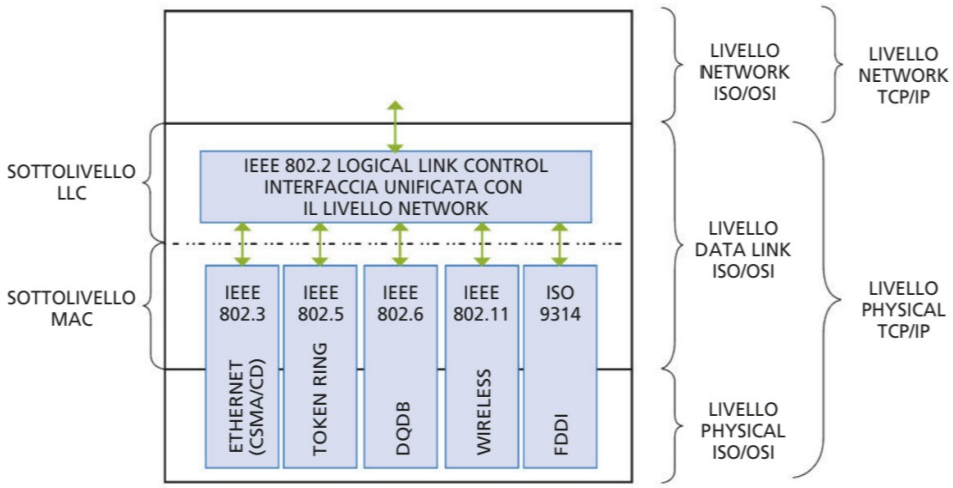
1 Livello datalink

IEEE, ISO e ANSI hanno sviluppato uno standard comunemente noto come **Progetto IEEE 802**, in cui venne stabilita la realizzazione delle reti LAN ai livelli Datalink e Physical. In questo progetto vennero stabilite **20 categorie** che **identificano le reti**, e, tra questi, le più importanti sono:

1. **802.2** – Specifiche del **LLC** (**L**ogical **L**ink **C**ontrol)
2. **802.3** – Specifiche **CSMA/CD** (chiamato comunemente *Ethernet*)
3. **802.5** – Le specifiche Token Ring
4. **802.6** – Specifiche **DQDB** (**D**istributed-**Q**ueue **D**ual-**B**us)
5. **802.11** – Specifiche reti Wireless (Wi-Fi)

Per comprendere la struttura del **livello Physical dell'architettura TCP/IP** occorre fase un passo indietro e dettagliare alcune caratteristiche dei *livelli 1 e 2*[1]. In particolare guarderemo il livello 2, che è stato suddiviso in due sottolivelli:

- **LLC** (**L**ogical **L**ink **C**ontrol)
- **MAC** (**M**edia **A**ccess **C**ontrol)



Il livello Physical deve preoccuparsi dell'**accesso alla rete di comunicazione** tenendo presente che le trasmissioni broadcast condividono un unico canale e che quindi è necessario verificare che il canale sia effettivamente libero prima di iniziare una trasmissione, risolvendo eventuali conflitti. Risulta allora evidente stabilire un

1 **Livelli 1 e 2**: Nell'architettura TCP/IP il livello fisico e datalink sono "compressi" in un unico livello, chiamato livello fisico.

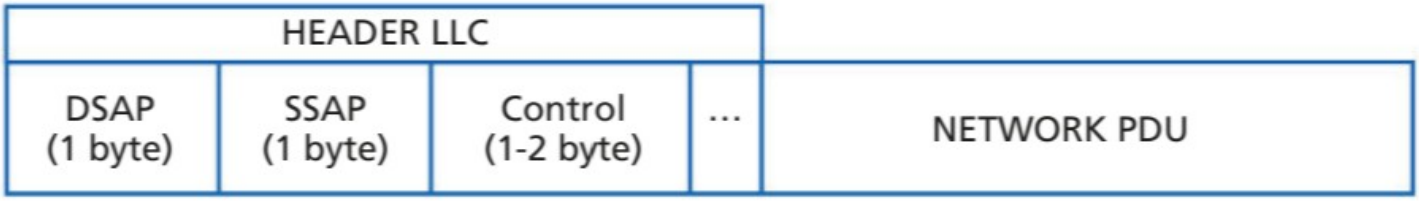
metodo di accesso, ovvero un **algoritmo che regoli il diritto a trasmettere** sul canale condiviso. Due sono i metodi utilizzabili:

- **tecnica a contesa**, prevede l'accesso casuale al canale e sue due o più stazioni cercano di trasmettere simultaneamente, il conflitto viene risolto secondo alcune regole di mediazione. La tecnica più nota è **CSMA/CD** (***C**arrier **S**ense **M**ultiple **A**ccess with **C**ollision **D**etection*).
- **tecnica deterministica**, dove ogni trasmissione avviene in un istante definito e sicuramente va a buon fine. Le tecniche più note sono quelle basate su **token** (ex. Token Ring) che autorizza a trasmettere.

1.1 I sottolivelli LLC e MAC

Il sottolivello **LLC** ha il fondamentale compito di **fornire un'interfaccia unificata verso il livello Network**, pur a fronte di tecnologie trasmissive e mezzi fisici differenziati. Può inoltre occuparsi del controllo di flusso di trasferimento dei dati.

Poiché a livello Network possono operare vari protocolli (anche se il principale è IP), LLC deve individuare qual è il protocollo usato. Proprio per questo scopo il frame LLC contiene due indirizzi, da un byte ciascuno, detti **DSAP** (***D**estination **S**ervice **A**ccess **P**oint*) e **SSAP** (***S**ource **S**ervice **A**ccess **P**oint*), che rappresentano, rispettivamente, l'**identificatore del protocollo di livello superiore** (quindi network) ***a*** **cui deve essere consegnato il pacchetto ricevuto**, e l'**identificatore del livello del protocollo di livello superiore**, ***da*** **cui è arrivato il pacchetto**.



Il sottolivello LLC prevede tre modi di funzionamento:

- **Unacknowledged Connectionless Service**: costituito solo da primitive di trasferimento dati. È un *servizio non affidabile* e *non connesso*[2]. Qualora tali funzioni siano necessarie, devono essere fornite dai protocolli di livello superiore.

2 **Servizio non affidabile e non connesso**: *non affidabile* significa che non viene mandato l'**ACK** di conferma ricezione, mentre *non connesso* significa che i pacchetti vengono inviati in modo casuale e non è detto che arrivino a destinazione.

- **Connection Oriented Service**: costituito da primitive di trasferimento e di apertura/chiusura di una connessione con le funzioni per il controllo di errore, *di flusso* e di conservazione della sequenza[3]. È un *servizio affidabile* e *connesso*.
- **Semireliable Connectionless Service**: costituito solo da primitive di trasferimento dati ma prevede una conferma di ricezione per i datagram inviati. È quindi un *servizio affidabile* ma *non connesso*.

Il sottolivello inferiore è il **MAC** che risolve il problema dell'accesso al mezzo trasmissivo condiviso. Cioè il suo compito è quello di **arbitrare l'accesso all'unico mezzo trasmissivo comune** tra tutti i sistemi che hanno necessità di trasmettere ("Sezione critica" cit. Ingegnere). Quindi mentre l'LLC è unico, si avrà invece uno **standard MAC diverso per ogni tipo di rete e mezzo fisico di trasmissione**. Anche il frame MAC contiene due indirizzi di tipo DSAP e SSAP, detti proprio indirizzi MAC del sorgente e del destinatario, che hanno lo scopo di identificare l'indirizzo fisico delle due entità che si stanno scambiando il pacchetto destinato al/in arrivo dal sottolivello LLC.

L'**FCS** (***F**rame **C**heck **S**equenze*) è di solito il **CRC** (***C**yclic **R**edundancy **C**heck*).

DSAP e SSAP sono **indirizzi MAC**, e sono costituiti da 6 byte codificati in esadecimale, ad esempio:

08-00-2B-C4-BE-F3

In cui i **primi 3 ottetti** sono chiamati **OUI** (***O**rganization **U**nique **I**dentifier*) che, appunto, identificano l'azienda produttrice della scheda, mentre gli **ultimi 3 ottetti** rappresentano il **numero seriale**.

La caratteristica fondamentale dell'indirizzo MAC è quello di essere **univoco**: non esistono due indirizzi MAC uguali in tutto il pianeta.

3 **Conservazione della sequenza**: quando un servizio è connesso, i pacchetti vengono inviati seguendo un percorso prestabilito e in modo ordinato.

Gli indirizzi MAC possono essere di 3 tipi:

- **unicast**: individua una stazione singola
- **multicast**: individua un gruppo di stazioni. In questi casi si trasmette per primo il bit meno significativo del primo byte messo a 1. La rappresentazione formale è del tipo:

  FF-FF-FF-[0:F]X-XX-XX

  usata per indicare tutti gli host aventi scheda di rete di qualsiasi produttore con la prima cifra del terzo byte compresa tra 0 e F
- **broadcast**: **FF-FF-FF-FF-FF-FF** individua tutti gli host connessi alla rete

1.1.1 Vulnerabilità

Per proteggere una LAN è utile attivare la **protezione delle porte** negli switch. I frame il cui indirizzo MAC non è specificato per una determinata porta dello switch, non potranno accedere allo switch attraverso quella porta e di conseguenza non potranno accedere alle reti alle quali lo switch è connesso.

Ogni porta dello switch, di regola, può fornire protezione a 1024 indirizzi MAC. Il numero max. di indirizzi MAC per ciascuna porta dipende dalla configurazione della LAN.

1.2 La rete ethernet e le sue evoluzioni

Ethernet è il più diffuso tipo di rete locale che esiste al mondo. Ethernet usa un solo cavo per collegare contemporaneamente tutto quel che passa sulla rete. Ogni stazione è indipendente, e una sola stazione a volta può trasmettere. Ogni messaggio nella rete reca al proprio interno l'indirizzo MAC di origine/destinazione.

La prima versione di rete Ethernet usava un cavo coassiale che garantiva una velocità di **10 Mbps**:

- **10Base-5** – **802.3**: prima versione
- **10Base-2** – **802.3a**: seconda versione

Le reti a cavo coassiale sono fuori standard dal 2003.

