

Reti

Internet

Internet è essenzialmente una rete di reti, ovvero una struttura composta da diverse componenti che si interconnettono e si scambiano dati. Può essere divisa attraverso due viste:

Vista “nuts and bolts”

La vista “nuts and bolts” è una vista a livello di componenti. Le componenti principali sono:

- **Host**: dispositivi di calcolo (computer, smartphone, tablet, ecc....) che eseguono delle applicazioni distribuite su Internet.
- **Switch di pacchetto**: dispositivi di rete (router, switch, etc...) che instradano i pacchetti di dati tra gli host e i collegamenti di comunicazione.
- **Collegamenti di comunicazione**: mezzi fisici che trasmettono i dati tra gli switch di pacchetto e gli host. Possono essere di diversi materiali, come fibra ottica, rame, onde radio, satellite, ecc.... La capacità di trasmissione di un collegamento si misura in bit al secondo ed è detta **bandwidth**.
- **Reti**: collezioni di host, switch di pacchetto e collegamenti di comunicazione che sono gestite da una stessa organizzazione. Esistono diversi tipi di reti, come le reti locali (LAN), geografiche (WAN), ecc.....
 - o **ISP**: le reti sono interconnesse tra loro tramite degli Internet Service Provider.
 - o **Protocolli**: ogni rete comunica tra di loro usando dei protocolli, che sono delle regole e delle convenzioni che definiscono il formato, l'ordine e le azioni dei messaggi scambiati. Alcuni esempi sono *HTTP*, *TCP/IP*, ec.. I protocolli sono definiti da degli standard di Internet, che sono dei documenti chiamati **RFC (Request for Comments)** e sono prodotti da un'organizzazione chiamata **IETF** (Internet Engineering Task Force)

Vista “services”

La vista services di Internet è quella dove internet è un'infrastruttura che fornisce dei servizi offerti ad applicazioni distribuite.

I servizi

Ci sono **applicazioni distribuite**, ovvero che coinvolgono più sistemi periferici e si scambiano reciprocamente dati. Le applicazioni per internet vengono eseguite su sistemi periferici, e non sui commutatori di pacchetto del nucleo della rete. I sistemi periferici collegati a internet forniscono delle **API** che specificano come il pezzo di software eseguito su un sistema periferico possa chiedere a Internet di recapitare dati a un altro specifico pezzo di software eseguito su un altro sistema periferico.

I protocolli

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

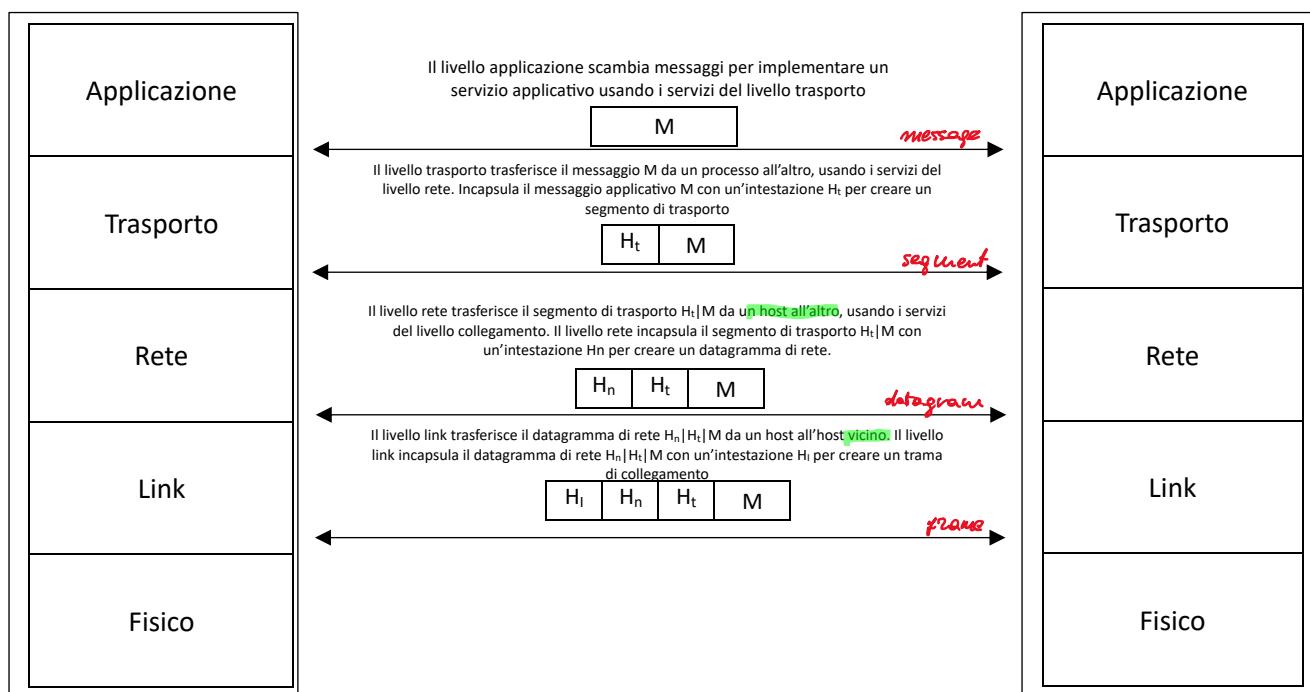
Protocol layers

Le reti sono sistemi complessi composti da molte parti, come host, router, collegamenti di vari mezzi, applicazioni, etc. Per organizzare l'architettura delle reti e la nostra discussione su di esse, si usa il concetto di **stratificazione**, che consiste nel suddividere le funzionalità di una rete in livelli gerarchici, ognuno dei quali offre un servizio specifico. Ha diversi vantaggi, come la *semplificazione della progettazione*, la *manutenzione* e l'*aggiornamento del sistema*, e la *trasparenza delle modifiche implementate e tra i livelli*.

Lo stack dei protocolli internet è composto da cinque livelli:

- Il livello **applicazione** supporta le applicazioni di rete, come il web (HTTP), la posta elettronica (SMTP) e la risoluzione dei nomi (DNS)
- Il livello **trasporto** gestisce il trasferimento dei dati tra i processi che eseguono sulle macchine sorgente e destinazione (TCP, UDP)
- Il livello **rete** si occupa dell'instradamento dei datagrammi dalla sorgente alla destinazione attraverso i router intermedi (IP).
- Il livello **collegamento** si occupa del trasferimento dei dati tra gli elementi di rete adiacenti, come gli host e i router (Ethernet, WiFi)
- Il livello **fisico** si occupa della trasmissione dei bit sul mezzo fisico di comunicazione, come il cavo o l'aria.

Services layers



Application Layer: DNS

Le persone e i dispositivi hanno diversi modi di identificarsi. Per esempio, una persona può avere un numero di previdenza sociale, un nome o un passaporto. Un host o un router può avere un indirizzo IP o un nome. Ma *come si fa a passare da un indirizzo IP a un nome, e viceversa?* Per risolvere questo problema, esiste il DNS, che sta per **Domain Name System**.

Il DNS è un sistema distribuito che gestisce una grande base di dati che associa i nomi dei domini ai loro indirizzi IP. Il DNS è implementato come un protocollo di livello applicazione, cioè usa i servizi del livello trasporto per comunicare tra gli host e i server DNS.

Quando un client vuole conoscere l'indirizzo IP di un nome di dominio, come www.amazon.com, deve interrogare diversi server DNS organizzati in una gerarchia. Il processo è il seguente:

- Il client interroga il **server DNS radice** responsabile del dominio .com
- Il client interroga il **server DNS .com** per trovare il server DNS responsabile del dominio amazon.com
- Il client interroga il **server DNS amazon.com** per trovare l'indirizzo IP del dominio www.amazon.com

Il DNS è il sistema che permette di associare i nomi dei domini ai loro indirizzi IP.

Categorie

Si possono suddividere in tre categorie:

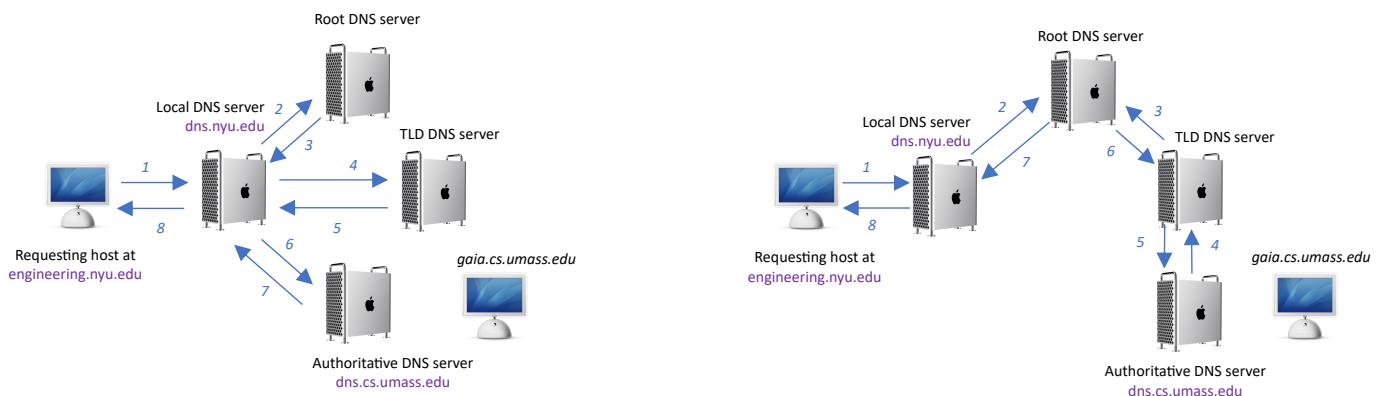
- **Root:** sono i server ufficiali e di ultima istanza che vengono contattati dai server DNS che non riescono a risolvere un nome. Sono fondamentali per il funzionamento di internet, perché senza di essi non si potrebbe accedere ai siti web
- **Primo livello:** sono i server responsabili dei domini di primo livello, come *.com*, *.org*, *.net*, ecc..... Gestiscono le richieste relative ai domini che appartengono alla loro categoria
- **DNS autoritativi:** sono i server propri di un'organizzazione, che forniscono le corrispondenze fra i nomi degli host e gli indirizzi IP dell'organizzazione. Possono essere mantenuti dall'organizzazione stessa o da un fornitore di servizi.

Richiesta DNS

Quando un host fa una richiesta DNS, la invia al suo server DNS locale, che gli restituisce la risposta, usando la sua cache locale di copie nome-indirizzo recenti, o inoltrando la richiesta alla gerarchia DNS per la risoluzione. Ogni ISP ha un server DNS locale, e si possono usare anche server DNS esterni locali.

La risoluzione dei nomi può avvenire in due modi:

- **Query iterativa:** il server DNS contattato risponde al server DNS locale con l'indirizzo IP del server da contattare.
- **Query ricorsiva:** il server DNS contattato si occupa di risolvere il nome e restituisce al server DNS locale l'indirizzo IP finale,. Questo alleggerisce il carico sul DNS locale, ma aumenta il carico sui livelli superiori della gerarchia.



Transport layer: TCP/UDP

Il layer di trasporto è il livello che si occupa di stabilire una **comunicazione logica** tra processi che si eseguono su host diversi, ovvero che fa sembrare che due processi siano direttamente connessi, anche se in realtà passano attraverso diversi nodi intermedi.

Per inviare un messaggio da un processo a un altro, il layer di trasporto lo divide in segmenti e li passa al livello di rete, che si occupa del trasporto fisico dei dati. Al lato ricevente, il layer di trasporto riunisce i segmenti in un messaggio, e lo passa al livello di applicazione.

I processi applicativi usano la comunicazione logica fornita dal livello di trasporto per scambiare messaggi, senza preoccuparsi dei dettagli dell'infrastruttura fisica utilizzata per trasportarli.

Dal lato del **mittente**, il livello di trasporto converte i messaggi che riceve da applicazione in segmenti, spezzando i messaggi applicativi in parti più piccole e aggiungendo a queste un'intestazione di trasporto. Il livello di trasporto passa quindi il segmento al **livello di rete**, dove viene incapsulato in un datagramma e inviato a destinazione.

Socket

Per identificare i processi coinvolti nella comunicazione viene usato un **socket**, che è una sorta di porta attraverso la quale un processo **manda o riceve messaggi**. Una volta che i messaggi sono spediti verso il layer di trasporto, questo si incarica di tutte le operazioni necessarie per consegnarli al processo destinatario; dalla socket quindi viene portato al processo applicativo corretto.

Procedimento

Sender:

- Gli viene passato un messaggio dal layer applicativo
- Determina i valori del segment header
- Crea il segmento
- Passa il segmento all'IP

Receiver:

- Riceve il segmento dall'IP
- Verifica i valori dell'header
 - Estraie il messaggio
- Porta il messaggio al layer applicativo attraverso il socket

Protocolli di trasmissione

I due principali protocolli di trasmissioni sono:

- **TCP**: protocollo che garantisce una consegna affidabile e ordinata dei dati. Implementa anche il servizio di **controllo di flusso e congestione**, che permette di regolare la velocità di trasmissione in base alle capacità del destinatario e della rete
- **UDP**: protocollo molto più semplice, non offre nessuna garanzia sulla consegna o l'ordine dei dati. È una semplice estensione di IP, che aggiunge solo la possibilità di identificare i processi tramite le socket.

TCP e UDP servono a estendere il servizio di consegna tra sistemi periferici di IP a un **servizio di consegna tra processi in esecuzione su sistemi periferici**.

