifiche). L'algoritmo di Dijkstra è iterativo: divide l'insieme dei nodi N in V (visitati) e T (non visitati). All'inizio inserisce in V la radice e in T gli altri nodi, inoltre setta il costo di c(i,j) = ∞ se i e j non sono adiacenti o al costo del ramo se sono adiacenti. In ogni iterazione sposta da T a V il nodo con distanza minima per arrivare all'origine fra quelli che ancora sono in T. Successivamente aggiorno i cammini minimi dalla sorgente ad ogni nodo ancora in T scegliendo il costo minimo fra quello attuale e la somma del costo per arrivare al nodo appena spostato con la distanza del nodo appena spostato dall'origine. Termino quando non ho più nodi in T. In ogni iterazione viene aggiunto nell'insieme V un

passo	T_k	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

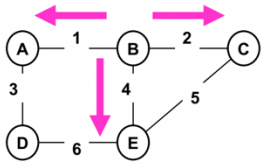


Percorsi a costo minimo tra il nodo "u" e tutti gli altri nodi
D(i) = costo minimo per percorso tra il nodo di origine ed il nodo "i"
p(i) = predecessore del nodo "i" sul percorso minimo tra in nodo origine il nodo "i"

nodo ed è sicuramente individuato il suo percorso minimo per arrivare all'origine. È molto importante che venga tutto aggiornato in **tempo reale** (nodi e costi) per evitare di usare percorsi che non esistono più o il cui costo non è più minimo. Non sempre conviene aggiornare immediatamente i costi di un link perché si rischia di spostare tutto il traffico lasciando il link originariamente congestionato totalmente vuoto. Si usano quindi i **costi quasi-statici** che quindi variano poco (magari a seconda di fasce orarie).

Instradamento gerarchico. Abbiamo finora considerato che ogni router fosse indistinguibile dagli altri e avesse una visione omogenea della rete. Nella realtà Internet è una rete di reti per cui ogni ISP deve essere libero di amministrare la propria rete come desidera, mantenendo naturalmente la possibilità di connetterla alle reti esterne. Inoltre archiviare informazioni di instradamento su ciascun host richiederebbe enormi quantità di memoria e il traffico generato dai pacchetti dedicati a questi protocolli non lascerebbe banda ai dati veri e propri. Per questo i router sono organizzati in sistemi autonomi AS, all'interno dei quali è usato lo stesso **protocollo IGP d'instradamento intra-AS**. I router appartenenti a diversi AS possono eseguire **protocolli EGP inter-AS concordati**. In questo modo ciascun router interno ad un AS conosce le destinazioni interne, nel caso in cui debba connettersi ad una destinazione esterna manda il pacchetto al gateway che si occupa di inoltrarlo verso destinazioni esterne ad un AS. Fra i vari operatori si tende a focalizzarsi spesso sull'aspetto economico lasciando le prestazioni in secondo piano, in particolare nella scelta degli AS esterni da cui passare per giungere a destinazione. Un AS potrebbe però avere più gateway per cui è necessario che i router interni sappiano che destinazioni esterne può raggiungere ogni gateway. Se un AS esterno è raggiungibile da due gateway diversi si hanno due scelte: instradamento a **hot potato** (si sceglie il percorso più breve per uscire dall'AS) o **cold potato** (si sceglie il gateway con costo minimo per raggiungere la destinazione).

Protocolli di instradamento **intra-AS** (IGP) sono:
RIP: **Routing Information Protocol**, basato su Bellman Ford. È un distance vector routing protocol. La metrica utilizzata per i rami è il loro stato (sano indicato con 1, guasto indicato con 15 – max hop). Viene utilizzato in reti di piccole dimensioni perché ha convergenza lenta. Prevede messaggi request con cui si richiedono i DV dei vicini e messaggi response con cui si inviano i DV. Il response viene inviato di default ogni 30 secondi. Utilizza il protocollo UDP su porta 520. Se non ricevo nulla per 180 secondi da un nodo adiacente lo considero spento o guasto. In questo caso RIP modifica la tabella di routing e manda annunci ai suoi router adiacenti, loro lo fanno a loro volta e l'informazione si propaga. Fa uso del poisoned reverse per evitare loop utilizzando come distanza infinita 16 (max hop + 1).