L'avvento del doppino telefonico e degli hub porta alla nascita di un nuovo standard:

- **10Base-T** – **802.3i**: velocità ancora di 10 Mbps

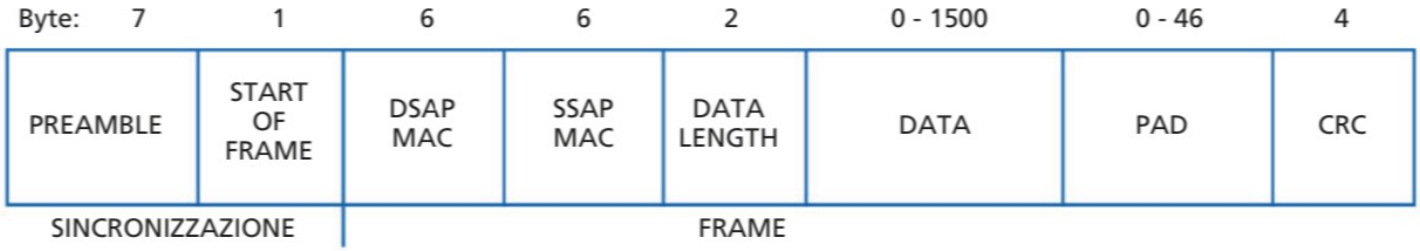
Cinque anni dopo si evolve:

- **100Base-TX** – **802.3u**: la velocità passa a 100 Mbps

L'ultima evoluzione ha portato dalla Fast Ethernet alla **Gigabit Ethernet**:

- **1000Base-T** – **1000Base-TX** – **802.3ab**: velocità fino a 1 Gbps

Il formato del **frame Ethernet** è costituito da 6 campi che sono preceduti da un **preambolo** e da un **byte di start**.



I singoli campi sono:

- **Preamble** – **preambolo**: costituito da 7 byte tutti uguali con valori 10101010 allo scopo di permettere al destinatario di sincronizzarsi
- **Start of frame**: 1 byte uguale a 10101011 che con gli ultimi due bit a 11 effettua una violazione della sequenza fissa 10, segnalando così la fine del sincronismo e l'inizio del frame
- **DSAP MAC**
- **SSAP MAC**
- **Data length**: lunghezza in byte del campo *Data*
- **Data**: contiene i dati da trasmettere, può anche essere vuoto (*frame di controllo*)

- **Pad**: riempitivo che garantisce che la lunghezza minima del frame sia di 64 byte al fine di rendere possibile distinguere un frame da un frammento di frame a seguito di una collisione
- **CRC**

1.2.1 Lo switching

L'uso dello switch è fondamentale per segmentare le reti e rendere compatibili le velocità di 10/100/1000 Mbps. Lo switch esegue tutte le proprie elaborazioni via hardware e non via software, perciò non rallenta il traffico.

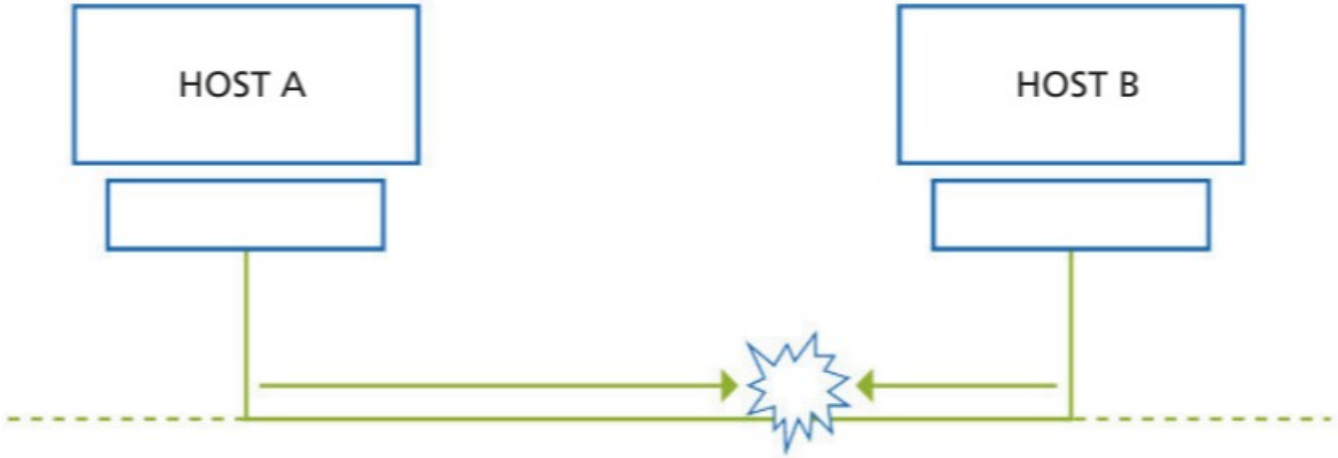
La prima tecnica di switching si chiama **store-and-forward**, dove ogni frame che arriva su una delle porte viene salvato in un buffer e quindi inoltrato o scartato a seconda che l'indirizzo di destinazione sia corretto oppure no. L'operazione è velocissima, ma comporta in ogni caso un certo rallentamento perché il frame deve arrivare per intero nel buffer dello switch prima di cominciare a essere ritrasmesso su un'altra porta a cui corrisponde un altro segmento di rete. Tuttavia su impianti molto veloci il numero di frame in circolazione è molto elevato e il ritardo che si accumula si fa sentire.

L'alternativa ideata per eliminare questo inconveniente è lo switching **cut-through**. La parola significa *prendere una scorciatoia*, e in effetti è proprio quello che accade. Non appena lo switch comincia a ricevere un frame su una qualsiasi delle sue porte, ne legge l'indirizzo di destinazione e se questo corrisponde a un segmento collegato ad un'altra porta, inizia immediatamente a trasmettere il frame senza aspettare che questo sia arrivato per intero.

Dal confronto di questi due approcci ne è stato ideato un terzo, **fragment-free**, il quale bufferizza solo i primi 64 byte allo scopo di verificare eventuali errori che statisticamente cadono più frequentemente nei primi byte del frame.

1.2.2 CSMA/CD

La caratteristica fondamentale delle reti Ethernet è di trasmettere su un bus unico e quindi condiviso tra molti host, utilizzando la *tecnica a contesa*.



L'Host A si mette in ascolto sul canale e, ritenendolo libero, trasmette. Stessa cosa fa l'Host B. In seguito all'avvenuto rilevamento della collisione si mette in moto il seguente procedimento:

1. L'host che si accorge per primo della collisione interrompe la trasmissione
2. lo stesso host immette sulla rete un pacchetto speciale, noto come **sequenza di jamming**, di 48 bit
3. gli host in ascolto, riconoscendo il pacchetto speciale, interrompono anch'essi le trasmissioni e scartano i frammenti ricevuti
4. prima di ricominciare a trasmettere, ogni host attende un tempo pseudocasuale dato dall'**algoritmo di backoff esponenziale binario**

Il tempo di attesa minimo (*Slot time*) è pari alla lunghezza minima del frame Ethernet diviso la velocità del canale. La lunghezza minima del frame è 64 byte.

Se supponiamo il canale a velocità 10 Mbps, abbiamo che il tempo minimo necessario per inviare i 64 byte è:

Slot Time = 64 byte / 10 Mbps = 51.2 µs

L'algoritmo di backoff stabilisce che il tempo di attesa effettivo sia un multiplo r del timeslot, calcolato nel seguente modo:

- tempo di attesa effettivo → r * slot time

- con $0 < r < 2^k - 1$
- con $k = \min(n, 10)$ dove n è il numero di collisioni consecutive

L'algoritmo tende a **premiare gli host col minor numero di collisioni consecutive** n, poiché esse minimizzano k che a sua volta riduce il range in cui viene estratto random il fattore r.

1.3 La rete wireless (IEEE 802.11)

Lo standard IEEE 802.11 nacque nel 1997 ma, per via delle insufficienti prestazioni, rimase solo sulla carta.

Nel 1999 IEEE 802.11 emise due nuovi standard:

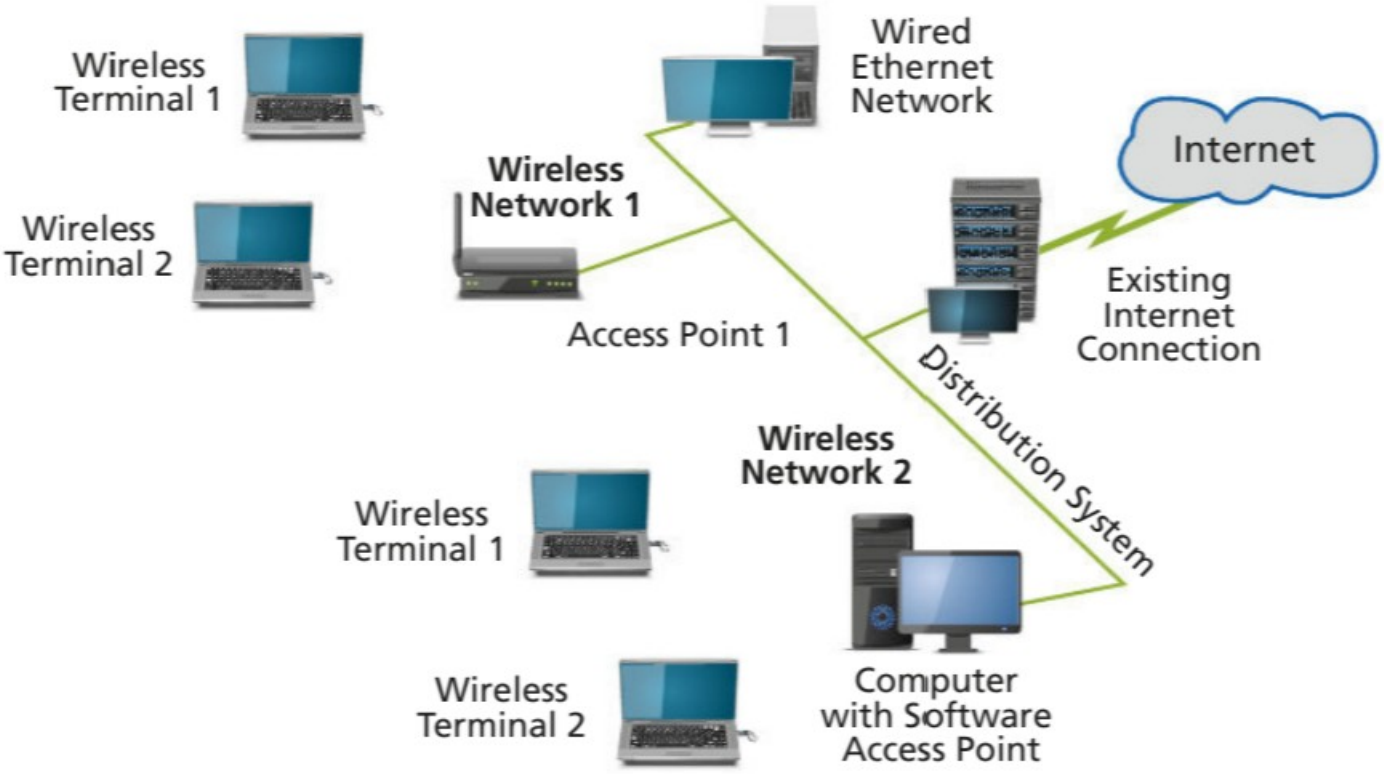
- **802.11a**, introduce la velocità di 5 GHz (54 Mbps)
- **802.11b**, introduce la velocità di 2.4 GHz (5,5 e 11 Mbps, noto anche come Wi-Fi)

Lo standard 802.11b ebbe più successo perché molti governi (tra cui quello italiano) hanno mantenuto libere alcune bande di frequenze tra cui quella a 2.4 GHz (nota come banda **ISM**, Industrial, Scientific and Medical). Tale frequenza può essere usata liberamente da chiunque, senza dover richiedere licenze.