Il passaggio da host-to-host a process-to-process è il **(de)multiplexing a livello di trasporto**; questi due protocolli forniscono un controllo di integrità basato sul riconoscimento di errori nelle intestazioni dei propri segmenti. UDP non garantisce che i dati inviati a un processo arrivino al destinatario; TCP fornisce un trasferimento dati affidabile grazie al controllo di flusso, numeri di sequenza, acknowledgement e timer. TCP converte il servizio inaffidabile fra sistemi periferici che è IP in un **servizio affidabile di trasporto dati fra processi**; UDP invece non regola il traffico.

Multiplexing/demultiplexing

Il multiplexing e il demultiplexing sono due operazioni che il layer di trasporto esegue per gestire i dati provenienti da o destinati a più socket.

Il **multiplexing** consiste nel prendere i dati da diverse socket, aggiungere un'intestazione di trasporto (*header*) che contiene le informazioni necessarie per il demultiplexing, e inviare i segmenti a livello di rete, da parte del mittente.

Il multiplexing, a livello di trasporto, richiede che le socket abbiano **identificatori unici**, e che ciascun segmento presenti campi che indichino la socket cui va consegnato il segmento.

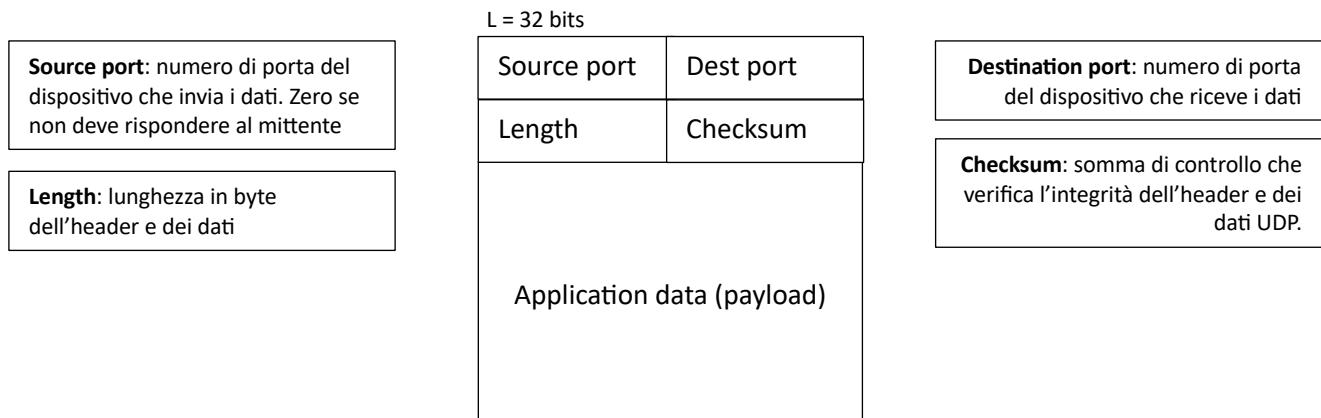
Il **demultiplexing** consiste nell' usare le informazioni presenti nell' header per consegnare i segmenti ricevuti alla socket corretta, da parte del ricevente.

- Nel caso di UDP (**protocollo senza connessione**), si basa sul numero di porta locale che viene specificato quando si crea la socket.

- Quando si crea un datagramma da inviare in una socket UDP, bisogna anche specificare l'**indirizzo IP** e il **numero di porta** del destinatario; questi due valori univocamente identificano una socket UDP.
- Quando l' host ricevente riceve un segmento UDP, controlla il numero di porta di destinazione nel segmento e lo indirizza alla socket con quel numero di porta.
- Segmenti IP/UDP con lo stesso numero di porta e indirizzo IP di destinazione, ma con indirizzi IP e/o numeri di porta di origine diversi, verranno indirizzati alla stessa socket nell' host ricevente
- Nel caso di TCP (**protocollo con connessione**), il demultiplexing si basa su una quadrupla di valori: di origine (indirizzo IP e numero di porta), e destinazione (indirizzo IP e numero di porta).
 - Consiste nell' usare tutti e quattro i valori per indirizzare il segmento alla socket appropriata.
 - Un server può supportare molte socket TCP simultanee: ogni socket è identificata dalla sua propria quadrupla. Ogni socket è associata a un diverso client in connessione.

UDP

È un protocollo di trasporto internet molto semplice ed essenziale. Offre un servizio di tipo **best effort**, cioè che non garantisce che i segmenti arrivino al destinatario, né che lo facciano nell'ordine corretto. È un protocollo senza connessione, cioè non prevede nessuna fase di negoziazione tra il mittente e il ricevente; ogni segmento UDP viene tratto in modo indipendente dagli altri.



A parte il multiplexing/demultiplexing e un basilare controllo degli errori, non aggiunge nulla a IP. Prende quindi i messaggi dal processo applicativo, aggiunge il **numero di porta di origine e di destinazione per il multiplexing/demultiplexing**, aggiunge altri due piccoli campi e passa il segmento risultante al livello di rete.

Visto che non esiste handshaking tra le entità di invio e di ricezione a livello di trasporto, UDP **non è orientato alla connessione**. Abbiamo quindi questi motivi per scegliere UDP e non TCP:

- **Controllo più fine a livello di applicazione su quali dati sono inviati e quando.** Spesso le applicazioni realtime vogliono i dati entro brevissimo tempo, e non si interessano ad eventuali piccole perdite di pacchetti
- **Nessuna connessione stabilita.** TCP usa 3-way-handshake, mentre UDP non introduce alcun ritardo nello stabilire la connessione. Per questo **DNS usa UDP**.
- **Nessuno stato di connessione.** UDP non mantiene niente, quindi risparmia spazio e può gestire più client
- **Meno spazio usato per l'intestazione del pacchetto.** TCP aggiunge 20 byte, mentre UDP aggiunge solo 8 byte.

Le applicazioni possono ottenere un trasferimento affidabile con UDP se l'affidabilità è insita nell'applicazione stessa.

Checksum

Serve a rilevare gli errori. Dal lato mittente, UDP effettua il **completamento a 1 della somma di tutte le parole a 16 bit del segmento**, e l'eventuale riporto finale viene sommato al primo bit; questo risultato viene poi messo nel campo checksum.

Dal ricevente, si sommano le tre parole iniziali e il checksum. Se non ci sono errori, allora **l'addizione darà 1111111111111111**; altrimenti, se un bit vale 0, sappiamo che è stato introdotto almeno un errore nel pacchetto.

Questo sistema viene usato perché non c'è garanzia che tutti i collegamenti tra origine e destinazione controllino gli errori, e comunque si potrebbe verificare un errore quando il pacchetto si trova nella memoria del router.

End-to-end

Dato che certe funzionalità devono essere implementate su base end-to-end, le funzionalità posizionate ai livelli inferiori possono diventare ridondanti o di scarso valore se confrontate con le stesse funzionalità offerte dai livelli superiori; UDP controlla la presenza di errori, ma non fa nulla per risolverli.

TCP

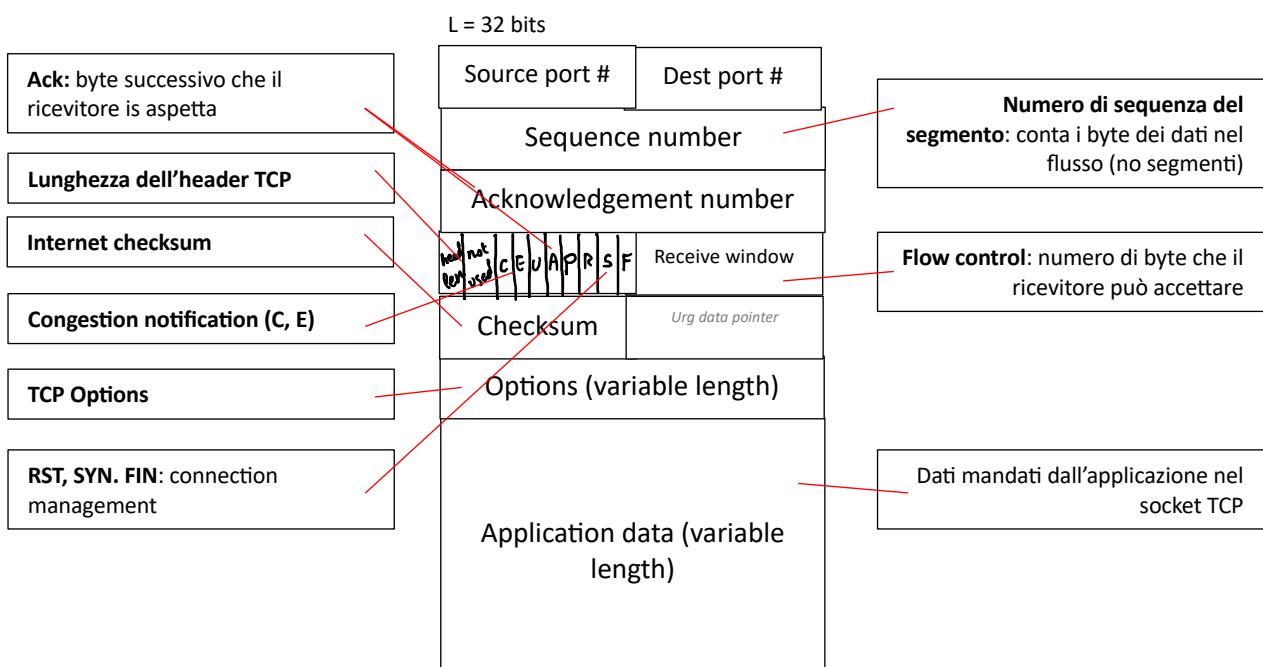
TCP è un protocollo di trasporto che garantisce la comunicazione affidabile e ordinata tra due entità (una sorgente e una destinazione). Trasmette i dati come un flusso continuo di byte, senza tenere conto dei confini dei messaggi. È inoltre in grado di **gestire la comunicazione bidirezionale**, cioè di inviare e ricevere dati sulla stessa connessione.

Utilizza il concetto di MSS (maximum segment size), che indica la dimensione massima dei segmenti che possono essere trasmessi.

TCP usa anche le **conferme cumulative**, che indicano il numero del prossimo byte atteso dal ricevitore. TCP sfrutta la tecnica del **pipelining**, che consiste nel trasmettere più segmenti senza attendere le conferme, aumentando così l'efficienza. La quantità di segmenti che possono essere trasmessi dipende dalla dimensione della finestra, che viene regolata dai meccanismi di controllo della congestione e del flusso di TCP. Offre un servizio **full duplex**, è a **punto a punto**, e le due parti devono effettuare l'**handshake**.

TCP è un protocollo orientato alla connessione, il che significa che prima di iniziare lo cambio di dati, la sorgente e la destinazione devono stabilire una connessione tramite una procedura di handshaking, che inizializza lo stato delle due entità.

È un protocollo a **controllo di flusso**, il che significa che la sorgente non trasmetterà dati a una velocità superiore a quella che il ricevitore può gestire.



TCP dirige i dati al buffer di invio della connessione, uno dei buffer riservato durante l'handshake a tre vie, da cui, di tanto in tanto, prevalerà blocchi di dati e li passerà al livello di rete. La **massima quantità di dati prelevabili** e posizionabili in un segmento viene limitata dalla dimensione massima di segmento (**MSS**). MSS viene impostato dopo aver determinato la lunghezza del frame più grande che può essere inviato a livello di collegamento dall'host mittente locale, la cosiddetta unità trasmissiva massima (**MTU**), e poi scegliendo un MSS tale che il segmento TCP, una volta incapsulato nel datagramma IP, stia all'interno di un solo frame a livello di collegamento.

TCP accoppia ogni blocco di dati del client con un'**intestazione**, formando **segmenti TCP** che vengono passati al sottostante livello di rete, dove sono incapsulati in **datagramma IP** che vengono poi immessi nella rete. Quando all'altro capo TCP riceve un segmento, i dati del segmento vengono **memorizzati nel buffer** di ricezione della connessione TCP.

L'applicazione legge il flusso di dati da questo buffer. Ogni lato della connessione presenta un proprio buffer di invio e di ricezione.

Numeri di sequenza e di ACK

TCP vede i dati come un flusso di dati non strutturati, ma ordinati. I numeri di sequenza si applicano al flusso di byte trasmessi e non alla serie di segmenti trasmessi. Il numero di sequenza di un segmento è il numero del primo byte del segmento nel flusso di byte.

TCP effettua l'ack solo dei byte fino al primo byte mancante nel flusso, quindi si dice che offre **acknowledgement cumulativi**.

Così'è che fa un host quando riceve segmenti fuori sequenza quindi? Può scartare i segmenti non ordinati, o mantenere i byte non ordinati e quindi aspettare quelli mancanti per colmare i vuoti.

Se il destinatario manda come ack un sequence number **n+1**, dice al mittente che *ha ricevuto con successo tutti i pacchetti fino a n*, e che sta aspettando n+1 invece. Il riscontro dei dati dal client al server viene trasportato in un segmento che a sua volta trasporta dati dal server al client: l' acknowledgement è **piggybacked** sul segmento dati dal server al client.