- **Step 8:**
  - B emette un messaggio verso A, C e E

B	Address	A	B	C	D	E
	Metric	1	0	1	2	1

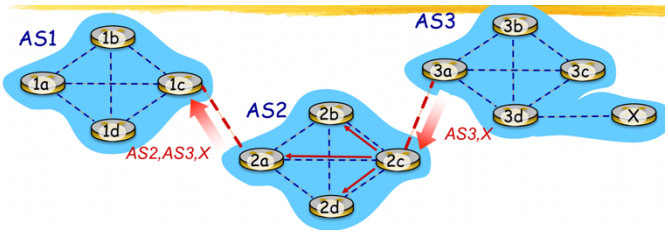
Routing Table

A	Destinazione	A	B	C	D	E
	Distanza	0	1	2	1	2
	Link	local	1	1	3	1
B	Destinazione	A	B	C	D	E
	Distanza	1	0	1	2	1
	Link	1	local	2	1	4
C	Destinazione	A	B	C	D	E
	Distanza	2	1	0	2	1
	Link	2	2	local	5	5
D	Destinazione	A	B	C	D	E
	Distanza	1	2	2	0	1
	Link	3	3	6	local	6
E	Destinazione	A	B	C	D	E
	Distanza	2	1	1	1	0
	Link	4	4	5	6	local

OSPF: **Open Shortest Path First**, basato su Dijkstra. È di tipo link state per cui conosce l'intera rete tramite il flooding, ovvero l'invio di messaggi LSA **Link State Advertisement** e il rilancio da parte di ogni router di questi pacchetti su tutte le interfacce eccetto quella di ricezione. Un LSA viene trasportato in pacchetti IP. Prevede una convergenza molto rapida. È il protocollo più utilizzato, è sicuro, è ECMP **equal path cost multipath** (associa ad ogni destinazione tutti i percorsi di pari costo minore senza doverne scegliere uno in anticipo, in questo modo posso dividere il traffico nei vari percorsi). Può tenere conto di diverse metriche e offrire percorsi diversi in base all'obiettivo. Ha supporto al multicast. I pacchetti LSA sono emessi se rilevo un nuovo router adiacente, se un link si guasta, se il costo di un link varia e periodicamente con intervalli di tempo prefissati. Gli LSA contengono un timestamp per poter capire se l'ho già ricevuto e devo ignorarlo. Ogni volta che ricevo un LSA lo aggiungo al database degli LSA. Il problema di questo protocollo è l'elevato traffico generato per il **flooding**. Per ovviare a questo e ad altri problemi nel caso in cui la rete sia di grandi dimensioni si suddivide in **aree OSPF**. In questo modo diminuisco il numero di LSA generati, riduco il numero di memoria utilizzata per memorizzarli e riduco il tempo di esecuzione dell'algoritmo di Dijkstra. Le