La differenza tra lo standard 802.11b e 802.11a è, oltre alla **velocità**, il fatto che 802.11b può raggiungere **distanze 4 volte superiori** alla 802.11a.

I dispositivi che costituiscono le reti wireless sono due:

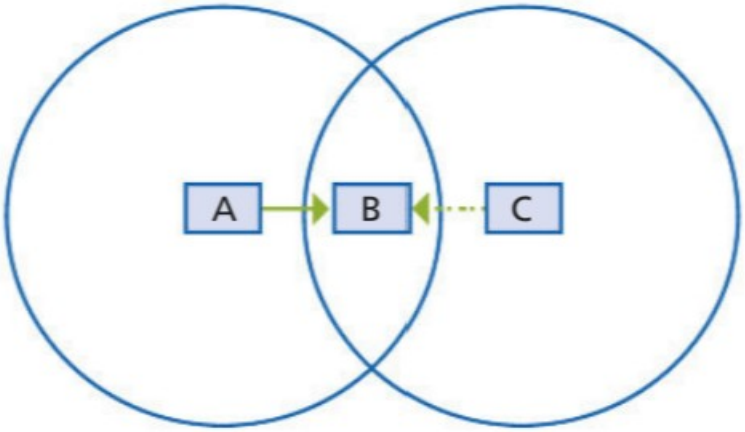
- **Wireless Terminal** (**WT**): dispositivi mobili, come notebook o smartphone
- **Access Point** (**AP**): hanno un doppio scopo, da un lato sono dei ponti che collegano la parte cablata con la parte wireless, dall'altro consentono ai WT di collegarsi alla rete wireless (è possibile anche usare dei computer dotati di apposito software per fungere da AP).



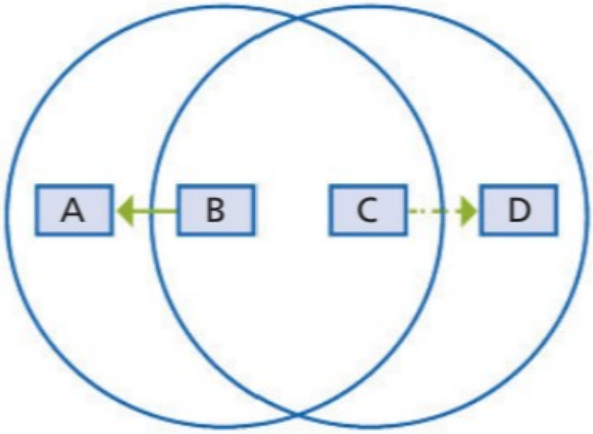
È possibile inoltre collegare più AP alla rete cablata o collegare tra loro più AP (**roaming**) creando una Wireless Distribution System.

Nelle trasmissioni wireless **non è possibile rilevare le collisioni**. Vanno dunque evitate a priori. Il modo più semplice è quello di costringere la stazione trasmittente ad ascoltare il canale e verificare che sia libero prima di iniziare la trasmissione. In alcuni casi, però, questo semplice accorgimento non basta. Due scenari, quella della **stazione nascosta** e quello della **stazione esposta**, lo dimostrano.

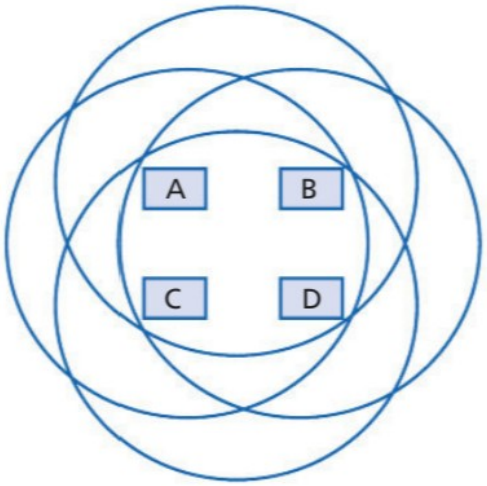
- **Problema della stazione nascosta**: ci sono 3 stazioni, A B e C. Se C ascolta il canale, lo troverà libero e sarà convinto di poter trasmettere a B; cominciando a trasmettere disturberà la trasmissione di A, impedendo a B di riceverla; sia A che C saranno costrette a ritrasmettere.



- **Problema della stazione esposta**: supponiamo di aver quattro stazioni, A B C e D. B sta trasmettendo ad A mentre C vuole trasmettere a D. Ascoltando il canale, C sentirà la trasmissione di B e concluderà erroneamente di non poter trasmettere; invece, essendo D fuori della portata di B, e A fuori dalla portata di C, le due trasmissioni potrebbero avvenire parallelamente senza interferenze



Il secondo problema della *stazione esposta* può essere risolto solo da un'accurata progettazione fisica della rete, sistemando le stazioni tutte nei rispettivi raggi d'azione.



Il primo problema della *stazione nascosta* è invece risolvibile mediante tecniche di **Carrier Sensing Virtuale**. Questa tecnica di ascolto consiste innanzitutto nell'invio, da parte del mittente, di un frame **RTS** (*Request To Send*) al destinatario contenente l'informazione sulla durata della trasmissione che intende effettuare. Il destinatario risponde con un frame **CTS** (*Clear To Send*) in cui ricopia il valore relativo alla durata della trasmissione del mittente. Alla ricezione del CTS, il mittente può cominciare a trasmettere. Però ha un problema, quello di non garantire la **mutua esclusione**, quindi qualora vi siano delle acquisizioni contemporanee del canale, ci saranno interferenze.

La soluzione più efficace è stata messa a punto con una tecnica di CSMA, che riduce le collisioni: **CSMA/CA** (***C**arrier **S**ense **M**ultiple **A**ccess with **C**ollision **A**voidance*). Questa tecnica introduce un intervallo di tempo chiamato AIFS (Arbitration Inter-Frame Space) durante il quale il trasmettitore attende al fine di accettarsi che non vi siano altri frame RTS e CTS sul canale, così invia. Se però un'altra stazione contestualmente tenta di trasmettere, provocando una collisione, allora viene

avviato un **algoritmo di backoff esponenziale binario**, simile al CSMA/CD, cioè si attende un tempo random prima di ritentare l'invio di un frame RTS.

2 Controllo di flusso

Nella trasmissione dati tra un mittente e un destinatario è necessario regolare il flusso dei dati in modo da evitare che i dati siano inviati ad una velocità superiore alla capacità di ricezione del destinatario. In generale i meccanismi del controllo di flusso prevedono che il destinatario invii un riscontro della corretta ricezione del messaggio, detto **acknowledge** (**ACK**), che può essere trasferito:

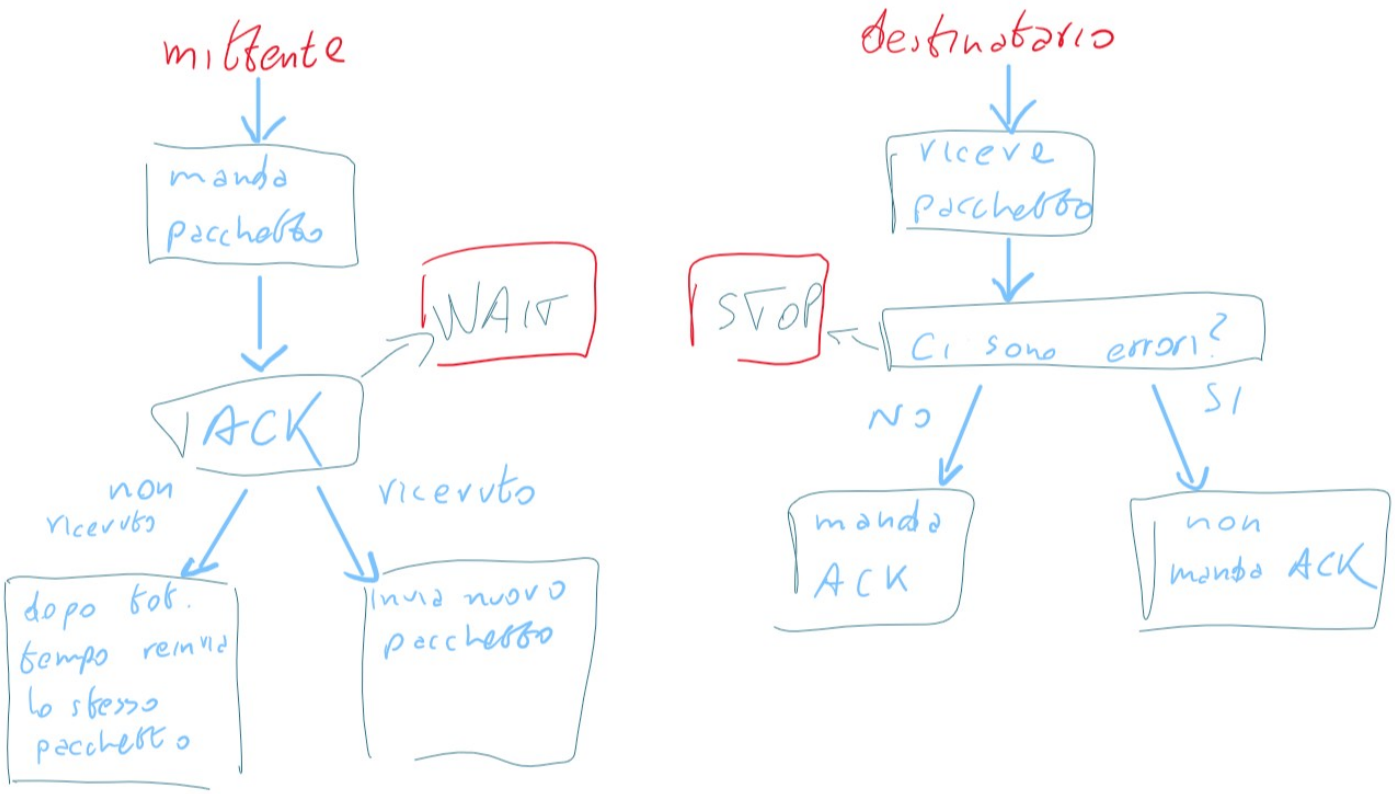
- con messaggi appositi contenenti solo informazioni di controllo
- con messaggi che contengono dati utente e sono trasferiti nel verso contrario a quello dell'informazione da riscontrare; in tal caso il riscontro è contenuto nella parte di controllo del messaggio (questa tecnica viene chiamata *piggybacking*).