Timeout

Il timeout deve essere più grande del tempo di andata e ritorno sulla connessione, chiamato **RTT** (Round Trip Time):

$$\text{EstimatedRTT} = (1-\vartheta) * \text{EstimatedRTT} + \vartheta * \text{SampleRTT}, \text{ con } \vartheta=0.125$$

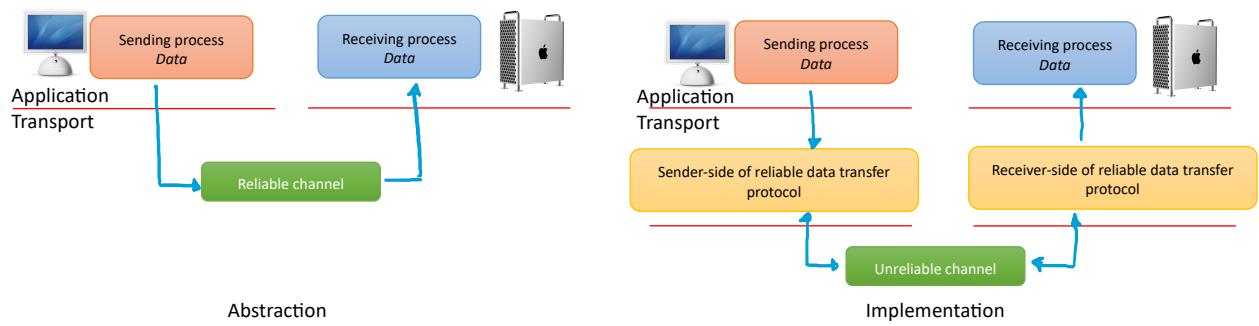
EstimatedRTT è una media mobile esponenziale ponderata dei valori di Sample RTT. **DevRTT** stima quanto generalmente RTT reale si discosta da Estimated RTT:

$$\text{DevRTT} = (1-\Omega) * \text{DevRTT} + \Omega * |\text{SampleRTT} - \text{EstimatedRTT}|$$

Inoltre la latenza è:

$$\text{Lat} = 2\text{RTT} + \text{dim_pacchetto}/R$$

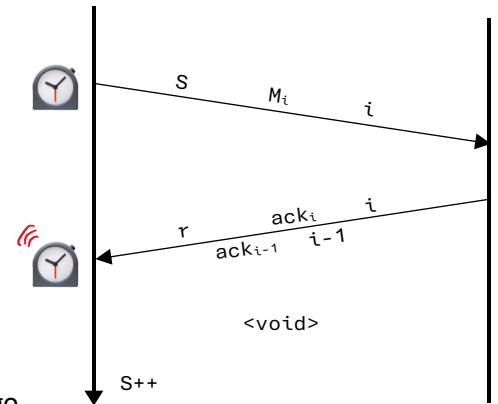
Reliable data transfer



Il trasferimento di dati affidabile è un processo che richiede un protocollo, cioè un insieme di regole, per garantire che **I dati inviati da una parte all'altra non si perdano**, non si corrompano e non si mescolino. Deve adattarsi quindi alle caratteristiche del canale di comunicazione, che può essere più o meno affidabile. Il mittente e il destinatario non conoscono lo stato l'uno dell'altro, a meno che non si scambino dei **messaggi** per informarsi.

Dobbiamo quindi usare un protocollo che corregga gli errori, e fanno uso di un determinato set di elementi:

- **Segmenti**: sono pezzi di dati che vengono inviati e ricevuti
- **Numeri**: servono per identificare i segmenti e per evitare le confusioni
- **Ack**: messaggi che confermano la ricezione di un segmento
- **Nack**: messaggi che segnalano la mancata ricezione di un segmento
- **Timer**: orologio che misura il tempo che passa tra l'invio e la ricezione di un segmento
 - Impostare una deadline per il timer però è un problema difficile: un timer troppo corto porterebbe a delle ritrasmissioni inutili, mentre invece un timer troppo lungo ferma la trasmissione per un periodo troppo lungo.



Raddoppio dell'intervallo di timeout

Se il timer **scade**, TCP ritrasmette il pacchetto, ma aspetta il doppio del tempo. Di conseguenza, **gli intervalli crescono esponenzialmente ad ogni ritrasmissione**.

Tutte le volte che avviene uno degli altri possibili avvenimenti, il timer viene resettato in base a EstimatedRTT e DevRTT.

Ritrasmissione rapida

Il mittente può rilevare la perdita di pacchetti ben prima che si verifichi l'evento di timeout, grazie agli **ACK duplicati** relativi a un segmento il cui ACK è già stato ricevuto dal mittente. Se il destinatario TCP riceve un pacchetto con un numero di sequenza superiore al prossimo numero di sequenza atteso, rileva un buco nel flusso di dati. Tale vuoto potrebbe essere dovuto a una perdita o a un riordinamento dei pacchetti all'interno della rete. Il destinatario **reinvia l'ack relativo all'ultimo byte di dati ricevuto correttamente**.

Se il mittente TCP riceve tre ack duplicati per lo stesso dato, considera questo evento come indice che **il segmento che lo segue è andato perduto**. Nel caso in cui siano stati ricevuti tre ack duplicati, il mittente effettua una **ritrasmissione rapida** rispedendo il segmento mancante prima che scada il timer.

Evento	Azione del ricevente TCP
Arrivo ordinato di segmento con numero di sequenza atteso. Tutti i dati fino al numero di sequenza sono riscontrati	ACK ritardato. Attesa fino a 500ms per l'arrivo di un altro segmento. Se non arriva, invia un ack
Arrivo ordinato di segmento con numero di sequenza atteso. Un altro segmento è in attesa di ack	Invia immediatamente un singolo ack cumulativo, riscontrando entrambi i segmenti
Arrivo non ordinato di segmento con numero di sequenza superiore a atteso. Rileva buco	Invia immediatamente un ack duplicato, indicando il numero di sequenza del prossimo byte atteso
Arrivo di segmento che colma parzialmente o completamente il buco nei dati ricevuti	Invia immediatamente un ack, ammesso che il segmento cominci all'estremità inferiore del buco.

Go-Back-N o Selective Repeat?

Anche se, per com'è fatto, si avvicina molto a Go-Back-N, esistono alcune differenze.

Molte implementazioni TCP memorizzano in un buffer i segmenti ricevuti correttamente, ma non in ordine. Una delle modifiche proposte a TCP è l'introduzione del **riscontro selettivo**, che consente al destinatario di mandare acknowledgement in modo selettivo per i segmenti non in ordine, anziché cumulativamente per l'ultimo segmento ricevuto senza errori e nell'ordine giusto.

Controllo di flusso

Se l'applicazione sul ricevente è lenta nella lettura dei dati può accadere che **il mittente mandi in overflow il buffer di ricezione** inviando molti dati troppo rapidamente.

TCP offre il **controllo di flusso** alle proprie trasmissioni per evitare che il mittente saturi il destinatario. Il servizio paragona la frequenza d'invio del mittente con quella di lettura del ricevente. I mittenti TCP possono anche essere rallentati dalla congestione nella rete IP.

Offre questo servizio facendo mantenere al mittente una variabile **finestra di ricezione** che fornisce al mittente un'indicazione dello spazio libero disponibile nel buffer del destinatario. Dato che TCP è *full duplex*, **i due mittenti hanno finestre di ricezione distinte**. Vediamo un esempio:

L'host sta inviando file all'host B. Definiamo le seguenti variabili:

- **LastByteRead**: numero dell'ultimo byte nel flusso di dati che il processo applicativo in B ha letto dal buffer
- **LastByteRcvd**: numero dell'ultimo byte che proviene dalla rete e che è stato copiato nel buffer di ricezione di B
- **RcvBuffer**: buffer di ricezione per la connessione
- **Rwnd**: finestra di ricezione

TCP non può mandare in overflow il buffer, quindi esegue:

$$\begin{aligned} \text{LastByteRcvd} - \text{LastByteRead} &\leq \text{RcvBuffer} \\ \text{Rwnd} &= \text{RcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead}) \end{aligned}$$

B comunica all'host A quanto **spazio disponibile** sia presente nel **buffer**, scrivendo il valore corrente di **rwnd** nel campo apposito dei segmenti che manda ad A. B inizializza rwnd con il valore di RcvBuffer e, ovviamente, deve tenere traccia di variabili specifiche per ogni connessione.

L'host A tiene traccia invece di *LastByteSent* e *LastByteAcked*; la differenza fra le due dice i byte mandati **per i quali si sta attendendo un ack**:

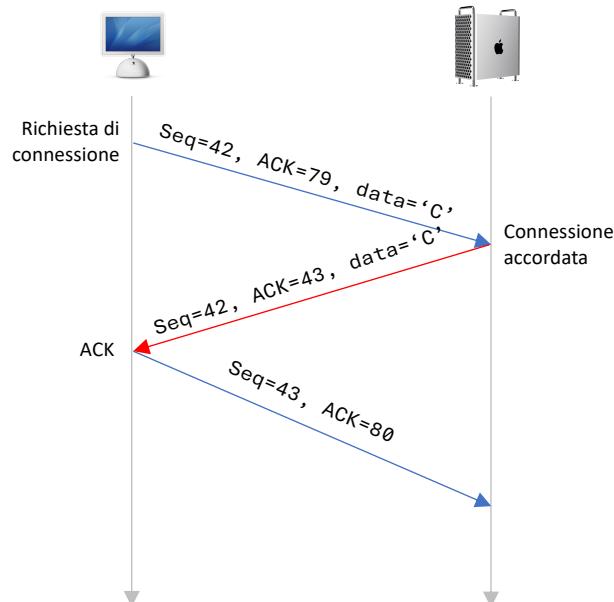
$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{rwnd}$$

L'host A quindi non manda in overflow il buffer di ricezione di B; le specifiche TCP richiedono che A continui a inviare segmenti con un byte di dati quando la finestra di ricezione di B è zero, per evitare che il trasferimento si blocchi completamente.

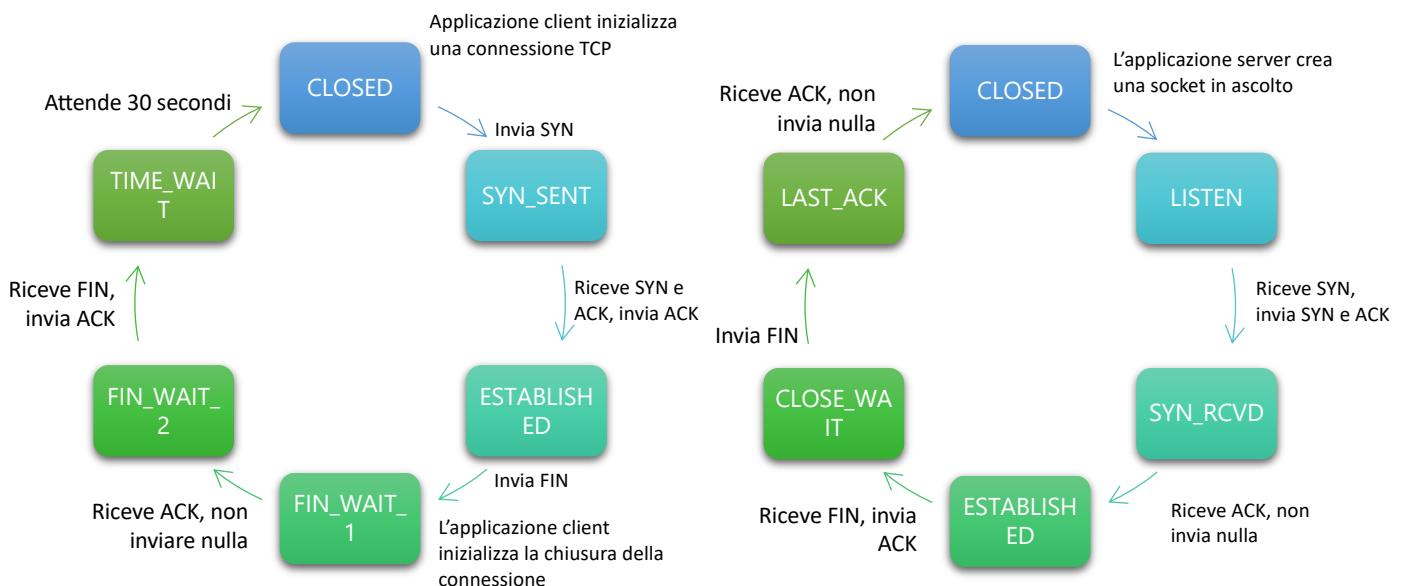
Gestione della connessione

Il TCP opera lo scambio di messaggi con un meccanismo chiamato **three-way handshake**:

1. Il TCP **lato client** invia uno speciale segmento al TCP **lato server**. Questo segmento non contiene dati, ma uno dei bit nell'header, **SYN**, è posto a 1; il segmento viene detto quindi **segmento SYN**. Il client sceglie a caso un numero di sequenza iniziale e lo pone nel campo *numero di sequenza* del segmento SYN
2. Quando il datagramma IP contenente il segmento SYN viene consegnato al server, questo estraе il segmento dal datagramma, **alloca I buffer e le variabili TCP** alla connessione e invia un segmento di **connessione approvata** al client TCP.
 - a. Il bit SYN è posto a 1, il campo ack assume il valore *client_isn+1*.
 - b. Il server sceglie il proprio numero di sequenza iniziale el o mette nel campo *numero di sequenza*; questo segmento è detto segmento **SYNACK**
3. Alla ricezione del segmento SYNACK anche il **client** **alloca buffer e variabili** alla connessione. Client invia un altro segmento al server, ponendo a *server_isn+1* il campo ack. Il bit SYN è a 0. Il campo dati può già contenere informazioni da mandare al server.
 - a. D'ora in poi il bit SYN sarà sempre a 0.



Quando si termina una connessione, le risorse vengono deallocate. L'host A manda un **comando di chiusura**, che forza l'invio di un segmento TCP speciale con bit FIN posto a 1. Quando B riceve questo segmento, manda un ack e spedisce il proprio **segmento di shutdown** con bit FIN posto a 1. A quindi risponde con un ack, e le risorse vengono deallocate.



Se un server riceve un pacchetto TCP SYN da un client il cui numero di porta/indirizzo IP non corrispondono ad alcuna socket nel host, allora invia al mittente un **messaggio di reset con il bit RST posto a 1**.