singole aree avranno database separati, meccanismi di flooding indipendenti e saranno connesse tramite aree di **backbone** (sempre interne all'AS). Ogni OSPF area avrà un ABR **area border router** che la collega alla backbone e permette di scambiare direttamente i percorsi minimi già elaborati (e non la topologia di rete completa). Ogni ABR contiene i database di tutte le aree a cui appartiene ed emette dei summary records che contengono la lista delle destinazioni raggiungibili da quell'ABR. Non comunica il percorso per raggiungere le destinazioni perché tanto il pacchetto passerà da lui che saprà come indirizzarlo. Un ASBR **autonomous system border router** connette diversi AS e per questo motivo dovrà implementare anche un EGP. Dall'interno dell'AS riceverà pacchetti che hanno destinazioni esterne generiche, all'esterno dovrà far sapere quali destinazioni contiene il suo AS. Nel caso in cui un AS abbia più ASBR, ciascuno dovrà notificare ai router interni i costi minimi per raggiungere destinazioni esterne (external record). Gli **LSA** possono essere di tre tipi: Router link advertisement (stato link di un router, inviati in area), Summary link advertisement (generati da ABR e individuano reti delle altre aree con relativi costi di raggiungimento, sono inviati in tutte le aree gestite da un ABR), AS external link advertisement (generati da ASBR e indicano i costi verso le reti esterne al dominio OSPF, sono inviati in tutte le aree del dominio OSPF). Tutti gli LSA hanno lo stesso header più alcune aggiuntive specifiche. Fra i campi abbiamo il TOS (tipo di metrica), Metric (costo). L'**ECMP** è una caratteristica che permette di salvare tutti i cammini a costo minimo uguale per poter suddividere il traffico sui diversi cammini. Viene utilizzato il flow hashing ECMP per ripartire il traffico, ovvero si sceglie il next hop sulla base dell'hash di alcuni campi dell'header del pacchetto (ip e port source e destination, ip protocol e interfaccia di ingresso). In questo modo i pacchetti dello stesso flusso tenderanno a prendere lo stesso percorso (se ho tre interfacce di uscita scelgo una funzione di hash che restituisca 0 1 2 e assegno ad ogni valore un'interfaccia di uscita).
IGRP: **Interior Gateway Routing Protocol**, basato su Dijkstra ed utilizzato da Cisco

L'unico protocollo di instradamento **inter-AS** (EGP) è il BGP **border gateway protocol** ed è costituito da due diversi protocolli: **eBGP** (scambio di informazioni fra due sottoreti di AS confinanti) e **iBGP** (propaga le informazioni sui router interni). Il BGP consente ad ogni rete di comunicare la sua esistenza al resto di internet. È un **path vector**, ovvero annuncia i cammini verso le sottoreti di destinazione. I gateway router di confine fra gli AS eseguono sia eBGP che iBGP. È un tipo di protocollo non link state ma path vector (distance vector più aggregato). Due BGP router (peers) si scambiano messaggi utilizzando una connessione TCP affidabile (BGP session). Annunciano i cammini verso le reti di destinazione, ovvero inviano all'altro BGP router una lista delle destinazioni che possono raggiungere (BGP advertisement). Esempio in figura: il router 2c riceve un messaggio di path advertisement AS3,X dal router 3a (indica che può raggiungere X dall'AS3) tramite protocollo eBGP. Il router 2c accetta il cammino AS3,X e lo rilancia a tutti i router del suo AS tramite protocollo iBGP. Il router 21 annuncia il cammino AS2,AS3,X al router 1c tramite protocollo eBGP. Se ci fosse stato anche un collegamento 3a-1c, anche 1c avrebbe ricevuto AS3,X e lo avrebbe preferito a AS2,AS3,X. In ogni caso quando si riceve uno di questi messaggi si salva nella propria routing table la destinazione e il next hop (ovvero il router da cui ho ricevuto la comunicazione). Ogni AS può avere delle politiche interne per accettare o rifiutare le rotte (ovvero i messaggi di cui sopra), e sono legate ad esempio ad accordi commerciali. ASPath è la lista di AS attraversati in un cammino. ASNumber è un identificatore a 16 bit che identifica univocamente un AS. Ogni router può ricavare più di una rotta verso un determinato prefisso e ne deve scegliere una secondo regole di eliminazione (utilizzo di valori di preferenza locale, AS-PATH più breve, router di next-hop più vicino ovvero hot potato). I messaggi BGP possono essere di tipo open (apre connessione TCP), keep alive (mantiene connessione), update (avvisa di nuovi percorsi), notification (avvisa in caso di errori). Le questioni di policy relative a sistemi Inter-AS sono legate al controllo del traffico instradato sul mio AS e al controllo di chi instrada attraverso le mie reti più che alle prestazioni, le policy in un sistema Intra-AS hanno un ruolo meno importante nella scelta delle rotte interne per cui si dà rilievo alle prestazioni.