Con tale tecnica si ritarda l'invio dell'ACK nell'attesa di eventuali messaggi da inviare nel verso opposto, risparmiando in questo modo sull'occupazione del canale e l'utilizzo della banda.

2.1 Il meccanismo Stop and Wait

Questo semplice meccanismo prevede che il mittente **attenda** il riscontro della

corretta ricezione del messaggio inviato prima di trasmettere il successivo. Il messaggio di riscontro (ACK) viene inviato dal destinatario solo se il messaggio è stato ricevuto senza errori, in caso contrario quest'ultimo verrà scartato e l'ACK non verrà inviato. Il mittente imposta un timer all'invio di ogni messaggio: se al suo scadere non avrà ricevuto l'ACK, ritrasmetterà il messaggio.



Lo svantaggio di questo meccanismo è che si potrebbero generare pacchetti duplicati. Per rilevarli, è necessario inserire un **numero di sequenza** nel messaggio, cosicché quando il ricevente controlla questo numero, e lo trova uguale all'ultimo ricevuto, scarta il messaggio. Il numero di sequenza può essere un bit che assume valori 0 e 1 in alternanza, infatti l'unica ambiguità è tra un messaggio e quello immediatamente precedente o successivo.

Si presentano 3 possibili scenari:

- tutto ok
- i dati trasmessi dal primo host non sono arrivati (o arrivati ma errati, quindi scartati): allo scadere del timeout il mittente ritrasmette la sequenza di dati
- nel caso in cui il messaggio ACK non arrivi (o risulti errato), allo scadere del timeout il mittente ritrasmette il pacchetto e, il destinatario, controllando il numero di sequenza riconosce che si tratta di un messaggio duplicato lo scarta ma invia nuovamente l'ACK

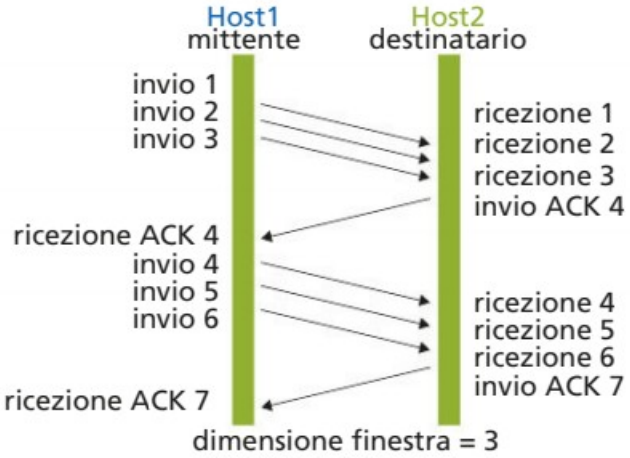
In questo meccanismo un elemento critico è il *timeout* del mittente, infatti se troppo breve potrebbero essere ritrasmessi messaggi che invece erano stati ricevuti

correttamente, mentre se è troppo lungo risulterebbero aumentati i tempi di trasmissione con conseguente basso utilizzo della banda disponibile.

2.2 La tecnica a finestra

Queste tecnica prevede di non inviare il riscontro per ogni messaggio ricevuto, permettendo che venga trasmesso un certo numero di messaggi (fino a un massimo prefissato che rappresenta la **dimensione della finestra** [n]) prima che venga inviato un ACK.

Ciascun messaggio contiene un numero di sequenza che può assumere valori compresi tra 0 e 2^n-1.



Supponendo che la dimensione della finestra sia n, si possono attuare due diversi meccanismi per gestire la ricezione di un messaggio errato in un punto intermedio della sequenza degli n messaggi:

- **Go-Back-N**: il mittente invia fino a n messaggi facendo di ognuno una copia, attiva un timer per ogni messaggio e si pone in attesa dei riscontri (ACK). Se si verifica un timeout prima dell'arrivo dell'ACK ripete la trasmissione di tutti i messaggi non ancora confermati. In questo protocollo Go-Back-N il ricevitore può accettare solo messaggi in sequenza, se arriva un messaggio corretto ma fuori sequenza viene scartato.
- **Selective repeat**: si chiede la ritrasmissione solo dei messaggi arrivati errati o non arrivati, il destinatario fornisce il riscontro di un messaggio ricevuto correttamente, che sia o meno giusto ordine. I messaggi fuori sequenza sono

archiviati in memoria fino a quando non vengono ricevuti tutti i messaggi precedenti. Questa tecnica implica che sia mantenuto un buffer in ricezione.

In generale, il meccanismo *Go-Back-N* è più efficiente rispetto allo *Stop and Wait* a fronte di un'elaborazione supplementare minima. Il *Selective repeat* ha un'efficienza leggermente maggiore del *Go-Back-N* ma i costi in termini di memoria ed elaborazione solo tali da far preferire il *Go-Back-N*.

2.2.1 La tecnica sliding window

La tecnica di sliding window è un'evoluzione del meccanismo a finestra in quanto se il trasmettitore riceve un ACK prima di terminare gli invii delle sequenze di dati previsti all'interna della finestra, può continuare a trasmettere. Infatti il trasmettitore utilizza un *contatore modulo* 2^k, dove *k* è il numero di bit che usa per numerare i messaggi inviati (ogni messaggio è numerato da 0 a k-1, poi da 0 a k-1, e così via) e 2^k-1 è la dimensione della finestra.

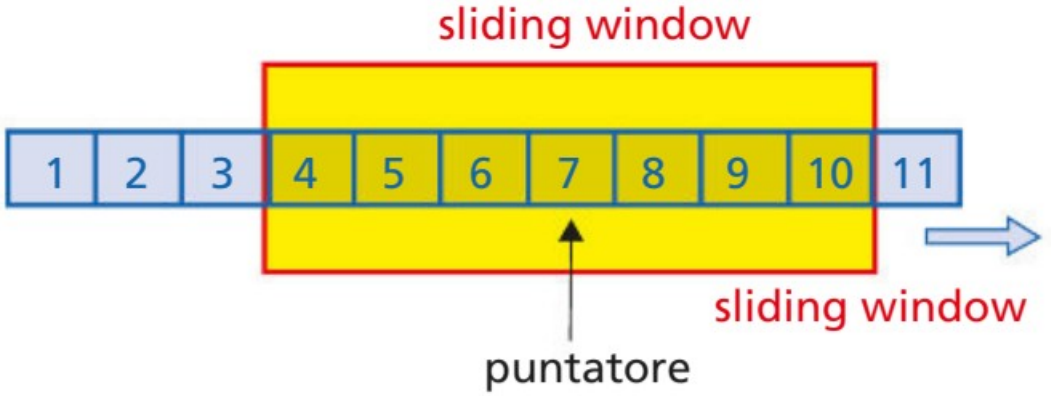
Un fattore critico di questa tecnica è la scelta del parametro k, infatti al crescere di k aumenta il numero di messaggi che possono essere contemporaneamente presenti sulla linea e aumenta la quantità di risorse che il trasmettitore e ricevitore devono riservare per la loro gestione. D'altra parte, se la finestra è troppo piccola rispetto al tempo medio di trasmissione, può succedere che il trasmettitore sia costretto all'inattività per un tempo non indifferente.

Esempio:

dimensione finestra = **n** = 7

$2^k - 1 = n \Rightarrow \mathbf{k} = 3$

a) Inizio trasmissione con finestra di dimensione 7



b) Dopo la conferma di ricezione dei primi 3 byte, la finestra viene traslata, il puntatore punta al successivo byte da spedire

Si supponga che ogni rettangolo corrisponda a una sequenza di bit da trasmettere: una finestra pari a 7 significa che possono essere inviate fino a 7 sequenze senza riceverne il riscontro (a).

Quando il mittente riceve la conferma dell'avvenuta ricezione delle prime 3 sequenze fa scorrere la finestra di 3 posizioni (b). Il mittente usa un puntatore per distinguere quali sequenza ha già inviato (da 4 a 6) e quali devono essere ancora spedite; di queste ultime potrà ancora spedire solo quelle dal numero 7 al numero 10, in quanto l'11 è "fuori" dalla finestra.

Si noti che nel trasmettitore il confine della finestra a sinistra si muove verso destra di una posizione ogni volta che una sequenza di dati è stata inviata, mentre il confine di destra si muove verso destra quando il trasmettitore riceve un ACK e si sposta di un numero di posizioni pari alle sequenze confermate dell'ACK.

3 Livello rete

Il livello Network ha i seguenti compiti fondamentali:

- **Instradamento dei messaggio** su una rete utilizzando un indirizzamento univoco
- localizzazione degli eventuali **instradamenti alternativi in caso di guasti**

Un protocollo di livello network deve pertanto conoscere la topologia della rete, scegliere di volta in volta il cammino migliore e gestire le eventuali problematiche derivanti dalla presenza di più reti con tecnologie di livello Physical diverse. Questo

livello è il primo a garantire una connettività con internet, quindi deve essere in grado di identificare univocamente ogni stazione sulla rete **mediante un ID** apposito.

È in grado di offrire servizi connessi e non connessi (**connection oriented**, **connectionless**).

Il principale protocollo del livello di rete è **IP** nelle versioni 4 e 6: protocollo di instradamento che si occupa dell'indirizzamento, della suddivisione in pacchetti e del trasferimento dei dati che arrivano al livello trasporto. Questo protocollo è connectionless, dunque consente a due host di scambiarsi pacchetti, senza essere connessi. Ovviamente la *consegna non è garantita*, ma di questo se ne occupa il protocollo TCP del livello trasporto.

Il protocollo IP aggiunge ai dati (chiamati payload) un header, grande al massimo 20 byte, per formare il pacchetto da mandare al livello fisico, di massimo 65535 byte. I campi più importanti dell'header sono rappresentati dagli **indirizzi IP del mittente e del destinatario**.