Controllo di congestione

TCP impone a ciascun mittente un limite alla velocità di invio sulla propria connessione in funzione della congestione di rete percepita. Il meccanismo di controllo di congestione TCP fa tener traccia agli estremi della connessione di una variabile aggiuntiva: la **finestra di congestione** (cwnd), che impone un vincolo alla velocità di immissione di traffico sulla rete da parte del mittente.

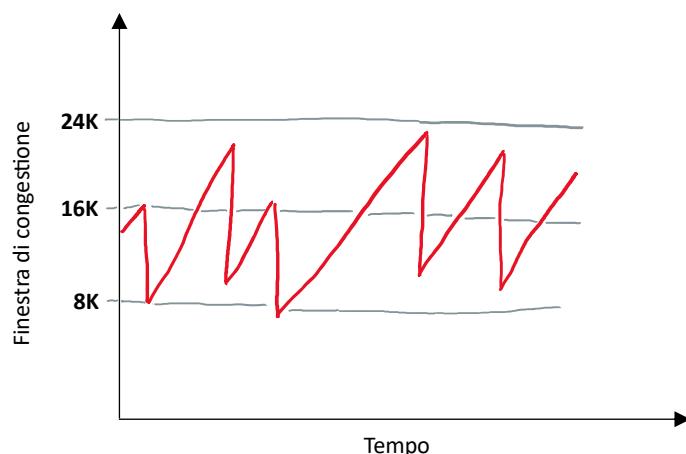
$$\text{LastByteSent} - \text{LastByteAcked} \leq \min(\text{cwnd}, \text{rwnd})$$

Definiamo **evento di perdita** per il mittente TCP l'occorrenza o di un **timeout** o della **ricezione di tre ack duplicati** da parte del destinatario. Un evento di perdita indica quindi che c'è congestione.

Se gli ack arrivano con frequenza relativamente bassa, allora la finestra di congestione verrà ampliata piuttosto lentamente. Se gli ack arrivano con frequenza più alta, la finestra di congestione verrà ampliata più rapidamente.

Dato che TCP utilizza gli acknowledgement per scatenare gli incrementi dell'ampiezza della finestra di congestione, si dice che TCP è auto-temporizzato.

Assumendo che le perdite siano indicate da triplo ack e non da timeout, il controllo di congestione consiste in un **incremento additivo** lineare della cwnd pari a **1 MSS per RTT**, e quindi di un decremento moltiplicativo che dimezza la cwnd in corrispondenza di un evento di triplice ACK duplicato. La maggior parte dei protocolli TCP usa l'algoritmo **Reno** che include fast recovery; l'originale Tahoe non lo includeva.



Principi quida

- Un segmento perso implica **congestione**, quindi i tassi di trasmissione del mittente TCP dovrebbero essere decrementati quando un segmento viene perso
 - Un **acknowledgement** indica che la rete sta consegnando i segmenti del mittente al ricevente, e quindi il tasso di trasmissione del mittente può essere aumentato quando arriva un acknowledgement non duplicato
 - Rilevamento della ragghezza di banda tramite gli ack

Algoritmo

Abbiamo tre componenti fondamentali.

Slow Start

Il valore di **cwnd** parte a 1 MSS e aumenta di 1 MSS ogni volta che il mittente riceve un ack. Ogni volta che il trasferimento va a buon fine, il mittente invia due segmenti per ogni ack ricevuto. **La velocità trasmissiva raddoppia ad ogni RTT.**

Se c'è un evento di timeout, il mittente reimposta *cwnd* a 1 e riparte da capo; pone il valore di una variabile **ssthresh** a $cwnd/2$, ovvero metà del valore che aveva la finestra di congestione quando la congestione è stata rilevata.

Quando *cwnd* arriva a valere *ssthresh*, TCP entra in una fase di **congestion avoidance**.

L'ultimo modo in cui una fase di slow start può terminare è se si ricevono **3 ack duplicati**: in questo caso, avvengono **ritrasmissione rapida e fast recovery**.

Congestion avoidance

Arrivato a $cwnd = ssthresh/2$, **si incrementa cwnd di 1 MSS ogni RTT**. L'algoritmo di congestion avoidance quando si verifica un timeout si comporta nello stesso modo di *slow start*: il valore di *cwnd* è posto uguale a 1 MSS, e il valore di *ssthresh* viene impostato alla metà del valore di *cwnd* al momento del timeout.

In caso di **acknowledgement duplicati**, TCP dimezza il valore di *cwnd* e imposta il valore di *ssthresh* pari al valore di *cwnd* al momento del ricevimento degli ACK. Infine, entra nello stato di fast recovery.

Fast recovery

Cwnd è incrementato di 1MSS per ogni ACK duplicato ricevuto relativamente al segmento perso che ha causato l'entrata di TCP nello stato di fast recovery. Infine, quando arriva un ACK per il segmento perso, TCP entra nello stato di *congestion avoidance* dopo aver ridotto il valore di *cwnd*.

I protocolli che possiamo usare

Il trasferimento di un segmento alla volta è l'attesa della sua conferma prima di una trasmissione successiva limita notevolmente le prestazioni.

I **protocolli a finestra scorrevole** possono trasmettere fino a W segmenti mentre aspettano la conferma del primo. La condizione per una trasmissione continua è che la finestra non si chiuda prima dell'arrivo della prima conferma.

Quando un segmento viene perso, cosa viene ritrasmesso?

- **Prima possibilità:** il protocollo **go-back-n** trasmette una serie di segmenti e li conferma in modo cumulativo. Se non riceve una conferma entro un certo tempo, ritrasmette tutti i segmenti non confermati. Questo protocollo è semplice da implementare, ma può essere inefficiente se ci sono molte perdite, perché ritrasmette anche i segmenti che sono stati ricevuti correttamente.
- **Seconda possibilità:** il protocollo **selective repeat** trasmette una serie di segmenti e li conferma in modo individuale. Se non riceve una conferma entro un certo tempo, ritrasmette solo il segmento perso. Questo protocollo è più efficiente, ma richiede un buffer al ricevitore per riordinare i segmenti prima di inviarli all'applicazione.

Go-Back-N

Il mittente trasmette più pacchetti senza aspettare un ACK, ma può avere al massimo N pacchetti nella pipeline. Introduciamo il **base** (numero di sequenza del più vecchio pacchetto che non ha ancora ricevuto ACK) e **nextseqnum** (più piccolo numero di sequenza non utilizzati). È un protocollo a **finestra scorrevole**.

Detto questo, abbiamo quattro intervalli di numeri di sequenza:

- **[0, base-1]:** pacchetti già trasmessi e che hanno ricevuto un ACK
- **[base, nextseqnum-1]:** pacchetti inviati, in attesa di ACK

- **[nextseqnum, base+N+1]**: numeri da usare per i pacchetti da inviare immediatamente, nel caso arrivassero dati dal livello superiore
 - o **N** = ampiezza della finestra (window size)
- **[base+N, +∞]**: non possono essere utilizzati finché non si ricevono ulteriori ACK

Il numero di sequenza di un pacchetto viene scritto in un **campo a dimensione fissa** di un pacchetto; il mittente GBN deve rispondere a 3 tipi di evento:

- **Invocazione dall'alto**: quando dall'alto si chiama `rdt_send()`, controlla se la finestra è piena. Se lo è, rimanda il dato all'applicazione, altrimenti manda il pacchetto. Nella realtà, spesso si può chiamare `redt_send()` solo se la finestra non è piena.
- **Ricezione di un ACK**: l'ACK del pacchetto con numero di sequenza n è **cumulativo**: tutti i pacchetti con numero di sequenza $\leq n$ sono stati ricevuti correttamente
- **Evento di timeout**: quando c'è un timeout il mittente reinvia tutti i pacchetti spediti che ancora non hanno ricevuto un ACK.

Il destinatario GBN deve invece eseguire i seguenti compiti:

- Se un pacchetto con numero di sequenza n viene ricevuto in ordine, manda ACK per quel pacchetto e consegna i dati al livello superiore
- In tutti gli altri casi, scarta il pacchetto e rimanda l'ACK dell'ultimo pacchetto ricevuto in ordine

Ripetizione selettiva

Si evitano ritrasmissioni non necessarie. Il destinatario è quindi obbligato a mandare degli **ack specifici** per i pacchetti ricevuti; si usa di nuovo una finestra di ampiezza N. Il destinatario SR manda ack per tutti i pacchetti ricevuti correttamente, sia in ordine che **fuori sequenza**: questi ultimi vengono memorizzati in un buffer, finché non sono stati ricevuti tutti i pacchetti mancanti.

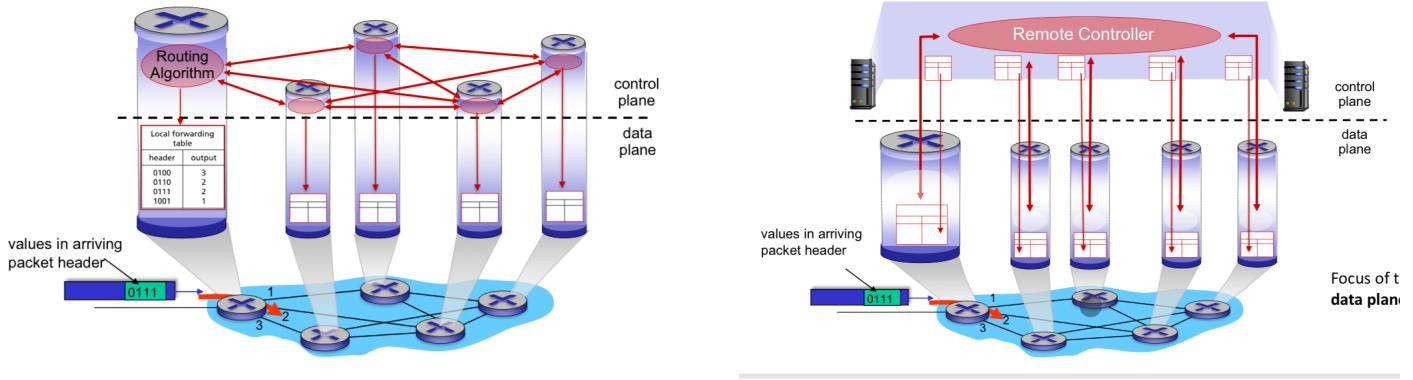
Il destinatario manda ack anche per pacchetti ricevuti con numero di sequenza sotto l'attuale base della finestra. Se il mittente non riceve questo ack, la sua finestra non può avanzare.

La finestra deve avere ampiezza inferiore o uguale alla metà dello spazio dei numeri di sequenza dei protocolli SR. Nella pratica, bisogna assicurarsi che un numero di sequenza non venga riutilizzato finché il mittente non sia sicuro che **ogni pacchetto mandato precedentemente con lo stesso numero di sequenza non si trovi più nella rete**. Si fissa un lasso di tempo, e lo si usa per determinare se possa trovarsi o meno nella rete.

Network layer

A livello di reti, la gestione del traffico e dell'instradamento dei dati coinvolge diversi piani e approcci:

- **Piano dati:** locale per ogni router, determina come i datagrammi vengono inoltrati dalle porte di ingresso a quelle di uscita
- **Piano di controllo:** instradamento gestito da algoritmi specifici in ciascun router. Due approcci comuni sono:
 - o **Piano di controllo per singolo router (tradizionale):** coinvolge componenti di algoritmo di instradamento individuali all'interno di ciascun router.
 - o **Software-Defined Networking (SDN):** il piano di controllo è decentralizzato dai router principali e spostato in un controller remoto, che *riceve informazioni da tutti i router, creando una visione completa della rete*. Successivamente, elabora tali informazioni per creare una tabella di inoltro, che viene distribuita ai router per l'instradamento.



Modello di servizio di rete

Il modello di servizio di rete può variare per datagrammi singoli o flussi di datagrammi. Le garanzie possono includere la **consegna con ritardo massimo**, o, per flussi, una banda minima, la consegna ordinata, o limitazioni sulla varianza dello spaziamento tra pacchetti.

A livello di rete, abbiamo un modello **best effort**, rappresentato semplicemente dal protocollo IP. All'interno del router, abbiamo delle input port e delle output port, un processore, e una **switching fabric**.

Il routing processor appartiene al *control plane*, mentre tutto il resto agisce sul piano hardware, inclusa la fabric.

Switching Fabric

Per interconnettere le porte di ingresso e uscita, viene utilizzata una high-speed switching fabric **implementata in hardware** nei router, che accelera il trasferimento dei dati tra le porte di ingresso ed uscita.

Per evitare l'accomodamento, la velocità della switching fabric deve essere ridimensionata intelligentemente, attraverso politiche di *accomodamento*, e *accomodamento di diversa priorità*.

La velocità della switching fabric viene detta **switching rate**, e, per essere ideale, dovrebbe essere $n \cdot r$ (r : rate, n : numero input/output).

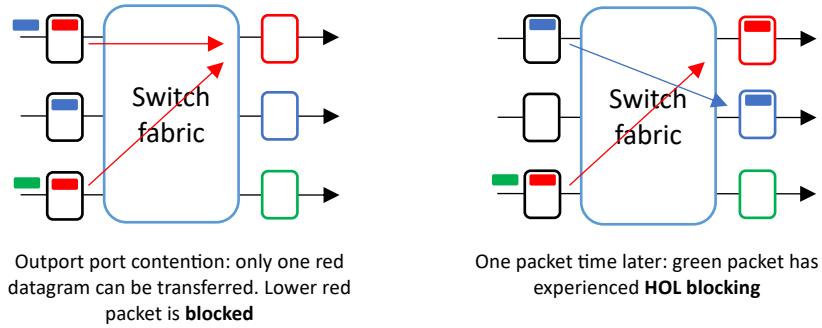
Funzioni porte di ingresso

Nella porta di ingresso, i datagrammi vengono ricevuti a livello fisico, e passano attraverso il link layer protocol per la decapsulazione. Successivamente, vengono sottoposti a lookup sulla tabella di inoltro e, in base ai risultati, vengono inoltrati alla porta corretta attraverso il **decentralised switching**.