IPv6 è l'evoluzione del protocollo IPv4 e si è reso necessario per risolvere il problema della scarsità di indirizzi. Un indirizzo è composto da 128 bit e l'indirizzamento è gerarchico e basato sui prefissi. Le header hanno lunghezza fissa, le differenze con IPv4 riguardano la codifica del campo options, l'eliminazione dei campi checksum e dei campi relativi alla frammentazione. I flussi di pacchetti diventano individuabili grazie alle **flow label**, utili per tenere i pacchetti di uno stesso flusso informativo sullo stesso percorso end-to-end. Questo protocollo autoconfigura le interfacce di rete ed è integrato con l'architettura IPSec (security). Si considera una **intestazione di base** a cui vengono aggiunte **extension header** solo se necessarie (per risparmiare spazio se non necessario e tempo di elaborazione su ogni router). Non è più necessario (ma in alcuni casi utilizzato) un server DHCP nella rete perché l'host può autoconfigurare autonomamente la propria interfaccia di rete. L'header ha lunghezza 40 bytes, il payload massimo 65535 bytes. Nella **header principale** abbiamo un campo Traffic Class legato alla priorità del pacchetto e un campo Next Header che indica il codice della header che segue nel pacchetto (vedi figura, quelle in grigio sono novità di IPv6). La numero **0** contiene informazioni elaborate da ogni sistema intermedio sul percorso del pacchetto, come ad esempio l'impostazione di jumbo payload che consente di trasportare payload >64kb. La **44** è relativa al servizio di **frammentazione** che viene permesso solo nel nodo sorgente per evitare di perdere tempo nei router e di aumentare eccessivamente l'overhead. Il problema di consentire la frammentazione solo nella sorgente è che non è a conoscenza degli MTU dei link nel percorso verso la destinazione, per questo è stata inserita una procedura di Path MTU Discovery che permette all'host sorgente di scoprire l'MTU minore del percorso e frammentare correttamente il pacchetto prima dell'invio. **50 e 51** sono legate al protocollo **IPSec**, solo sorgente e destinazione sono in grado di leggere il contenuto del payload grazie ad algoritmi di cifratura. Inoltre serve ad assicurare che il pacchetto provenga veramente dal mittente indicato e che questo campo non sia stato modificato lungo il tragitto. La **43** serve ad implementare il **source routing** che permette alla sorgente di stabilire dei nodi per cui viene richiesto che il pacchetto passi prima di arrivare a destinazione, indichiamo con strict explicit routing la pratica di settare tutti i nodi che verranno attraversati, loose explicit routing la pratica di settare solo alcuni nodi intermedi. Viene indicato nel campo Segment Left la quantità di router intermedi specificati. Per realizzare questa richiesta viene impostato come destination address iniziale il primo router indicato nella header 43 e viene decrementato di 1 il campo segment left. Appena raggiunta la nuova destinazione si controllerà il campo segment left e, se maggiore di 0, si sostituirà il campo destinazione della main header con l'indirizzo del successivo nodo obbligato e si decrementerà segment left. Si considererà arrivato a destinazione il pacchetto con segment left uguale a 0. Tutte le intestazioni sono generate dalla sorgente ed elaborate solo nella destinazione eccetto la numero 0 e la 43. Ogni **extension header** ha il campo NH next header e la HdrLength che indica la lunghezza della header esclusi i primi 8 byte.