3.1 Header IP

La composizione dell'header:

- **Version**: 4 bit che rappresentano il protocollo (0100 = v. 4). Se l'host destinatario non è grado di gestire questo protocollo, il pacchetto viene scartato
- **HLEN**: 4 bit che indicano la lunghezza dell'header IP. Tutti i campi dell'header sono di lunghezza fissa tranne options e padding.
- **TOS** (***Type Of Service***): 8 bit che servono a far capire al protocollo come gestire il pacchetto. È formato da 5 sottocampi:
  - **Precedence**: 3 bit che indicano la priorità del pacchetto, da 0 (normale) a 7 (controllo di rete)
  - **Delay**: un bit che, se segnato a 1, indica che ci deve essere un ritardo minimo

    - **Throughput**: un bit che, se segnato a 1, indica che si vuole massimo trhoughput
    - **Reliability**: se impostato a 1 signfica che vogliamo la massima affidabilità
    - **Monetary Cost**: se impostato a 1 significa che vogliamo percorrere il cammino dal costo minimo
    - **Unused**: 1 bit inutilizzato. Nella v. 4 è impostato a 0 sempre
- **Total length**: 16 bit che contengono la lunghezza totale del pacchetto (datagram, dato da header + payload) espressa in byte, che potrà essere al massimo 2 16 – 1 = 65.535
- **ID**: 16 bit che identificano univocamente i frammenti di un medesimo datagram. A volte può essere necessario per un router frammentare un datagram per consentirgli di attraversare una rete con caratteristiche diverse da quelle di provenienza.
- **Flags**: 3 bit di controllo per la frammentazione
    - il primo non è attualmente utilizzato
    - il secondo è detto **DF** (***D**on't **f**ragment*): se impostato a 1 indica che il datagram non può essere frammentato
    - il terzo è detto **MF** (***M**ore **f**ragment*): se impostato a 1 indica che il frammento è seguito da altri frammenti. Solo l'ultimo frammento avrà quindi MF = 0
- **Fragment Offset**: 13 bit che indicano l'offset del frammento rispetto all'inizio del datagram. Il valore è fissato dal router
- **TTL** (***T**ime **T**o **L**ive*): 8 bit inizializzati al numero massimo di hop (salti) che il datagram può effettuare. Il valore viene decrementato di 1 ogni volta che il datagram attraversa un router. Quando sarà 0, il pacchetto verrà scartato
- **Protocol**: 8 bit usati per indicare quale protocollo di livello superiore è stato utilizzato per creare il payload. Ogni protocollo è identificato da un PIN (Protocol Identification Number) assegnato dal NIC (Network Information Center), ad esempio: 1 = ICMP, 2 = IGMP, 6 = TCP, 17 = UDP

- **Header Checksum**: 16 bit, somma di controllo relativa solo all'header. A ogni router attraversato viene ricalcolato essendosi modificato l'header per via del campo TTL che subisce un decremento.
- **Source IP Address**: 32 bit dell'indirizzo IP del mittente
- **Destination IP Address**: 32 bit dell'indirizzo IP del destinatario
- **Options**: ogni opzione è lunga 8 bit e un datagram può contenere più opzioni. Gli 9 bit di un'opzione sono suddivisi in 3 campi:
  - **Copy flag**: 1 bit che impostato a 0 indica, in caso di frammentazione, l'opzione va copiata solo sul primo frammento; se impostata a 1 viene copiata su tutti i frammenti
  - **Option Class**: 2 bit che al valore 0 indicano che l'opzione è di controllo del datagram o della rete; al valore 2 indicano che l'opzione serve per debug o misurazioni; i valori 1 e 3 sono riservati a usi futuri
  - **Option Number**: 5 bit che identificano l'opzione nell'ambito della Option Class di appartenenza
- **Padding**: riempitivo con dati fittizi la cui dimensione dipende dal numero di opzioni presenti. È usato per rendere l'header di lunghezza multipla di 32 byte.

3.2 Struttura indirizzi IP

Il protocollo IP fornisce l'indirizzo logico degli host di una rete TCP/IP. A ciascuno host viene assegnato un **indirizzo IP univoco** rispetto alla rete su cui sta lavorando.

Quindi l'indirizzo IP assegnato ad un host non solo rappresenta l'host, ma indica anche su quale sottorete logica si trovi, consentendo insieme alla subnet mask l'inoltro dei pacchetti da parte dei router solo quando è necessario.

In realtà un indirizzo IP non identifica un host individuale, ma un'**interfaccia di rete**.

Gli IP address v4 sono numeri di 32 bit suddivisi in 4 byte (detti *ottetti*). Vengono solitamente espressi nella notazione decimale costituita da 4 numeri decimali compresi tra 0 e 255, separati da un punto.

192.168.1.20

3.2.1 Classi degli indirizzi IP

I quattro ottetti che contengono gli indirizzi IP sono suddivisi in cinque classi: **A**, **B**, **C**, **D**, **E**, ma solo le prime 3 classi possono essere utilizzate per assegnare indirizzi agli host.

- **Classe A**
  - Ha il primo bit del primo ottetto fisso al valore 0 (quindi il valore massimo sarà **0**1111111 = 127)
  - Dedica il primo ottetto alla rete e gli altri 3 agli host (Network, Host, Host, Host)
  - Ha 7 bit dedicati alla rete ma può indirizzare solo 2 7 - 2 = 126 reti perché i valori 0 (this network) e 127 (loopback net) non possono essere assegnati in quanto indirizzi speciali
  - Ha 24 bit dedicati agli host e può indirizzare 2 24 – 2 = 16.774.214 host per ogni rete perché i valori 0.0.0 (this host) e 255.255.255 (broadcast) non possono essere assegnati in quanto indirizzi speciali
  - gli indirizzi di classe A sono adatti a network di grandi dimensioni
- **Classe B**
  - Ha i primi due bit del primo ottetto fissi al valore 10 (quindi il valore massimo **10**111111 = 191)
  - Dedica i primi 2 ottetti alla rete e gli altri due agli host (Network, Network, Host, Host)
  - Range: 128.0.0.0 a 191.255.255.255
  - Ha 14 bit dedicati alla rete e può indicizzare 2 14 = 16.384 reti

- Ha 16 bit dedicati agli host e può indicizzare 2 16 – 2 = 65.534 host per ogni rete perché i valori 0.0 (this host) e 255.255 (broadcast) non possono essere assegnati in quanto indirizzi speciali
- Gli indirizzi di classe B sono adatti a network di medie dimensioni

- **Classe C**
  - Ha i primi 3 bit del primo ottetto fissi al valore 110 (quindi il valore massimo **110**11111 = 223)
  - Dedica i primi 3 ottetti alla rete e l'ultimo agli host (Network, Network, Network, Host)
  - Range: 192.0.0.0 a 223.255.255
  - Ha 21 bit dedicati alla rete e può indicizzare 2 21 = 2.097.152 reti
  - Ha 8 bit dedicati agli host e può indicizzare 2 8 – 2 = 254 host per ogni rete perché i valori 0 (this host) e 255 (broadcast) non possono essere assegnati in quanto indirizzi speciali
  - Gli indirizzi di classe C sono adatti a network di piccole dimensioni
- **Classe D**
  - Ha i primi 4 bit del primo ottetto fissi al valore 1110 (quindi valore massimo: **1110**1111 = 239)
  - Range: 224.0.0.0 a 239.255.255.255
  - Non sono indirizzi assegnabili ai singoli host
  - Servono per il multicasting cioè ad indirizzare gruppi di host (per esempio un'intera rete)
- **Classe E**
  - Ha i primi 4 bit del primo ottetto fissi al valore 1111 (quindi valore massimo **1111**1111 = 255)
  - Range: 240.0.0.0 a 255.255.255.254 (è escluso tutti 1, cioè 255.255.255.255)

- Non sono indirizzi assegnabili ai singoli host
- Sono indirizzi riservati per usi futuri

3.3 Indirizzi speciali

Esistono degli indirizzi che non possono essere assegnati agli host di una rete.

- **Indirizzi di rete**: sono quegli indirizzi che hanno tutti 0 nella parte dedicata agli host:
  - **Classe A**

    X.0.0.0

    Esempio:

    100.0.0.0 <= indirizzo di rete di classe A
  - **Classe B**

    X.Y.0.0

    Esempio:

    129.32.0.0 <= indirizzo di rete di classe B
  - **Classe C**

    X.Y.Z.0

    Esempio:

    192.168.1.0 <= indirizzo di rete di classe C
- **Indirizzi di broadcast**: sono quegli indirizzi che hanno tutti 1 nella parte dedicata agli host. Sono indirizzi usati per mandare pacchetti a tutti gli host di quella rete (broadcast limited: riferiti solo alla rete locale specificata).
  - **Classe A**

    X.255.255.255

Esempio:

100.255.255.255 <= indirizzo di broadcast di classe A

- **Classe B**

  X.Y.255.255

  Esempio:

  129.32.255.255 <= indirizzo di broadcast di classe B

- **Classe C**

  X.Y.Z.255

  Esempio:

  192.168.1.255 <= indirizzo di broadcast di classe C

- **Indirizzo di rete di default**: è l'indirizzo in cui tutti i byte sono settati a 0 (0.0.0.0) ed è usato per il routing o per identificare l'host corrente in fase di bootstrap
- **Indirizzo di broadcast di default**: è l'indirizzo in cui tutti i byte sono settati a 255 (255.255.255.255) ed è usato per inviare pacchetti a tutta la rete corrente. È anch'esso un indirizzo di tipo broadcast limited essendo riferito alla rete locale corrente
- **Indirizzo di loopback**: è l'indirizzo localhost, 127.0.0.1, e serve per identificare se l'host è correttamente configurato rispetto al protocollo TCP/IP, quando ancora non gli è stato assegnato un indirizzo IP. Rappresenta l'indirizzo IP dell'host stesso

3.4 Indirizzi pubblici/privati e statici/dinamici

Gli indirizzi che si affacciano sulla rete sono detti pubblici e sono univoci in tutto il pianeta.

Poiché il numero degli indirizzi IP non è sufficiente per indirizzare tutti gli host esistenti (2 32 = 4.294.967.292 indirizzi possibili) sono stati riservati dei **range di indirizzi privati** per ogni classe. Questi indirizzi non possono essere utilizzati per affacciarsi direttamente alla rete pubblica ma servono per indirizzare gli host di rete private.