Una volta arrivata al decentralised switching, succedono due operazioni: **lookup** (sulla base dei cambi dell'intestazione di livello 3 bisogna vedere sulla tabella di inoltro dove bisogna mandarlo), e **forwarding** (operazione di invio verso la porta corretta. Se la velocità di ingresso è maggiore rispetto alla velocità di switching, allora abbiamo **accomodamento**).

Se accade accomodamento, allora abbiamo un fenomeno che si chiama **head of the line blocking**.

Immaginiamo una situazione: abbiamo una serie di pacchetti che devono raggiungere una serie di porte: il pacchetto davanti a uno non può essere portato perché la porta di destinazione al momento è bloccata, ma il pacchetto subito dietro potrebbe essere portato. Questo prova un accomodamento, perché il segmento davanti non potrà muoversi fino a quando non avrà la sua porta libera.



Funzioni porte di output

Può avvenire accostamento anche nella output port, ma nella realtà, viene implementato un sistema di **priorità** che evita queste situazioni. Questo si basa su due politiche:

- **Dropping**: basate sulla priorità, determinano lo scarto di datagrammi a bassa priorità
- **Scaling**: gestiscono la velocità di trasmissione, garantendo un equilibrio tra la capacità del sistema e il flusso dei dati.

Il datagramma IP

Il datagramma è strutturato con vari campi che contengono informazioni fondamentali per il trasporto dei dati:

1. **Versione (Ver – 4 bit)**: specifica versione in uso
2. **Lunghezza intestazione (head len – 4 bit)**: indica lunghezza intestazione in parole di 32 bit
3. **Tipo di servizio (type of service – 8 bit)**: fornisce informazioni su qualità servizio e gestione traffico
4. **Lunghezza totale (Length – 16 bit)**
5. **Identificazione (16 bit)**: identifica univocamente il datagramma all'interno di un flusso di dati
6. **Flags (flgs – 3 bit) e Offset di Frammentazione(13 bit)**: il campo di flag include i bit “non frammentare” (DF), “frammentare” (MF), e il “Bit di Frammento” (offset) per la gestione del frammento del datagramma
7. **Time to Live (TTL – 8 bit)**: numero massimo di router attraverso i quali il datagramma può passare prima di essere scartato
8. **Upper Layer Protocol**: specifica il protocollo di livello superiore al quale il datagramma deve essere consegnato
9. **Checksum (16 bit)**: verifica l'integrità dell'intestazione del datagramma
10. **Indirizzo IP sorgente (32 bit)**: indica l'indirizzo IP del mittente
11. **Indirizzo IP di destinazione (32 bit)**: indica l'indirizzo IP del destinatario
12. **Opzioni**: questo campo può contenere opzioni aggiuntive

Ver	Head len	Type service	Length
16-bit identifier		Flgs	Fragment offset
Time to live		Upper layer	Header checksum
Source IP address			
Destination IP address			
Options (if any)			
Payload data (variabile length, typically a TCP or UDP segment – 20 bit)			

13. Payload data: rappresenta dati trasportati dal datagramma con lunghezza variabile; solamente TCP o UDP

Per contare la lunghezza di intestazioni, dobbiamo usare i dati di payload: 20 bit per l'IP, e poi 20 bit per l'UDP o il TCP.

Indirizzamento IP e sottoreti

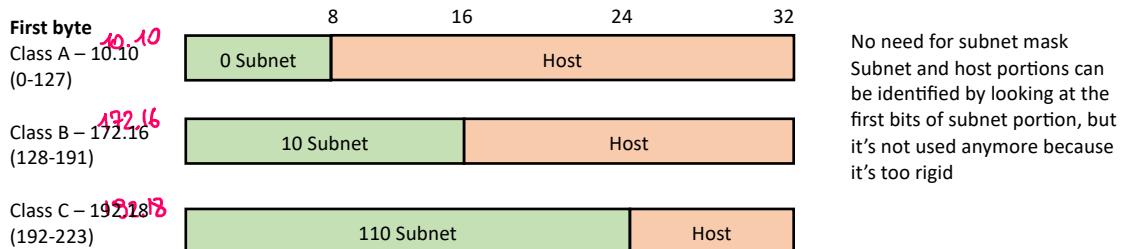
Le sottoreti sono gruppi di interfacce fisicamente raggiungibili senza l'Intermediazione di un router. Ciascun indirizzo IP è composto da una parte di sottorete (**bit più significativi**), e una parte di host (**bit meno significativi**). Tutte le interfacce nella stessa sottorete condividono la stessa parte di sottorete, ma hanno host diversi. La parte di host viene rappresentata con un */number*, il quale indica esattamente quanti bit io considero per la parte di sottorete, e quanto per la parte di host. Per rappresentare le subnet, abbiamo due metodi:

Classless

Chiamata anche **CIDR** (Classless InterDomain Routing), utilizza la notazione a.b.c.d/x per indicare la lunghezza dell'indirizzo di sottorete. Le sottoreti possono avere un indirizzo di **lunghezza arbitraria**, ovvero che viene definita dallo slash. Se abbiamo /23, può essere rappresentato anche con la **subnet mask**, ovvero un indirizzo dove i bit sono: tutti uno per la parte subnet, e tutti zero per la parte host.

Classful

Non richiede una subnet mask; l'identificazione della classe dell'indirizzo di sottorete avviene guardando i primi bit, ma è meno flessibile. In base a quanti bit sono riservati per la subnet e l'host, si può capire se un indirizzo appartiene a una certa classe o a un'altra.



Indirizzi IP speciali

Abbiamo vari indirizzi IP che possono essere definiti "speciali" per la loro struttura:

- **Sottorete:** parte host con tutti zero
- **Direct Broadcast Address:** parte host con tutti uno, utilizzato per il *broadcast verso tutti gli host nella stessa subnet*.
- **Limited Broadcast Address:** utilizzato per il broadcast nella stessa subnet, senza superare i router.

Assegnazione degli indirizzi IP

Gli indirizzi IP possono essere assegnati staticamente, o dinamicamente, tramite **DHCP** (Dynamic Host Configuration Protocol), quando l'host si connette alla rete. Assegna dinamicamente indirizzi IP agli host, con un tempo di assegnazione e rassegnazione. Per fare questo, utilizza quattro messaggi:

- **DHCP Discover:** convogliato verso tutti i dispositivi nella rete con porta **67**, e include anche il transaction ID, importante per identificare univocamente questa richiesta. L'host identifica il server con cui interagire
- **DHCP Offer:** il server manda il messaggio DHCP offer. Ciascun messaggio di offerta contiene l'ID di transazione del messaggio di identificazione ricevuto, l'IP proposto al client, la subnet mask, e la durata della concessione. Dal server DHCP al client, manda:

- *Src: DHCP server:67*
- *Dest: 255.255.255.255:68*
- *Your address: l' address dell'host*
- *Transaction ID: lo stesso del precedente*
- *Lifetime (per quanto questo indirizzo è valido)*
- *DHCP Request:* il client sceglie tra le offerte dei server e risponde con un messaggio che riporta i parametri di configurazione. Dall'host al server, manda:
 - *Src: 0.0.0.68*
 - *Dest: 255.255.255.255:67*
 - *Address: nuovo address*
 - *Transaction ID: nuovo Transaction ID*
 - *Lifetime: lo stesso di prima*
- *DHCP Acknowledgement:* conferma che l'indirizzo è dell'host

Subnetting e Route Aggregation

Queste due funzionalità permettono di svolgere un compito alquanto particolare:

- **Subnetting** permette sottoreti di lunghezza variabile, utile per l'aggregazione delle rotte
- **Route Aggregation** semplifica la gestione delle rotte, particolarmente importante quando un'organizzazione cambia ISP

Queste due operazioni sono importanti quando ci troviamo in un caso dove, per esempio, un router si interfaccia a due ISP. Se una organizzazione si sposta da un ISP a un altro, bisogna cambiare da quale organizzazione arriva. Grazie al **Longest Prefix Match**, posso supportare i cambi di ISP praticamente. Quindi, se io ho due ISP con, nella maggior parte, lo stesso indirizzo ma alla fine cambia, grazie al LPM posso indirizzarlo correttamente.

IPv4/IPv6

IPv4 ha uno spazio di indirizzamento a 32-bit, il che vuol dire che è abbastanza limitato; infatti, è già stato superato il limite nel 2011. Per questo, è stato introdotto **NAT** (Network Address Translation), che è tuttora in uso per affrontare l'esaurimento degli indirizzi IP. Nonostante l'esistenza di versioni più recenti di IP come **IPv6**, con uno spazio di indirizzamento a 128-bit, NAT+IPv4 continua ad essere utilizzato in attesa di una transizione completa.

I router abilitati al NAT appaiono all'esterno come un **unico dispositivo** con un unico indirizzo IP; nasconde i dettagli della rete domestica al mondo esterno. Se tutti i datagrammi in arrivo al router NAT dalla rete geografica hanno lo stesso indirizzo IP di destinazione, si usa una **NAT translation table** nel router NAT, e include nelle righe di tale tabella i numeri di porta oltre che gli indirizzi IP; in tal modo, si saprà anche soltanto attraverso l'indirizzo e porta, a quale router deve andare.

UpnP

L'Universal Plug&Play è un protocollo che consente a un host di individuare e configurare un NAT vicino; viene usato per fornire il servizio di **attraversamento del NAT**. Con UpnP un'applicazione in esecuzione in un host può richiedere una corrispondenza NAT tra i propri, per un qualsiasi numero di porta pubblico. Se il NAT accetta la richiesta, e crea una corrispondenza, allora i nodi esterni possono iniziare connessioni TCP verso l'indirizzo IP/porta pubblica.

Quindi, **UpnP consente a un host esterno di iniziare sessioni di comunicazione con host con NAT**.

ICMP

ICMP significa **Internet Control Message Protocol**, e i messaggi ICMP vengono trasportati come payload di IP, esattamente come i segmenti TCP o UDP.

I messaggi hanno un campo **tipo**, e un campo **codice**, e contengono l'intestazione e i primi 8 byte del datagramma IP che ha provocato la generazione del messaggio, in modo che il mittente possa identificare il datagramma che ha generato l'errore.

Livello di collegamento

Nel contesto del livello di collegamento, i nodi possono essere host o router, mentre i canali di comunicazione che collegano nodi adiacenti lungo il percorso di comunicazione sono definiti **link**, che possono essere cablati o wireless, compresi nelle reti LAN. A questo livello, i dati vengono incapsulati in frame che trasportano il datagramma.

I datagrammi sono trasferiti utilizzando differenti protocolli di collegamento su varie tipologie di link. Ogni protocollo di collegamento offre *servizi diversi*; usiamo un esempio per capirlo meglio:

La diversità è paragonabile a un viaggio da Princeton a Losanna, dove diversi mezzi di trasporto rappresentano protocolli di livello di collegamento. In questa analogia:

- Il turista è il datagramma
- Il segmento di trasporto è il collegamento di comunicazione
- La modalità di trasporto è il protocollo di livello di collegamento

I servizi offerti

Il livello di collegamento fornisce diversi servizi, tra cui:

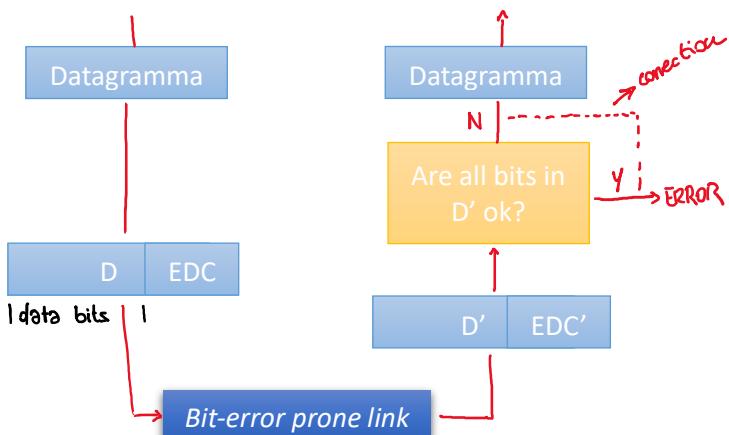
- **Incapsulamento del datagramma in un frame** (aggiungendoci header e trailer)
- **Accesso al canale** (nel caso di un mezzo condiviso)
- **Consegna affidabile tra nodi adiacenti** (particolare utile su link con bassi tassi di errore)
- **Controllo del flusso** (che controlla il flusso tra nodi riceventi e mittenti adiacenti)
- **Rilevazione e correzione degli errori** (errori causati dall'attenuazione del segnale, può occorrere una ritrasmissione del segnale)
- **Half-duplex o full-duplex** (a seconda che i nodi ai due estremi del link possano trasmettere contemporaneamente o meno)

Rilevazione e correzione degli errori

Si impiegano bit di rilevamento e correzione degli errori (**EDC**, Error Detection and Correction bits) per garantire l'affidabilità dei dati.

Questi bit, aggiunti ai dati stessi, forniscono un meccanismo di controllo degli errori, tipicamente applicato anche ai campi di intestazione.

Importante notare che la rilevazione degli errori non è sempre completamente affidabile, poiché possono verificarsi situazioni in cui alcuni errori sfuggono alla verifica.