0	Hop-by-hop options header
4	Internet Protocol (IP)
6	Transmission Control Protocol (TCP)
17	User Datagram Protocol (UDP)
43	Routing
44	Fragment Header
50	Encapsulating Security Payload
51	Authentication Header
58	Internet Control Message Protocol
59	No Next Header
60	Destination Options Header

Gli **indirizzi IPv6** sono composti da 128 bit ed indicati generalmente in esadecimale. FF è 1111 1111, FE è 1111 1110, F1 è 1111 0001. Si scrivono in 8 gruppi da 16 bit, ovvero 8 gruppi da 4 esadecimali, separati da :. L'utilizzo di :: indica che sono presenti solo gruppi 0000 in quel punto, tanti quanti sono quelli necessari a completare i 128 bit totali, l'utilizzo di :: è consentito una sola volta per indirizzo. Si possono omettere gli zeri all'inizio di un gruppo: 001E diventa 1E. Si utilizzano solo sottoreti con **maschera /64** e si considerano tre tipi di indirizzi: **unicast** (indirizzo relativo ad una interfaccia e si consegna solo a quella), **anycast** (indirizzo relativo a più interfacce ma si consegna solo ad una, è il caso dei DNS per cui basta consegnare la richiesta al server più vicino) e **multicast** (indirizzo relativo a più interfacce e si consegna a tutte, generalmente utilizzato per live streaming). Non è più predisposto un indirizzo broadcast la cui funzione viene sostituita dal multicast. Anche in questo caso gli indirizzi sono divisi in due parti logiche: net_id e host_id. Gli indirizzi si indicano sempre con IPv6 address/prefix address. Indirizzi tipici sono ::/128 (non specificato), ::1/128 (loopback ovver 127.0.0.1), FF00::/8 (multicast, comprende anycast), FE80::/10 (**link-local unicast**, per connettere dispositivi della stessa LAN), FEC0::/10 (**site-local unicast**, connessione di dispositivi della stessa organizzazione o regione ad esempio quelli privati della Fastweb, sono deprecati), qualsiasi altro per **global unicast** (equivalenti agli IPv4 pubblici). Ogni scheda di rete avrà configurati almeno due indirizzi IPv6 (uno link-local e uno global). La struttura di un indirizzo è: n bit per il global routing prefix (identifica un sito complesso), m