I range di indirizzi privati sono definiti dalla RFC 1918 e valgono:

- **Classe A**: da 10.0.0.0 a 10.255.255.255
- **Classe B**: da 172.16.0.0 a 172.31.255.255
- **Classe C**: da 192.168.0.0 a 192.168.255.255 (usabile anche come se fosse classe B avendo gli ultimi due ottetti a 0)

Un'altra tecnica utilizzata per sopperire allo scarso numero di indirizzi IP a disposizione è quello di assegnare, in particolare agli utenti privati, degli **indirizzi dinamici**, cioè degli indirizzi che cambiano ogni volta che ci si collega a internet. In questo modo gli ISP (Internet Service Provider) possono utilizzare uno stesso indirizzo IP pubblico per più utenti in momenti diversi, sfruttando il fatto che difficilmente un utente privato resta collegato 24/h a internet.

Alle aziende vengono invece solitamente assegnati degli **indirizzi statici**, cioè fissati una volta per tutte, che tali aziende utilizzeranno per collegarsi a internet, quindi usati come indirizzi pubblici per connettere tutti gli host dell'azienda aventi indirizzi privati.

Lo **IANA** (***I**nternet **A**ssigned **N**umbers **A**uthority*) è l'organismo responsabile dell'assegnazione degli indirizzi IP pubblici.

È parte integrante dello **IAB** (***I**nternet **A**rchitecture **B**oard*).

Lo **IANA** delega la gestione di blocchi di indirizzi IP a enti locali denominati **RIR** (***R**egional **I**nternet **R**egistry*).

Ogni RIR assegna gli indirizzi per una specifica zona del mondo. Al momento esistono cinque di questi registri nel mondo, ciascuno con la sua area di competenza.

3.5 Subnetting

Per ottimizzare il traffico in una rete risulta particolarmente utile suddividerla in una serie di sottoreti logiche, collegate tra loro da router interno alla rete stessa.

Questa operazione di segmentazione della rete in sottoreti prende il nome di **subnetting** ed è realizzata "sacrificando" alcuni dei bit che le classi A, B, C dedicano agli host per definire un indirizzo di sottorete.

Per segmentare una rete occorre, in fase di progettazione, stabilire quante subnet servono, e di conseguenza quanti bit occorrono per indirizzarle univocamente.

Se per esempio servono 50 subnet, ci vorranno 6 bit, essendo 2 6 = 64 quindi maggiore di 50.

In generale, posto X il numero di subnet richieste, si avrà che il numero N il numero di bit necessari a indirizzarle univocamente è dato da:

$$N = \lfloor \log 2 X \rfloor + 1$$

cioè parte intera del log in base 2 di X, più 1.

Occorre anche ricordare che bisogna evitare che nella parte subnet e in quella host vi siano contemporaneamente tutti 0 o tutti 1 perché diventerebbero indirizzi speciali, dedicati rispettivamente alla rete e al broadcast. Per semplificare e non incorre in errori, spesso si consiglia di evitare la subnet 0 e quella con tutti gli N bit a 1 e indirizzare solo 2 N – 2 sottoreti.

Oltre a questo si deve definire una nuova stringa da 32 bit che prende il nome di **subnet mask** e che di default ha valore:

- 255.0.0.0 per Classe A (111111111.0.0.0)
- 255.255.0.0 per Classe B (11111111.11111111.0.0)
- 255.255.255 per Classe C (111111111.111111111.111111111.0)

Notare il fatto che le subnet mask di default hanno settatto tutti i bit a 1 nella parte dedicata agli host.

Le maschere di sottorete, dopo il subnetting, devono avere tutti i bit a 1, oltre che la parte di rete, anche la parte dedicata alle sottoreti create.

Per esempio le 50 subnet, che necessitano di 6 bit per essere indirizzate, originano le seguenti possibili subnet mask a seconda della classe di appartenenza:

- 255.252.0.0 per Classe A (6 bit per subnet, 18 per gli host)
- 255.255.252.0 per Classe B (6 bit per subnet, 10 per gli host)
- 255.255.255.252 per Classe C (6 bit per subnet, 2 per gli host)

I bit a 1 vanno scritti da sinistra verso destra nel primo ottetto che non è dedicato alla network. Come detto, il subnetting e le subnet mask hanno il fondamentale scopo di ottimizzare il traffico evitando, per esempio, che pacchetti inviati da un host ad un altro host residente nella stessa sottorete escano e rientrino dalla sottorete medesima per giungere a destinazione.

Il meccanismo che consente tutto ciò è detto processo di messa in **AND bit a bit** (**Anding process**):

1. Un'operazione di AND bit a bit tra l'indirizzo IP del mittente e la subnet mask del mittente ottenendo l'ID net e l'ID subnet del mittente e azzerando l'ID host
2. Un'operazione di AND bit a bit tra l'indirizzo IP del destinatario e la subnet mask mittente ottenendo l'ID net e l'ID subnet del destinatario e azzerando l'ID host
3. Il confronto tra i due risultati ottenuti:
    - Se sono uguali allora mittente e destinatario sono nella stessa subnet (comunicazione diretta)
    - Se sono diversi mittente e destinatario non sono nella stessa subnet (comunicazione attraverso uno degli switch o router/switch della rete)

- Esempio:

Supponiamo di avere due host di classe B e di aver utilizzato 8 bit per mascherare le subnet:

host A => IP_A = 150.169.3.8

host B => IP_B = 150.169.5.2

Subnet mask (SM): 255.255.255.0 uguale per tutte le subnet della

rete

1. IP_A & IP_B

10010110.10101001.00000011.00001000

11111111.11111111.11111111.00000000

10010110.10101001.00000011.00000000

2. IPB & SM:

10010110.10101001.00000101.00000010

11111111.11111111.11111111.00000000

10010110.10101001.00000101.00000000

3. I risultati delle due operazioni hanno prodotto due risultati diversi, quindi i due host non si trovano nella stessa subnet.

10010110.10101001.00000011.00000000

10010110.10101001.00000101.00000000

Dunque il pacchetto andrà inoltrato al di fuori della sottorete.

È possibile riassumere la coppia indirizzo IP e subnet mask mediante la **slash notation** in cui all'indirizzo IP viene fatto seguire il *prefix length*, un numero decimale che indica il numeri di bit a 1 della maschera.

Per esempio:

150.169.3.2 / 24

indica che la maschera ha 24 bit a 1, e cioè vale 255.255.255.0.

Se la maschera avesse, per esempio, 26 bit a 1, allora la subnet mask diventerebbe:

11111111.11111111.11111111.11000000

4 Livello trasporto

Grazie ad **IP** (***I**nternet **P**rotocol*) possiamo trasferire i pacchetti attraverso internet. Una volta arrivato a destinazione, però, l'header del datagram IP non contiene alcuna informazione utile all'host ricevente per individuare l'applicazione o l'utente destinatario del messaggio.

4.1 Indirizzi a livello trasporto

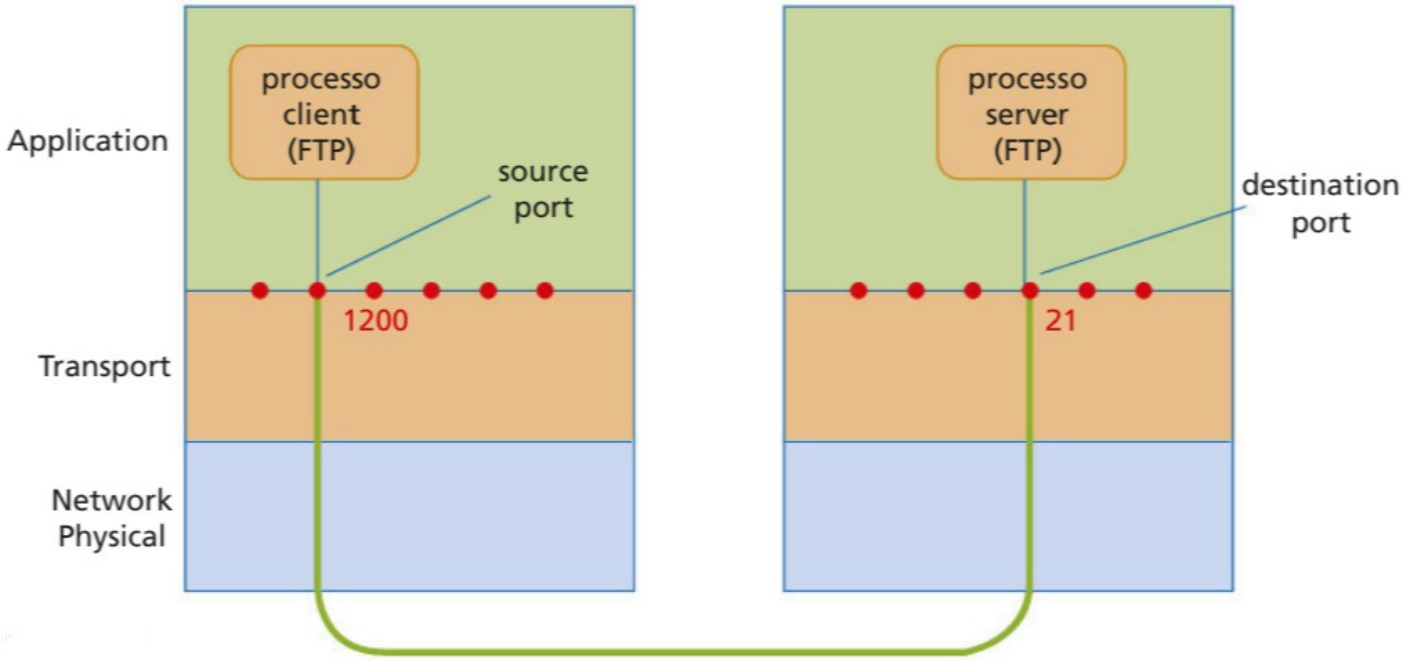
Tipicamente i sistemi operativi sono multi-tasking e multi-user, consentendo al computer di svolgere più task contemporaneamente. Quindi nel momento in cui l'host riceve un datagram, il **destinatario finale è uno dei processi attivi** e può riferirsi sia a un programma applicativo di sistema, sia a un programma applicativo utente. Quindi a questo punto il problema è quello di consegnare il messaggio all'applicazione finale, dato che i processi vengono creato ed eliminati dinamicamente, non possono quindi essere noti ai potenziali mittenti.

Le applicazioni finali devono poter essere individuate in base alla loro funzione, non in base al processo. Seguendo questa soluzione si sono stabiliti dei *punti di accesso ai quali* consegnare i pacchetti che arrivano, chiamati **porte** (ogni porta è identificata da un numero a 16 bit, quindi 0-65535). I **numeri di porta** sono assegnati a livello internazionale da **IANA** e suddivisi in 3 gruppi: **Well Known Ports** (0-1023), **Registered Ports** (1024-49151), **Dynamic and/or Private Ports** (49152-65535).