Protocolli ad accesso multiplo

Dobbiamo considerare due tipi di collegamento:

- **Point-to-point**: si verifica quando c'è un collegamento diretto tra due nodi, ad esempio, un collegamento Ethernet tra un host e uno switch
- **Broadcast/cavo condiviso**: tipico dell'Ethernet tradizionale, o di tecnologie più recenti come il WiFi

Quando parliamo di protocolli ad accesso multiplo, ci troviamo davanti a un problema: *se abbiamo un canale di trasmissione condiviso dove due o più nodi decidono di trasmettere simultaneamente, si crea*

interferenza. Abbiamo quindi un problema di **collisione**, quando un nodo riceve due o più segnali contemporaneamente.

Per gestire questo scenario, è necessario un algoritmo distribuito che stabilisca quando ciascun nodo può trasmettere. Le comunicazioni su come condividere il canale deve avvenire utilizzando il canale stesso.

Abbiamo tre classi generiche di tipi di protocolli:

- **Channel partitioning**: divide il canale in pezzi più piccoli, e alloca ciascun pezzo a un nodo per uso esclusivo
- **Random access**: il canale non è diviso, ci possono essere collisioni e quando questo occorre, cerca di correggere
- **Taking turns**: i nodi si alternano, ma i nodi che devono mandare più dati ci metteranno più tempo

Random access protocols

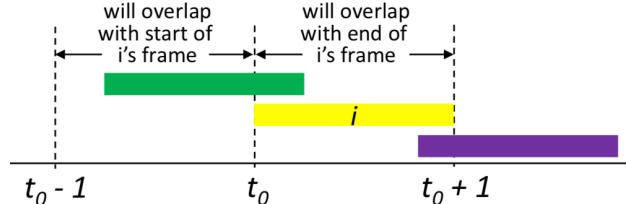
Non c'è alcuna coordinazione preventiva tra i nodi. Quando un nodo ha un pacchetto da inviare, lo trasmette alla *massima velocità del canale*, denominata **frequenza dati R**. Tuttavia, se due o più nodi trasmettono contemporaneamente, si verifica una **collisione**.

I protocolli ad accesso casuale specificano come rilevare le collisioni e come recuperarle e, ad esempio, mediante ritrasmissioni ritardate. Alcuni esempi includono *ALOHA*, *slotted ALOHA*, *CSMA*, *CSMA/CD*, e *CSMA/CA*.

Pure ALOHA

Il protocollo ALOHA, nella sua versione unslotted, opera senza la suddivisione di slot di tempo, e senza sincronizzazione tra i nodi. Quando un frame arriva, viene trasmesso immediatamente senza dover attendere l'inizio di uno slot specifico.

Tuttavia, la probabilità di collisione aumenta in assenza di sincronizzazione, poiché un frame trasmesso al tempo t_0 può collidere con frame inviati nei tempi t_0-1 e t_0+1 . Questo accade perché l'intervallo tra la fine di un frame e l'inizio di un altro può sovrapporsi.



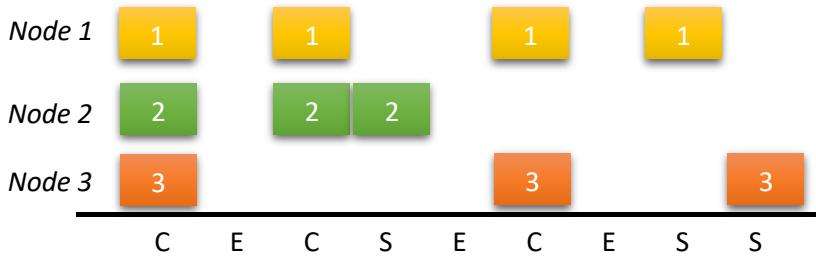
L'efficienza si attesta attorno al **18%**.

Slotted ALOHA

Opera considerando **slot di tempo di dimensioni uguali**, in cui i nodi possono trasmettere i loro frame. L'efficienza di questo protocollo è influenzata da diverse ipotesi, come la *dimensione uniforme dei frame*, la *suddivisione del tempo in slot* e la *sincronizzazione tra i nodi*.

Quando un nodo ha un nuovo frame da trasmettere, lo invia nel successivo slot disponibile. In caso di collisione, causata da più nodi che trasmettono contemporaneamente, i nodi rilevano l'evento e ritrasmettono il frame in slot successivi, con una probabilità p fino al successo.

Questo approccio ha vantaggi, come la *possibilità per un singolo nodo attivo di trasmettere continuamente alla massima velocità del canale*, e la *struttura altamente decentralizzata*. Tuttavia, ha la *presenza di collisioni* che portano allo spreco di slot, la presenza di slot inattivi, e la necessità di sincronizzazione dell'orologio tra i nodi.



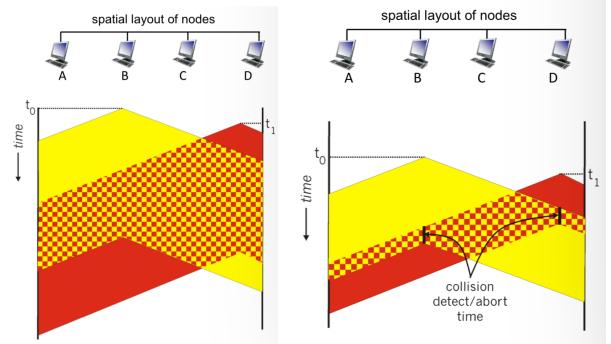
N nodi con tot frame da mandare, ognuno trasmette in slot con probabilità p
 N = nodi
 Probabilità che il nodo dato ha un successo nello slot: $p(1-p)^{N-1}$
 Total expected overall utilisation = $Np(1-p)^{N-1}$
 Efficienza massima: find p that maximises $Np(1-p)^{N-1}$
 At best: channel used for useful transmissions **37%** of the time

CSMA (Carrier Sense Multiple Access)

È un protocollo di accesso al canale che integra la “carrier sense” come principio guida. Esistono due varianti principali: la versione semplice, e **CSMA/CD**.

Nella **CSMA semplice**, una stazione ascolta il canale prima di trasmettere. Se il canale è libero, la stazione trasmette l'intero frame; se il canale è occupato, la stazione ritarda la trasmissione.

CSMA/CD include il rilevamento delle collisioni. Le collisioni vengono individuate entro un breve periodo, e le trasmissioni coinvolte vengono interrotte, riducendo lo spreco del canale.



Possono comunque verificarsi *collisioni*; la propagazione del segnale introduce ritardi che potrebbero impedire a due nodi di rilevare le rispettive trasmissioni appena iniziata.

Con una *propagation delay*, due nodi potrebbero non ascoltarli tra di loro con le trasmissioni appena iniziata. E quando avviene una collisione, tutto il tempo di trasmissione del pacchetto è perduto.

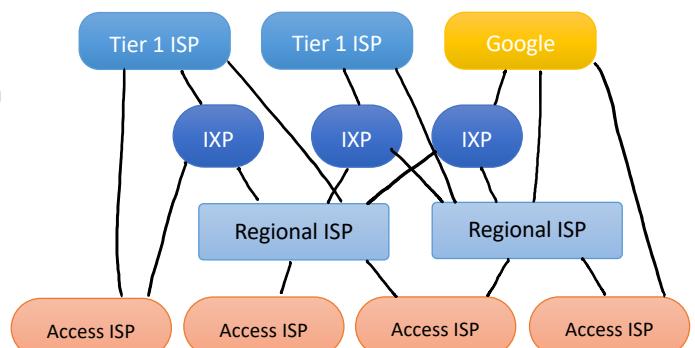
La struttura

Internet ha una struttura gerarchica composta da tre livelli principali:

- **Network edge**, dove si trovano gli host (client e server) che usano internet per comunicare tra loro
- **Reti d'accesso**, dove si trovano le reti che permettono agli host di collegarsi a internet, sia con cavi che con onde radio
- **Network core**, dove si trovano i router che interconnettono le diverse reti e fanno da ponte tra gli host

Esistono dei nodi chiamati **Internet Exchange Point (IXP)**, dove i fornitori di servizi internet si connettono tra loro per scambiarsi il traffico. Alcuni ISP si connettono anche a reti di livello superiore, chiamate **tier 1**, per avere una maggiore copertura; questi sono i **commercial ISPs**.

Infine, ci sono delle reti specializzate chiamate **content provider network**, che offrono contenuti e servizi agli utenti bypassando le reti di tier 1.



Network edge

Il network edge è dove ci sono gli **host**, ovvero i dispositivi che si connettono alla rete e si scambiano dei dati. Quando un host vuole mandare un messaggio a un altro host, deve prima spezzare il messaggio in tanti pezzi chiamati **pacchetti**; ognuno di questi ha una certa lunghezza, misurata in **bit**.

Per mandare un pacchetto da un host a un altro, ci vuole un certo tempo, che dipende dalla velocità della rete; questo si chiama **ritardo di trasmissione**; viene calcolato dividendo la lunghezza del pacchetto per la velocità della rete.

Più ho **bandwidth**, meno ho ritardo di trasmissione.

$$R_t = \frac{L \text{ bits}}{R \text{ bits/sec}}$$

I messaggi lunghi sono suddivisi in pacchetti, che viaggiano attraverso collegamenti e commutatori di pacchetto.

Le reti di accesso

Le access network sono le reti che connettono fisicamente un sistema al suo **edge router**, che è il primo router sul percorso dal sistema d'origine a un qualsiasi altro sistema di destinazione collocato al di fuori della stessa rete di accesso.

Accesso residenziale

Può essere per esempio DSL, via cavo o fibra ottica. Il modem DSL dell'utente usa la linea telefonica esistente per scambiare dati con un **Digital Subscriber Line Access Multiplier (DSLAM)** che si trova nella centrale locale della compagnia telefonica.

Il modem DSL residenziale converte i dati digitali in toni ad altra frequenza per poterli trasmettere alla centrale locale sul cavo telefonico; tutti i segnali analogici in arrivo dalle abitazioni vengono riconvertiti poi in formato digitale nel DSLAM.

Le linee telefoniche residenziali trasportano contemporaneamente dati e segnali telefonici codificando lì in tre bande di frequenza non sovrapposte:

- **Canale di downstream** verso l'abitazione ad alta velocità, con banda tra 50 kHz e 1 MHz
- **Canale di upstream** verso il DSLAM a velocità media, con banda tra 4 e 50 kHz
- **Canale telefonico** ordinario a due vie, nella banda tra 0 e 4 kHz.

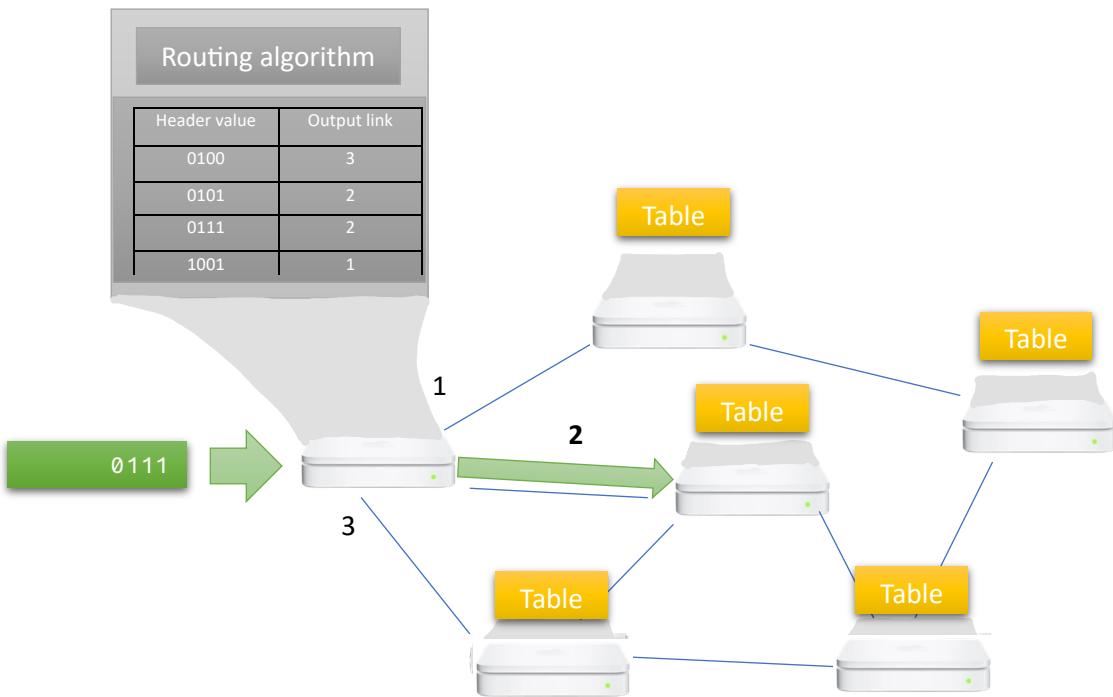
Network core

Il network core è dove ci sono i **router**, cioè i dispositivi che collegano tra loro le varie reti.

Packet switching

La rete funziona in modo che i pacchetti di dati vengano mandati da un router all'altro, fino a raggiungere la destinazione.

I router hanno all'interno una **local forwarding table**, che indica come inoltrare i pacchetti in base al loro contenuto. Abbiamo pertanto una serie di operazioni che vengono svolte:



Visto che l'header corrisponde al valore **0111**, e ha come output link **2**, allora verrà presa la strada rappresentata dal numero 2. Internet ha parecchi protocolli di instradamento specifici che usa per impostare automaticamente le tabelle di inoltro.

Forwarding

Nel momento in cui arriva un pacchetto, si vede l'intestazione di esso, si controlla la tabella e si cerca una **corrispondenza tra il valore e il collegamento in uscita**. È un'operazione locale: muove i pacchetti in arrivo dall' input link del router all' appropriato output link.

Routing

Permette di costruire la “strada” tra sorgente e destinazione; ogni nodo all'interno della strada **effettua l'operazione di forwarding**.

Store-and-forward

Lo store-and-forward è un modo di funzionare delle reti a commutazione di pacchetto, che ha delle conseguenze su queste reti.