bit per il subnet ID (identifica una sottorete nel sito), 128 - n - m bit per **l'interface ID** (corrisponde all'indirizzo fisico dell'interfaccia che deriva in automatico dall'indirizzo MAC unico globalmente della scheda di rete composto da 48 bit). Generalmente si usa n=48bit, m=16bit e 64bit per l'interface ID. Gli indirizzi link-local non possono essere utilizzati su pacchetti rilanciati da un router, ma sono comodi perché una scheda di rete può **autoconfigurarsi** in automatico un indirizzo di questo tipo.
I prefissi sono /48 + 16 per le sottoreti nel caso di provider, prefisso /64 quando un sito gestisce una singola sottorete, /128 per un singolo dispositivo. Un provider di grandi dimensioni può richiedere che gli venga assegnato uno spazio delimitato da un prefisso più breve di 48 bit. La **partizione degli indirizzi** è quindi quasi statica, ogni rete ha generalmente lo stesso spazio di indirizzamento, una rete mobile (con dispositivi che variano spesso) ha la possibilità di gestire molteplici dispositivi terminali. La configurazione dell'indirizzo di una interfaccia può essere di due tipi:
Stateless autoconfiguration: non serve un server. Viene generato il link-local address per l'interfaccia, viene inviato sulla rete un messaggio di Neighbor Solicitation con il nuovo indirizzo per verificarne l'autenticità. Nel caso non si ricevano risposte negative dagli altri dispositivi viene inviato un messaggio ddi Router solicitation per ottenere dal router l'indicazione del prefisso di rete per formare gli indirizzi site-local e global. Il router emette un messaggio Router Advertisement per rispondere alla richiesta di un nodo o periodicamente per consentire verifica e aggiornamento degli indirizzi esistenti. Questo tipo di configurazione non permette di controllare l'assegnazione ed è eseguibile solo in una rete multicast come la LAN.
Stateful autoconfiguration: si ricevono le informazioni da un server che può quindi controllare l'assegnazione. Il DHCPv6 prevede che per comunicare con il DHCP server si usi il proprio link-local address per poter ottenere indirizzi global. Ha la possibilità di fornire molteplici indirizzi per ogni interfaccia e utilizza pacchetti IPv6 per scambiare messaggi.
Ogni indirizzo può avere durata infinita o finita, in questo caso alla scadenza l'indirizzo potrà essere associato ad un altro host. L'unicità degli indirizzi è garantita da un **algoritmo di rivelazione di indirizzi duplicati** che è eseguito prima di utilizzare l'indirizzo assegnato (in stateless e stateful).
La **transizione da IPv4 a IPv6** dura da molti anni e continuerà, infatti questi due protocolli coesistono tramite meccanismi ben definiti. La soluzione più semplice è il **Dual Stack** e prevede che ogni router possa supportare entrambi i protocolli. Un'altra soluzione sono gli indirizzi IPv4 immersi: un nodo che supporta IPv6 può indirizzare nodi che supportano solo IPv4 creando indirizzo formato da 80 bit 0 + 16 bit 1 + 32 bit dell'indirizzo IPv4 di destinazione (**IPv4 mapped**), un nodo IPv6 può assegnarsi un indirizzo convertibile in IPv4 del tipo 64 bit 0 + 16 bit 1 + 16 bit 0 + indirizzo IPv4 (**IPv4 translated**). Un router dual stack provvederà a convertire l'indirizzo IPv4 mapped dell'host IPv4 e l'indirizzo IPv4-translated dell'host IPv6. L'ultima soluzione prevede il **Tunneling**, che può essere router to router, host to router, router to host o host to host. La rete IPv6 si sviluppa a isole all'interno della rete IPv4 esistente e le varie isole IPv6 si connettono attraverso l'infrastruttura IPv4. La presenza di un payload IPv6 in un pacchetto IPv4 è indicata dal valore 41 nel campo protocol. I tunnel possono essere configurati se l'endpoint del tunnel è un router, automatici se l'endpoint del tunnel è la destinazione finale del pacchetto. Tunnel automatico Router to Host: ho due host IPv4/IPv6, il primo host invia pacchetto IPv6 (con indirizzi IPv6 ma compatibili con IPv4), un primo router dual stack converte gli indirizzi in IPv4 e aggiunge una header IPv4 con l'indirizzo di destinazione convertito, l'indirizzo proprio come sorgente e il campo protocol settato a 41, questo nuovo pacchetto viaggerà in una rete IPv4 fino ad arrivare all'host finale che provvederà a eliminare l'header IPv4. Tunnel automatico Host to Host: ho due host IPv4/IPv6 che generano pacchetti IPv6 destinati a viaggiare in una rete esclusivamente IPv4, prevedono direttamente loro a incapsularli in pacchetti IPv4 come spiegato sopra e decapsularli all'arrivo. Tunnel configurati Router to Router: ho due host IPv6 che generano pacchetti IPv6 con indirizzi non necessariamente compatibili con IPv4, questi raggiungono un primo router dual stack che aggiunge l'header IPv4 con il proprio indirizzo IPv4 come sorgente e l'indirizzo IPv4 del router di destinazione, il pacchetto viaggia nella rete IPv4 e arrivato al router dual stack di destinazione viene riconvertito in IPv6 e mandato all'host di destinazione.