L'host mittente, che deve inviare un messaggio a una destinazione, deve conoscere IP e porta di destinazione. Quindi ogni messaggio deve contenere al suo interno:

- **destination port**: numero della porta di destinazione

- **source port**: numero della porta presente sull'host mittente, sulla quale è in attesa il processo che riceve le riposte dal destinatario



4.2 Servizi del livello trasporto

I protocolli implementati a livello trasporto svolgono funzioni simili a quelli del livello DataLink. La differenza fondamentale è lo **scenario di rete in cui operano**: a livello DataLink la connessione tra il router, che invia un pacchetto, ad un altro è diretta, invece a livello trasporto la connessione avviene attraverso l'intera rete, cioè viene creato un **canale logico di trasmissione**[4] che unisce host mittente ad host destinatario, non a caso si dice che questo livello si occupa della comunicazione **end-to-end**, dove gli estremi della connessione sono appunto i 2 host.

Il livello trasporto svolge quindi una funzione "*cuscinetto*" tra il livello Application e livelli inferiori, che si occupano della trasmissione in rete dei dati. In questo modo le applicazioni non necessitano di conoscere la rete, né il computer di destinazione, né il percorso che prenderanno dati, né quanto e grande la rete.

Esistono vari protocolli di trasporto che sono stati standardizzati per soddisfare le differenti esigenze applicative, tra cui i più diffusi sono **UDP** (***User Data Protocol***) e

4 **Canale logico di trasmissione**: canale virtuale

TCP (***T**rasmission **C**ontrol **P**rotocol*). I pacchetti UDP sono chiamati *datagram*, mentre i pacchetti TCP sono chiamati *segmenti*.

I protocolli del livello trasporto mettono in comunicazione le applicazioni (o processi) offrendo un servizio denominato **Multiplexing/Demultiplexing**, insieme al controllo dell'integrità dei dati. Inoltre, TCP fornisce anche la *garanzia* di *consegna dei dati*, effettuando il controllo della gestione e il controllo di flusso.

4.3 Multiplexing e Demultiplexing

Si definisce **socket** una **coppia di punti di accesso**, formata dalla coppia **IP:porta**.

Attraverso i socket si realizzano le funzionalità:

- **multiplexing**: *in trasmissione* il livello trasporto riceve i dati dai socket e gli aggiunge il proprio header
- **demultiplexing**: *in ricezione* il livello trasporto determina a quale socket consegnare i dati

Queste operazioni possono avvenire in presenza o meno di una connessione:

- **multiplexing/demultiplexing connectionless**: è il caso dell'UDP in cui è previsto che più client accedano allo stesso servizio sullo stesso server. Il socket è individuato da *ServerIP:Porta*
- **multiplexing/demultiplexing connection-oriented**: è il caso del TCP in cui è previsto che più client accedano allo stesso servizio sullo stesso server e che uno stesso client possa attivare più sessioni dello stesso servizio. Il socket è individuato: *ClientIP:ServerIP:ClientPort:ServerPort*.

Si noti il fatto che non vengono più utilizzate le terminologie host mittente o host destinatario, in quanto è proprio a livello transport che si inizia ad individuare come la comunicazione avvenga tipicamente tra un'applicazione client e una server.

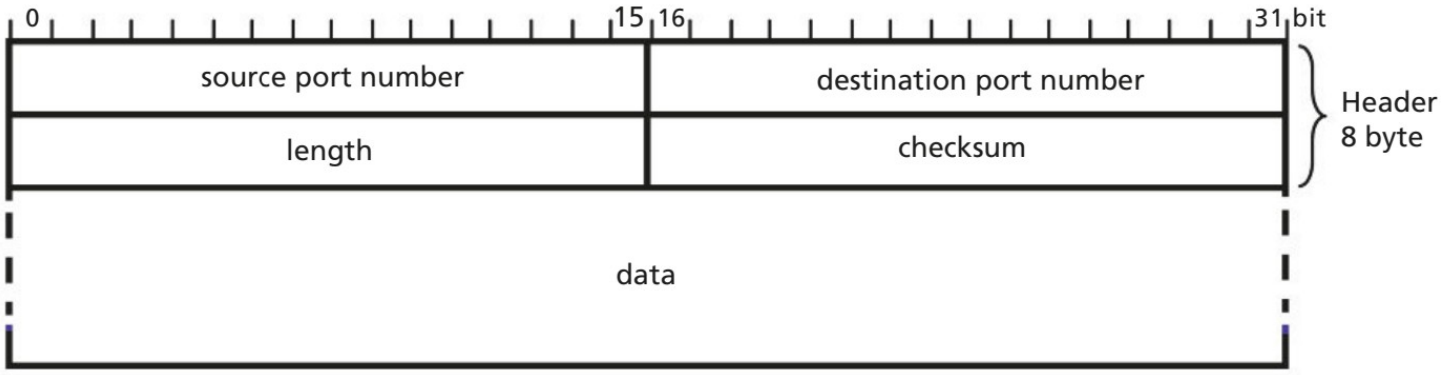
4.4 UDP

UDP (***User Datagram Protocol***) è un protocollo del livello Transport che non prevede l'uso di una connessione, infatti ciascun datagram UDP è trattato in modo indipendente.

Il servizio offerto da UDP è di tipo **Best Effort**: i datagram UDP possono essere persi o arrivare fuori sequenza, non si ha quindi alcuna garanzia sulla consegna dei dati trasmessi.

A prima vista quindi sembrerebbe non offrire un servizio diverso da quello dell'IP, in realtà non è così: UDP fornisce le funzionalità tipiche del livello trasporto in termini di multiplexing, grazie all'uso delle porte, e di controllo dell'integrità dei dati (anche se opzionale).

4.4.1 Datagram UDP



I campi del datagram sono:

- **source port number**: 16 bit, numero di porta sull'host mittente
- **destination port number**: 16 bit, numero di porta sull'host destinatario
- **length**: 16 bit, contiene la lunghezza totale in byte del datagram UDP
- **checksum**: 16 bit e opzionale, contiene il codice di controllo del datagram, di solito CRC
- **data**: contiene le informazioni trasmesse/ricevute

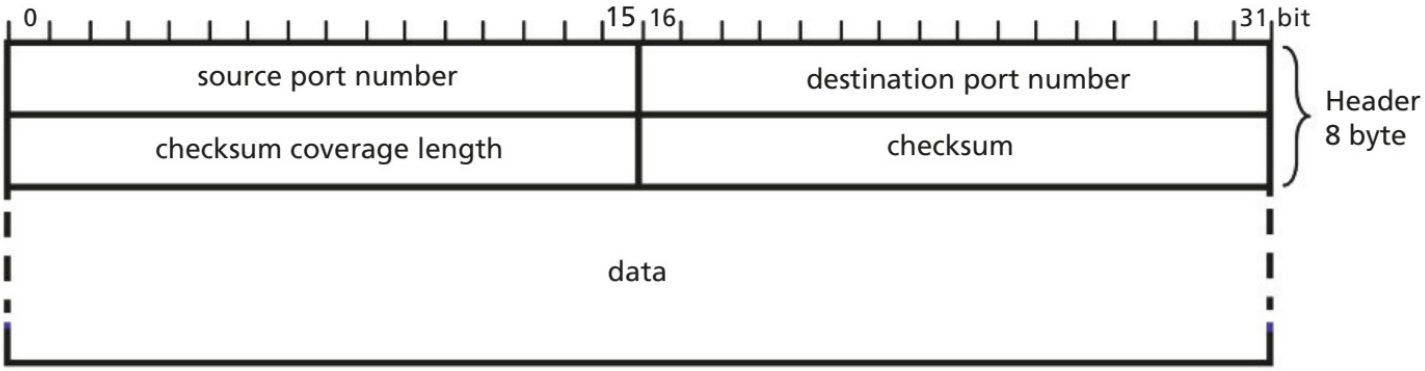
La **dimensione massima** di un datagram UDP è 65508 byte. Infatti deve essere contenuto in un pacchetto IP che ha al massimo 65536 byte ai quali si devono togliere 20 byte minimi dell'header IP, che porta ad avere un campo dati IP al massimo di 65516 byte ai quali si devono ancora togliere gli 8 byte dell'header UDP.

Ci sono alcuni vantaggi derivanti dall'uso di UDP:

- **non richiede di stabilire una connessione**, non introducendo dovuto alla fase di set-up della connessione
- **non mantiene lo stato della connessione**: un server può supportare molti più client attivi
- **il sovraccarico** dovuto all'intestazione del pacchetto **è minimo**
- il controllo del livello applicativo è più efficacie: in mancanza di un controllo della congestione, il mittente non viene mai bloccato

– UDP-Lite

Nel corso degli ultimi anni è sorta l'esigenza di *non scartare* i datagram UDP che risultano errati dopo il controllo del campo checksum. Infatti, per certi tipi di applicazioni, ricevere parte dei dati è meglio che non ricevere nulla. Tipici esempi sono le applicazioni VoIP e di streaming audio/video che usano pacchetti con una gran quantità di dati. Con il protocollo UDP tradizionale, è sufficiente avere un byte errato per scartare tutto il datagram, mentre con **UDP Lite** su può salvare la parte di dati che sono arrivati corretti.



Il *checksum coverage length* specifica quanti byte del datagram UDP saranno controllati (solo 8 oppure tutti). La lunghezza del datagram UDP si deduce dalla lunghezza del pacchetto IP + 8 byte dell'header UDP.

4.5 TCP

TCP (***T**rasmission **C**ontrol **P**rotocol*) è un protocollo di trasporto più diffuso dell'UDP, in quanto offre un servizio **connection-oriented** e **affidabile**, garantendo quindi la consegna dei dati in modo ordinato (infatti si parla di *stream-oriented*, tutti i pacchetti arrivano come sono stati inviati e nello stesso ordine).

La connessione che TCP stabilisce tra mittente e destinatario offre all'applicazione l'impressione che ci sia un canale dedicato; quindi la connessione è intesa come un canale logico le cui caratteristiche sono:

- è **full-duplex**: sulla stessa connessione si può trasmettere e ricevere contemporaneamente
- è **point-to-point**: un solo mittente e un solo destinatario

TCP usa più risorse del computer host rispetto al protocollo UDP, sia in termini di CPU sia in termini di memoria. Inoltre necessità di maggiore capacità trasmissiva (banda) per via della ritrasmissione e dell'header più grande (20 byte rispetto agli 8 dell'UDP).

4.5.1 La comunicazione tra TCP e processo applicativo

Tipicamente i sistemi operativi permettono di accedere alle porte in **modo sincrono**, ciò implica che l'esecuzione del processo si interrompa durante un'operazione di accesso alla porta.