Supponiamo di avere *3 nodi diversi* (sorgente, commutatore e destinazione) e *3 pacchetti da mandare*. Prima di poter mandare un pacchetto al nodo successivo, bisogna aspettare che il pacchetto sia arrivato tutto al nodo commutatore.

I pacchetti che arrivano si mettono in fila nel **buffer** che li tiene in attesa. Se la memoria si riempie, i pacchetti in eccesso vengono scartati; questo può causare dei problemi, poiché si perderà tempo ad aspettare che i pacchetti si mettano in fila e si elaborino.

Si consideri il caso generale della trasmissione di un pacchetto dalla sorgente alla destinazione su un percorso di **N collegamenti** ($N-1$ router), ognuno con velocità di trasmissione R ; il ritardo di trasmissione sarà:

$$R_t = N * \frac{L \text{ bits}}{R \text{ bits/sec}}$$

È ideale?

È adatta per i dati che non sono costanti, ma hanno delle variazioni nel tempo. Per esempio, se ho dei dati da andare e altre volte no. In questo caso, la commutazione di pacchetto permette di **condividere le risorse** della rete con altri utenti senza dover riservare un canale fisso.

La commutazione di pacchetto è più semplice da realizzare, perché non richiede una configurazione di chiamata tra i nodi.

Può invece causare dei **problemi di congestione**, cioè un eccesso di traffico che rallenta la rete e fa perdere dei pacchetti. Ci vogliono quindi dei **protocolli** che assicurino un trasferimento affidabile e che controllino la congestione.

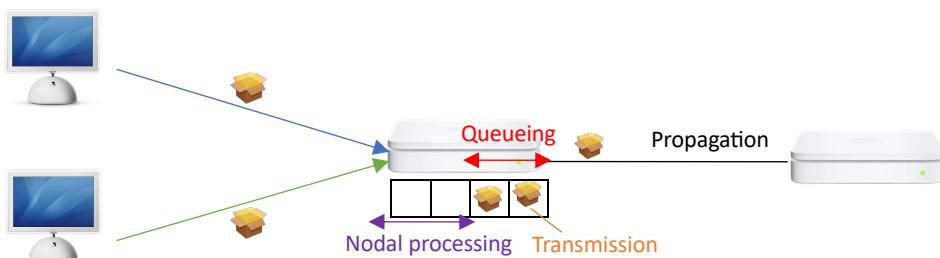
Altro problema è come garantire un comportamento simile a quello della commutazione di circuito, ovvero una *qualità costante e prevedibile della comunicazione*. Ci sono delle tecniche che cercano di avvicinarsi il più possibile, usando per esempio delle priorità o delle riserve di banda per i pacchetti.

Le reti a pacchetto sono molto più efficienti, perché sfruttano meglio le risorse a disposizione. Se pensiamo che gli utenti non usano la rete tutto il tempo ma solo per il 10% del tempo, con la commutazione di pacchetto possiamo farne comunicare 10 con le stesse risorse che con la commutazione di circuito ne servirebbero solo per uno.

Packet delay and loss

I pacchetti **si accodano** nel buffer del router, in attesa di essere trasmessi. La lunghezza della coda cresce quando il *arrival rate to link*, temporaneamente, eccede l' *output link capacity*.

La **perdita** di questi pacchetti avviene quando la memoria per tenere questi pacchetti accodati si riempie.



$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

- Nodal processing (d_{proc})

- È il tempo che impiega un router a processare l'intestazione del pacchetto, cioè a leggere le informazioni sul mittente, destinatario e protocollo del pacchetto.
- **Dipende dalla velocità del processore del router e dal numero di operazioni che deve eseguire.**
- Vede quali sono gli errori di bit, di solito determina l'output link, e tipicamente è meno di un microsecondo

- Queueing delay (d_{queue})

- È il tempo che il pacchetto trascorre in coda nei buffer del router, in attesa di essere trasmesso al link successivo.
- **Dipende dal tasso di arrivo dei pacchetti al link, dalla capacità del link di uscita, dalla dimensione della coda e dal numero di server o link disponibili.**
- È il tempo che bisogna aspettare all' output link per la trasmissione. Tipicamente è nell'ordine dei millisecondi

$$d = \frac{L \alpha}{R}$$

$\alpha = \frac{\text{pacch.}}{\text{sec}}$
 $L = \text{bit}$
 $R = \text{bps}$

- **Transmission delay (d_{trans})**
 - È il tempo che impiega a trasmettere il pacchetto dal router al mezzo di trasmissione, cioè a inviare tutti i bit del pacchetto sul link.
 - **Dipende dalla dimensione del pacchetto e dalla larghezza di banda del link.** Se la larghezza di banda è alta, il ritardo di trasmissione è basso.
 - Si può calcolare come $d_{trans} = L/R$, con L =packet length (bits), R =link transmission rate(bps)
- **Propagation delay (d_{prop})**
 - È il tempo che impiega il segnale a propagarsi attraverso il mezzo di trasmissione, cioè a raggiungere la destinazione dal router.
 - **Dipende dalla distanza tra i due nodi e la velocità del mezzo.** Se la distanza è lunga o la velocità è bassa, il ritardo di propagazione è alto.
 - Si può calcolare come $d_{prop} = d/v$, con d =length of physical link, v =propagation speed($\sim 2 \times 10^8$ m/s)
- **End-to-end delay (d_{nodal})**
 - È il tempo che impiega un pacchetto a viaggiare da una sorgente a una destinazione attraverso una rete di nodi intermedi, come i router.
 - Dipende da diversi fattori, come la **lunghezza del pacchetto, la capacità del link, la distanza tra i nodi e il numero di nodi attraversati.**

$$d = \frac{L}{R} \quad L = b \text{ bit} \\ R = b \text{ ps}$$

$$d = \frac{x}{v} \quad x = m \\ v = \frac{m}{s}$$

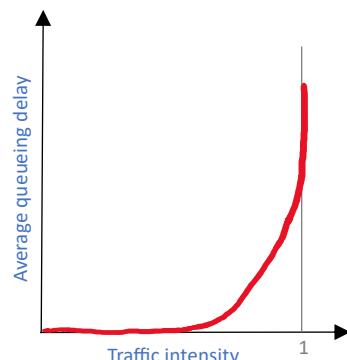
Possiamo riassumere tutto ciò quindi attraverso una formula matematica e un grafico:

$$\text{Traffic intensity} = \frac{L * a}{R} = \frac{\text{Arrival rate of bits}}{\text{Service rate of bits}}$$

a = average packet arrival rate; L = packet length (bits); R = link bandwidth (bit transmission rate)

$TI \sim 0$ $TI \rightarrow 1$ $TI > 1$
average queueing, small delay average queueing, large delay average delay infinite

$$d_{end} = N(d_{elab} + d_{trans} + d_{prop})$$



Packet loss

Il packet loss si verifica quando uno o più pacchetti di dati che viaggiano attraverso una rete informatica non raggiungono la loro destinazione. Può essere causato da errori nella trasmissione dei dati, tipicamente nelle reti wireless, o da **congestione della rete**. Si misura come una percentuale di pacchetti persi rispetto ai pacchetti inviati.

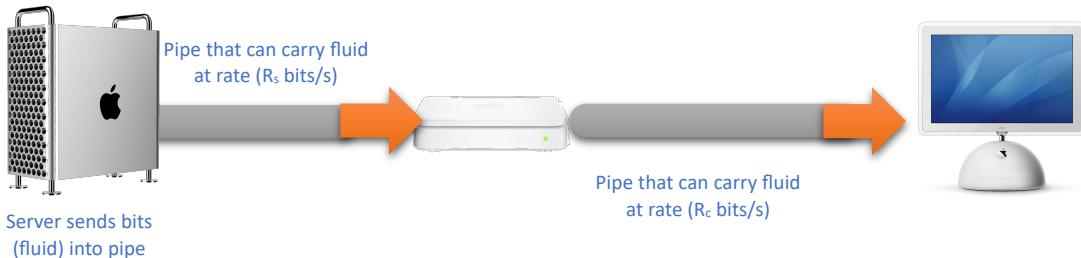
Quando un pacchetto arriva a un router, deve essere messo in coda prima di essere trasmesso al link successivo. La coda ha una capacità finita, quindi se è **piena**, il pacchetto **in arrivo viene scartato**, ovvero viene effettuato un **packet drop**. Può essere quindi ritrasmesso dal nodo precedente, dal sistema terminale sorgente o non essere trasmesso affatto.

Per mitigare questo problema, esistono diverse tecniche di recupero della perdita di pacchetti, come la *codifica di canale*, la *ritrasmissione*, l'*interleaving*, le *correzioni degli errori in avanti* e il *packet loss*.

Throughput

Il throughput è una misura della quantità di dati che possono essere trasmessi da un mittente a un destinatario in una rete informatica in un'unità di tempo. È spesso chiamato anche **end-to-end throughput**, perché considera il percorso completo che i dati devono seguire tra la sorgente e la destinazione. Il throughput dipende dalla **capacità dei link, ritardo di trasmissione e propagazione, la perdita di pacchetti, il controllo di flusso e il controllo della congestione**. Due tipi di throughput sono:

- **Throughput instantaneo:** è il tasso di trasmissione dei dati in un dato istante di tempo. Può variare molto a seconda delle condizioni della rete e delle applicazioni in esecuzione. Si può calcolare come il rapporto tra la quantità di dati trasmessi e l'intervallo di tempo considerato.
- **Throughput medio:** tasso medio di trasmissione dei dati un periodo di tempo più lungo. È più stabile, ed è rappresentativo del comportamento della rete e delle applicazioni. Si può calcolare come il rapporto tra la quantità totale di dati trasmessi e il tempo totale impiegato.



Il throughput end-to-end per una singola connessione è dato dal minimo tra la capacità del link di uscita dal mittente (R_s), la capacità del link di entrata al destinatario (R_c) e la capacità del link condiviso tra più connessioni; questo perché il throughput è limitato dal link più lento nel percorso, che è chiamato il link bottleneck.

$$\min(R_c, R_s, R/10)$$

R_c or R_s is bottleneck

Spesso il link bottleneck è quello di uscita dal mittente o quello di entrata dal destinatario, perché hanno una capacità inferiore a quella del link condiviso.

Circuit switching

Un circuito è un insieme di risorse che **vengono dedicate a una sola comunicazione**. Quando si stabilisce un circuito tra due nodi, nessun altro può usare quelle risorse fino a quando il circuito non viene chiuso (end-to-end), e viene ancora usata per garantire una qualità della comunicazione.

Il mittente e il destinatario mantengono la connessione per tutta la durata della comunicazione. Quando la rete stabilisce un circuito, gli garantisce una certa larghezza di banda.

Multiplexing nelle reti a commutazione di circuito

Il multiplexing è una tecnica che permette di **trasmettere più segnali su un unico canale**, ed esistono due tipi principali:

- **FDM** (Frequency Division Multiplexing) consiste nel dividere lo spettro di frequenza del canale in bande più strette, assegnate a ciascuna connessione.
- **TDM** (Time Division Multiplexing) consiste nel dividere il tempo in intervalli regolari, chiamati **frame**, e suddividere ogni frame in slot temporali, assegnati a ciascuna connessione. Ogni connessione può trasmettere un segnale per uno slot di tempo in ogni frame. Questo è più efficiente, perché non spreca banda durante i periodi di silenzio.

Una rete di reti

Internet è costituito a sua volta, in poche parole di varie reti:

- **Struttura di rete 1:** tutti gli ISP sono connessi in un unico ISP globale di transito immaginario
- **Struttura di rete 2:** centinaia di migliaia di ISP di accesso e più ISP globali di transito sono connessi tra loro. In ogni regione può esservi un **ISP regionale** al quale tutti gli ISP di accesso della regione si connettono. Ogni ISP regionale si connette al **tier-1 ISP**
- **Struttura di rete 3:** Aggiunge ISP provinciali, regionali, ecc.....
- **Struttura di rete 4:** aggiunge
 - **PoP (Point of Presence):** su tutti i livelli tranne quelli degli ISP di accesso. Sono gruppi di router tramite i quali gli ISP clienti si collegano al fornitore
 - **Multi-homing:** qualunque ISP, tranne quelli di primo livello, possono connettersi a due o più ISP fornitori
 - **Peering:** coppia di ISP vicini e di pari livello gerarchico connettono direttamente le loro reti in modo che tutto il traffico tra di esse non debba passare per un intermediario.
 - **IXP (Internet Exchange Point):** punto d'incontro in cui più ISP possono fare peering tra loro
- **Struttura di rete 5:** Aggiunge **content provider networks**, come per esempio la rete di Google.

Algoritmi e protocolli di routing

Gli algoritmi permettono di **determinare i percorsi o i cammini da una sorgente a una destinazione** attraverso una serie di router, mentre i protocolli permettono di **determinare come le informazioni devono essere condivise tra i router** per l'esecuzione degli algoritmi di routing.

Le etichette di ciascun arco viene detto **costo di collegamento**, e vengono definiti dagli operatori di rete. Se due router non sono direttamente collegati, per definizione il loro costo è ∞ .

Possiamo classificare gli algoritmi di routing in due categorie:

1. Quanto frequentemente i cammini sul grafo cambiano
 - a. **Statici:** i cammini cambiano poco frequentemente
 - b. **Dinamici:** i cammini cambiano più frequentemente
2. Riguardo la valenza di un algoritmo
 - a. **Globali:** tutti i router sono in grado di avere una conoscenza completa della topologia della rete, e si può ottenere condividendo con gli altri router lo stato dei propri link (*link state*)
 - b. **Decentralizzato:** tutti i router nella rete inviano delle informazioni di distanza a tutti i nodi adiacenti (*distance vector*)

Algoritmo distance vector

In fase di inizializzazione della rete, ogni nodo va a stimare il proprio vettore di distanza; successivamente, ogni nodo **invia ai propri vicini il proprio vettore di distanza**.