SDN **Software Defined Networking**. Il protocollo IP, oggi alla base di internet, è datato, i dispositivi di rete sono proprietari con procedure di configurazione differenti e ciascuno specializzato per una funzione. Introdurre un nuovo protocollo richiederebbe però l'aggiornamento del SO o la sostituzione del dispositivo. Il **Clean State Project** della Stanford University ha portato alla definizione del protocollo **Open Flow** cercando di ripensare internet partendo da zero. Un apparato IP classico è integrato verticalmente, ha un hardware dedicato molto rapido nell'esecuzione di operazioni di forwarding, ha un sistema operativo che implementa le funzionalità di base del router e le funzionalità aggiuntive sono implementate tramite nuovi SO o nuovi dispositivi. Sono dispositivi chiusi e legacy, ovvero eseguono azioni sia nel data plane (instradamento di pacchetti) sia nel control plane (implementazione di protocolli come quello di instradamento, ovvero intelligenza di rete). L'idea principale del SDN è la divisione del **piano di controllo**, che diventa **centralizzato**, dal **piano dati**, che diventa **programmabile** e astratto rispetto al piano di controllo. Di conseguenza i singoli router diventano solo hardware e vengono programmati da un piano di controllo non più distribuito. Un **SDN switch** sarà in grado di fare solo forwarding sulla base di tabelle fornitegli dall'**SDN controller**, questo controllerà e programmerà gli switch SDN eseguendo le decisioni delle **Network Application** (software che esegue le funzionalità di rete come switching, routing, NAT, network slicing, traffic engineering, firewalling). Cosi per modificare le funzionalità di rete basta configurare il software sul controller senza dover agire sui singoli switch. Il controller necessita ora di avere una visione completa dello stato della rete. Può essere fisicamente su un server qualsiasi abbastanza potente insieme alle network application. Quindi le applicazioni sono disaccoppiate dall'hardware del piano dati, la rete è aperta. Uno **switch SDN**, nonostante sia meno complesso di un router IP, ha molte più potenzialità poiché basta scriverle nel software del controller. Il protocollo open flow si occupa dell'interfaccia southbound (comunicazione fra data plane e control plane), l'interfaccia northbound è invece quella fra controller e network application. Ogni switch SDN ha una **flow table** in cui ogni entry specifica l'azione che eseguirà il piano dati su pacchetti in ingresso appartenenti ad uno specifico flusso. Una entry contiene molti dati con cui un pacchetto può avere matching e non solo l'indirizzo di destinazione, può far scaturire diverse azioni (instradamento, processamento locale, scarto, invio al controller), ha un campo per contare il numero di bytes elaborati, un campo priorità e un timeout al termine del quale la entry si cancellerà. I messaggi del protocollo open flow sono: packet_in (S to C) se non ho matching per pacchetto ricevuto, flow_mode (C to S) per aggiungere o rimuovere o modificare una entry, flow_removed (S to C) per notificare che una entry è stata rimossa per timeout, port_status (S to C) per notificare il cambiamento di stato di una interfaccia, read_state (C to S) per collezionare statistiche su flow table, interfacce e specifici flussi. Posso inizialmente **configurare una flow table** in due modi:
Strategia reattiva: ogni primo pacchetto di un flusso causa l'azione del controller che crea l'entry nella flowtable. Se serve una regola che non ho allora la installo e ogni tanto la rinnovo. Tiene le flow tables leggere ma implica un ritardo (set up time) per ogni flusso e inoltre tutti gli switch lungo il percorso del primo pacchetto di un flusso faranno richieste al controller che potrebbe quindi intasarsi.
Strategia proattiva: le entries sono pre-installate del controller negli switch. La dimensione delle flow tables è molto grande ma non ho ritardi né setup time.
L'approccio generale utilizzato è un ibrido fra le due strategie. Il SDN ha molti vantaggi ma anche alcune criticità. Prima fra tutte il **bottleneck** del piano di controllo, servono scalabilità e sicurezza per cui si usa un **unico controller logico** che si declina su **multicontroller fisici**. È ancora aperta la ricerca riguardo decisioni legate ai controller fisici, come quanti devono essere, chi si deve occupare di quale switch, come si sincronizzano.
Le **flow tables** sono più complesse delle tabelle di routing IP poichè il matching è possibile su molti campi. Sono realizzate per questo in TCAM con circa 1000 entries, ma la ricerca sull'assegnamento delle entries agli switch è ancora aperta.
Per la **comunicazione switches/controller** non esiste ancora una specifica ufficiale, ma una possibile soluzione pratica è l'utilizzo della stessa infrastruttura del piano dati. Rimangono aperte le questioni riguardo la gestione di congestioni e la priorità dei messaggi di controllo.