Quando un nodo riceve un vettore di distanza da un qualsiasi nodo vicino, questo effettua l'aggiornamento del suo vettore di distanza usando l'equazione di Bellman-Ford. Se con l'aggiornamento ha una modifica del vettore di distanza originario, il vettore di distanza nuovo va inoltrato a tutti i vicini. I vicini poi ricevono il nuovo vettore, aggiornano il proprio, e continua finché l'algoritmo non raggiunge uno stato di convergenza in cui ogni nodo ha il proprio vettore di distanza aggiornato che gli indica qual è la distanza per raggiungere ogni nodo nella rete.

Questo algoritmo è di tipo **iterativo, asincrono, distribuito e self-stopping**:

- Iterativo: questo processo si ripete fino a quando non interviene un ulteriore scambio informativo tra vicini
- Asincrono: non richiede che tutti i nodi operino al passo con gli altri
- Distribuito: ciascun nodo riceve informazioni dai nodi adiacenti
- Self-stopping: il calcolo semplicemente si blocca.

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

Ciascun nodo x inizia con $D_x(y)$, una stima del percorso a costo minimo da se stesso al nodo y , per tutti i nodi in N .

Sia $D_x = [D_x(y): y \in N]$ il vettore delle distanze del nodo x , che è il vettore delle stime dei costi da x a tutti gli altri nodi, y , in N . Ciascun

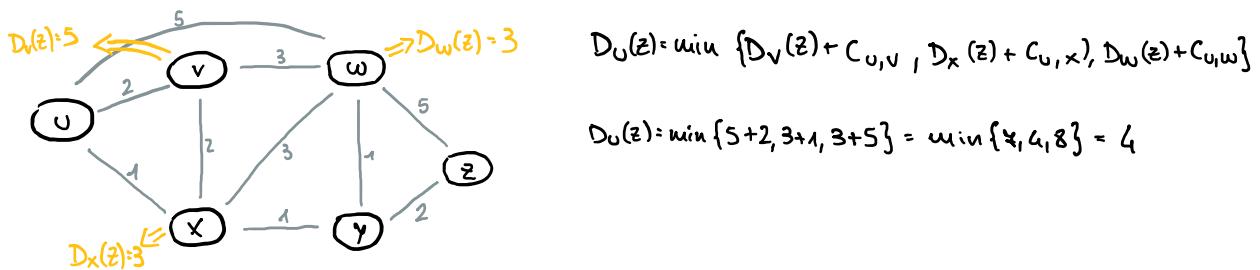
nodo x mantiene:

- Per ciascun vicino v , il costo $c(x,v)$ da x al vicino v
- Il vettore delle distanze del nodo x , che è $D_x = [D_x(y)]$, contenente la stima presso x del costo verso tutte le destinazioni, y , in N
- I vettori delle distanze di ciascuno dei suoi vicini, $D_v = [D_v(y)]$, per ciascun vicino y di x

Equazione di Bellman-Ford

Gli algoritmi distance vector si basano sull'implementazione distribuita dell'algoritmo di **Bellman-Ford**:
 $\text{dist}(A,X) = \min_v \{\text{dist}(V,X) + c(A,V)\}$

Vediamo un esempio per calcolare la distanza da un nodo U e Z per esempio:



Il vettore di distanza è il vettore di tutte le distanze di un nodo verso tutti i nodi della rete.

Link state

Nell'algoritmo link state, si ha una distribuzione in selective flooding verso tutti i nodi della rete, e l'informazione che viene mandata è lo stato dei propri link. Dato che tutti i nodi sono a conoscenza della topologia della rete, questi possono calcolare la distanza tra loro e ogni altro nodo nella rete attraverso l'algoritmo di **Dijkstra**.

Algoritmo di Dijkstra

Questo algoritmo calcola l'albero dei cammini di costo minimo. È di tipo centralizzato, in cui ogni nodo invia messaggi in broadcast verso tutti gli altri nodi della rete sullo stato dei propri link, e, grazie alla computazione dell'albero dei cammini minimi tra una sorgente e tutte le altre destinazioni, è facile andare a popolare la tabella di inoltro. È di tipo **iterativo**, e con un numero di iterazioni pari a $n-1$ (n = numero di nodi), conosco la distanza verso tutte le destinazioni.

Abbiamo le seguenti notazioni:

- $D(v)$: stima del costo del cammino a costo minimo dalla sorgente alla destinazione v
- $p(v)$: nodo predecessore tra il nodo sorgente e il nodo che sto considerando
- N' : insieme dei nodi la cui distanza è conosciuta in modo definitivo; quindi, l'obiettivo dell'algoritmo è di aggiungere a questo insieme tutti i nodi della rete, quando questo accade, l'algoritmo si ferma.

Abbiamo due fasi:

1. Fase di inizializzazione:

- a. Si inserisce in N' il nodo U (radice dell'albero)
- b. Per tutti i nodi adiacenti a u chiamati v , si definisce la distanza come il costo tra u e v ; per tutti gli altri nodi, si imposta a ∞ .

2. **Fase di iterazione:**

- a. Trovare un nodo w tale che non appartiene a N' , in modo tale che la distanza dalla sorgente a w sia il valore minimo
- b. Si aggiunge w a N'
- c. Si aggiorna la distanza di tutti i nodi v adiacenti a w che non appartengono a N' . Il valore della distanza verso il nodo v è uguale al minimo tra la distanza di w e la distanza verso il nodo w + il costo tra w e v .

$$D(v) = \min(D(v) \text{ e } D(w) + C_{w,v})$$

Le differenze

Distance vector	Link state
Implementazione semplice e intuitiva, Bellman-Ford	Implementazione complessa; i nodi devono ricostruire la topologia della rete attraverso le informazioni di stato dei link degli altri nodi
I vettori di distanza vengono inviati solo ai vicini (numero messaggi ridotto)	Il numero di messaggi in rete è molto elevato perché i messaggi sono inviati in selective flooding verso tutti i nodi della rete
Convergenza lenta	Convergenza rapida
Problemi nel raggiungere convergenza	Convergenza riesce sempre a raggiungere
RIP, IGRP, EIGRP, BGP	OSPF, IS-IS

Nella pratica, all'interno delle regioni amministrative della rete, l'entità amministrativa che gestisce la rete vuole avere il controllo sull'instradamento, e sapere come i pacchetti circolano nella rete. Se aggreghiamo varie regioni amministrative, queste prendono il nome di **domini**:

Domini

inserisci spiegazione di chatgpt

Abbiamo due tipi di domini che specificano determinate cose:

1. **Intra-AS/intra-domain:** come instradare i pacchetti all'interno di un sistema autonomo.
 - a. Si definisce in che modo va effettuato l'instradamento se voglio far comunicare due host che sono collegati a due diverse interfacce verso lo stesso sistema autonomo.
 - b. Ogni singolo sistema autonomo si gestisce il routing come vuole, e può utilizzare un protocollo diverso rispetto a quello utilizzato da un altro sistema autonomo. Tutti i router dello stesso sistema autonomo devono utilizzare lo stesso protocollo di routing.
 - c. I routing all'interno dei sistemi autonomi viene eseguito dai protocolli **IGP** (Interior-gateway), che sono **RIP** (Routing Information Protocol) e **EIGRP/OSPF**.
2. **Inter-AS/inter-domain:** come instradare i pacchetti tra sistemi autonomi diversi. Qui sono importanti i gateway router, che si trovano ai bordi del sistema autonomo e permettono di interconnettersi, ovvero di *inoltrare il traffico verso altri sistemi autonomi*.

OSPF (Open Shortest Path First)

È un protocollo di tipo **link-state**, che quindi utilizza l'algoritmo di Dijkstra. Si inviano messaggi di tipo *link-state advertisements* direttamente encapsulati nell'header IP.

È un protocollo sicuro perché tutti i messaggi OSPF sono autenticati; si ha un livello a doppia gerarchia all'interno dello stesso sistema autonomo: aree/aree locali e **backbone**, che mette in comunicazione diverse aree locali. I messaggi di tipo link-state vengono inviati in broadcast in una dei due tipi di aree.

Grazie a questi, ogni nodo all'interno dell'area conosce la topologia dettagliata dell'area, quali sono i router di bordo, ecc....

I router della backbone si scambiano messaggi link-state advertisements per ricreare la topologia della rete di backbone.

Di seguito, un pseudo codice per mostrare come funzionerebbe, a grandi linee, un algoritmo di questo genere:

Inizializzazione:

```
N' = {u}
Per tutti i nodi v
Se v è adiacente a u
    Allora D(v) = c(u,v)
Altrimenti D(v)=∞
```

Ciclo:
 Determina un w non in N' tale che D(w) sia minimo
 Aggiungi w a N'
 Aggiorna D(v) per ciascun nodo v adiacente a w, e non in N':
 $D(v) = \min((D(v), D(w) + c(w,v)))$
 Ripeti in ciclo finché non si verifica che $N' = N$

Quando termina, abbiamo per ciascun nodo il suo predecessore lungo il percorso a costo minimo dal nodo origine. Per ciascun predecessore abbiamo il rispettivo predecessore, e in questo modo riusciamo a costruire l'intero percorso dall'origine a tutte le destinazioni. Ha un costo di $O(n^2)$.

Cheatsheet primo parziale

Numero di pacchetti

Calcolare il numero di pacchetti data la **quantità totale di dati** e la **quantità di dati in ogni pacchetto**.

$$N = \frac{L}{l} \quad \begin{array}{l} L = \text{quantità totale} \\ l = \text{quantità in singolo pacchetto} \end{array} \quad [-] = \frac{\text{bit}}{\text{bit/s}}$$

Ritardo di trasmissione

Calcolare il tempo che una certa quantità di dati impiega per essere trasmesso (solo immesso in rete).

$$d_{trans} = \frac{L}{R} \quad \begin{array}{l} L = \text{quantità totale} \\ R = \text{velocità trasferimento} \end{array} \quad [s] = \frac{\text{bit}}{\text{bps}}$$

Ritardo di propagazione

Calcolare il tempo che una certa quantità di dati impiega per propagarsi in un dato mezzo.

$$d_{prop} = \frac{d}{v} \quad \begin{array}{l} d = \text{distanza} \\ v = \text{velocità nel mezzo} \end{array} \quad [s] = \frac{\text{m}}{\text{m/s}}$$

Store and forward senza frammentazione

Calcolare il tempo che una certa quantità di dati impiega a passare attraverso Q link (Q-1 router) con la stessa velocità di trasmissione.

$$d_h = Q \frac{L}{R} \quad \begin{array}{l} Q = \text{n. link da attraversare} \\ L = \text{quantità di dati} \\ R = \text{velocità trasmissione} \end{array} \quad [s] = \frac{\text{bit}}{\text{bps}}$$

Store and forward con frammentazione

Calcolare il tempo che una certa quantità di dati divisa in pacchetti impiega a passare attraverso un numero di link con la stessa velocità di trasmissione.

$$d_h = \frac{L}{R} + (Q-1) \frac{L}{R}$$

*L: quantità di dati
R: velocità
Q: numero link
L: quantità dati
sopra pacchetto*

$[s] = \frac{\text{bit}}{\text{bps}} + [-] \cdot \frac{L}{R}$

Attraversamento di un nodo

Calcolare il tempo totale che il pacchetto impiega ad attraversare un nodo.

$$d_n = d_{proc} + d_{queue} + d_{trans} + d_{proc}$$

*d_{proc}: processing
d_{queue}: attesa in coda
d_{trans}: transazione*

Utilizzazione del canale

Calcolare l' utilizzazione del canale TCP e una finestra da N_w pacchetti. Si può convertire in percentuale: 100%=1, utilizzo canale è continuo e si ha efficienza massima.

$$U = \frac{N_w \cdot L / R}{RTT + L / R}$$

*N_w: n. pacchetti in finestra
L: quantità dati
R: velocità di trasmissione
RTT: round trip time*

[-] = $\frac{[-] \cdot \frac{\text{bit}}{\text{bps}}}{S + \frac{\text{bit}}{\text{bps}}}$

Intensità del traffico

Calcolare il carico della rete; rapporto fra la velocità dei dati in entrata e in uscita.

$$i = \frac{a \cdot L}{R}$$

*a: n. pacchetti in arrivo al secondo
L: quantità di dati da trasmettere
R: velocità di trasmissione*

[-]: $S^{-1} \cdot \frac{\text{bit}}{\text{bps}}$

Percentuale: 100%=1, rete rischia di andare in sovraccarico: se supera arrivano più dati di quelli che può trasmettere.

N. Pacchetti ideale nella finestra

Calcolare la dimensione ideale della finestra (quanti pacchetti deve contenere come minimo affinchè il canale sia utilizzato continuamente).

$$N_w \geq \frac{RTT \cdot R}{L} + 1$$

*L: quantità di dati
R: velocità di trasmissione
RTT: round trip time*

[-] = $\frac{S \cdot \text{bps}}{\text{bit}}$

Latenza in TCP

Calcolare il tempo passato dal momento di inizio trasmissione alla ricezione di ACK.

$$d_{tot} = 2RTT + \frac{L}{R}$$

*RTT: round trip time
L: quantità di dati
R: velocità di trasmissione*

[s] = [s] + $\frac{\text{bit}}{\text{bps}}$

Throughput

Calcolare il throughput medio, ossia la quantità media di dati trasmessi al secondo.

$$T = \frac{3N_w \cdot L}{4RTT}$$

*N_w: n. pacchetti in finestra
L: quantità di dati
RTT: round trip time*

[bps] = $\frac{[-] \cdot \text{bit}}{s}$