A volte i dati ricevuti vengono scartati perché:

- il processo di destinazione non è pronto a ricevere o quella porta scelta non esiste
- **non c'è spazio sufficiente nel buffer** di ricezione per contenere tutti i dati

Le informazioni di controllo che l'applicazione deve passare al TCP sono:

- **source address**: indirizzo completo del mittente (network+host+port)
- **destination address**: indirizzo completo del destinatario

- **next packet sequence number**: il numero di sequenza che TCP deve assegnare al prossimo pacchetto che trasmetterà
- **current buffer size**: dimensione del buffer del mittente
- **next write position**: indirizzo dell'area del buffer in cui il processo pone i nuovi dati da trasmettere
- **next read position**: indirizzo dell'area del buffer da cui TCP deve leggere i dati per costruire il prossimo segmento da inviare
- **timeout/flag**: indica il tempo, trascorso il quale, i dati non riscontrati (non viene mandato un ACK di conferma ricezione) devono essere ritrasmessi; il flag è usato per sincronizzare TCP e processo (ex. tramite *semafori*)

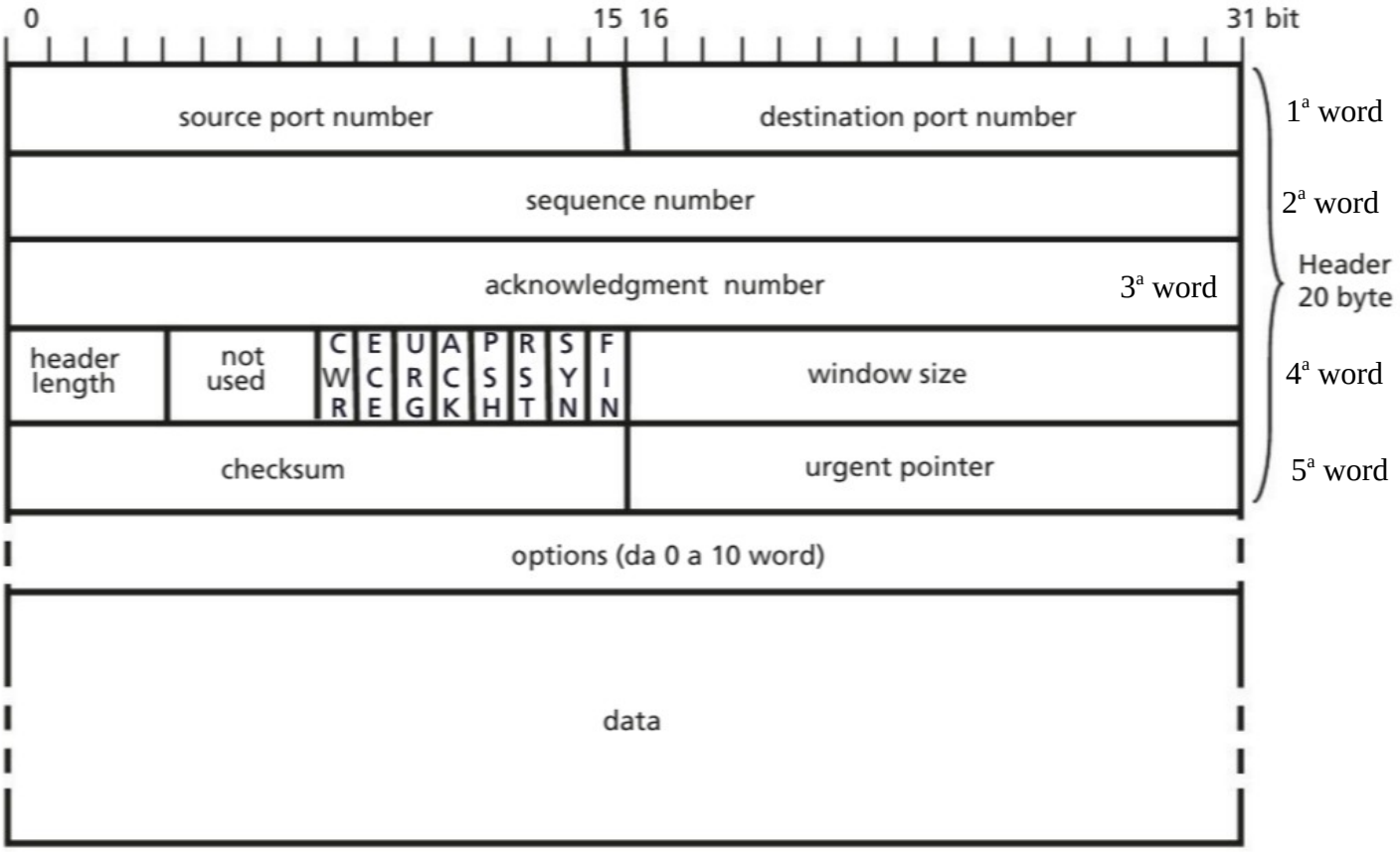
- Buffer

Quando un'applicazione passa dei dati al TCP, quest'ultimo può salvarli in un buffer oppure spedirli subito; la **bufferizzazione** consente di migliorare l'efficienza della comunicazione: i dati verranno spediti quando si raggiunge una certa quantità.

Lo standard prevede due eccezioni alla bufferizzazione:

- l'applicazione richiede che i dati vengano spediti immediatamente, allora imposta a 1 il flag *PUSH*, nell'header TCP (al contrario, in ricezione, il pacchetto viene passato direttamente all'applicazione)
- l'applicazione imposta a 1 il flag *URG* dell'header TCP: i dati non vengono accumulati nel buffer e TCP trasmette immediatamente ciò che riguarda quella connessione. In ricezione, quando arrivano i dati con flag URG settati a 1, l'applicazione interrompe la sua attività per esaminare immediatamente i dati urgenti

4.5.2 Segmento TCP



Ecco una descrizione dei campi del segmento TCP:

- **source port number**: 16 bit, numero di porta host mittente
- **destination port number**: 16 bit, numero di porta host destinatario
- **sequence number**: 32 bit, numero di sequenza progressivo del primo byte di dati contenuto nel segmento
- **acknowledgment number**: 32 bit, numero di riscontro, ha significato solo se il flag di ACK è settato a 1, conferma la ricezione di una parte del flusso di dati indicando il valore del prossimo *sequence number*
- **header length**: 4 bit, indica la lunghezza (in word da 32 bit) dell'header; tale lunghezza varia da 5 word (20 byte) a 15 word (60 byte) a seconda della presenza e della dimensione del campo Options (facoltativo). Serve quindi a indicare l'inizio dei dati del segmento
- **not used**: 4 bit, tutto a 0
- **flags** (8 bit): bit utilizzati per il controllo del protocollo:

  - **CWR**: Congestion Window Reduced, se impostato a 1 indica che l'host sorgente ha ricevuto un segmento TCP con il flag ECE impostato a 1 e ha di conseguenza abbassato la sua velocità di trasmissione per ridurre la congestione
  - **ECE**: *ECN-Echo*, se impostato a 1 indica che l'host supporta *ECN* (*Explicit Congestion Notification*) durante il *3-way handshake*
  - **URG**: se impostato a 1 indica che nel flusso sono presente dati urgenti e che deve essere letto il campo *urgent pointer*
  - **ACK**: se impostato a 1 indica che il segmento TCP in questione è *in risposta* a un altro ricevuto, che conteneva dati, di conseguenza indica che il campo *acknowledgment number* è valido e si devono leggere le informazioni contenute
  - **PSH**: *push*, se impostato a 1 indica che i dati in arrivo non devono essere bufferizzati, ma passati subito ai livelli superiori dell'applicazione
  - **RST**: *reset*, se impostato a 1 indica che la connessione non è valida; viene utilizzato in caso di grave errore, a volte utilizzato insieme al flag ACK per chiudere la connessione
  - **SYN**: *synchonize sequence numbers*, è usato nella fase di creazione di una connessione; se impostato a 1 indica che l'host mittente vuole aprire una connessione TCP con l'host destinatario
  - **FIN**: *final*, se impostato a 1 indica che l'host mittente non ha più dati da inviare e vuole chiudere la connessione. Il destinatario ricevente invia la conferma di chiusura con un FIN-ACK.

- **window size**: 16 bit, è usato dall'host destinatario per dire al mittente quanti dati può ricevere in quel momento (*finestra di ricezione*), cioè il numero di byte che il mittente può spedire a partire dal byte confermato (specificato dall'acknowledgment number). Il valore 0 indica di non inviare altri dati per il momento; quando il destinatario sarà di nuovo in grado di ricevere dati invierà al mittente un segmento con window size diverso da 0 ma con lo stesso acknowledgment number
- **checksum**: 16 bit, utilizzato per la verifica della validità del segmento

- **urgent pointer**: 16 bit, ha significato solo se il flag URG è settato a 1, contiene il numero che deve essere sommato (*offset*) al sequence number per ottene il numero dell'ultimo byte urgente nel campo dati
- **options**: campo facoltativo, da 0 bit a 320 bit (40 byte). L'opzione più importante è quella che consente a un host di specificare la dimensione massima del segmento che è in grado di accettare. Di default è 536 byte e 20 byte di header (quindi ogni host deve poter gestire segmenti di almeno 556 byte)
- **data**: contiene i dati

4.5.3 Gestione della congestione

L'architettura TCP/IP adotta un modello Best Effort: la rete fa del suo meglio per consegnare i pacchetti. La conseguenza di questo comportamento è che possono verificarsi delle **congestioni in rete**: la coda di ricezione del router è piena, di conseguenza scarta i nuovi pacchetti arrivati.

Prima di gestire la congestione è necessario rilevarla e a questo scopo TCP utilizza dei **timer** per misurare il tempo trascorso tra l'invio di un pacchetto e il relativo ACK. Se quest'ultimo non arriva entro un dato tempo, si genera un timeout. **TCP presuppone sempre che la cause della perdita sia la congestione**.

TCP lavora applicando insieme diversi algoritmi e configurandone i parametri da usare. Esistono perciò diverse implementazioni del protocollo, che differiscono in base alle opzioni scelte. Tutti questi algoritmi hanno in comune una variabile: la **finestra di congestione**. Viene utilizzata dal mittente per ogni connessione attive e serve per avere un'indicazione sul massimo numero di byte non riscontrati che si possono ancora trovare nella rete.

In particolare:

$$\text{maxWindow} = \min(\text{FinestraCongestione}, \text{FinestraRicezione})$$

Dove:

- **Finestra di congestione**: è il max. numero di byte che la rete è in grado di trasmettere senza che si verifichino dei timeout

- **Finestra di ricezione**: indica quanti byte il destinatario è in grado di ricevere
- **maxWindow**: quando un host deve inviare dei dati prenderà come numero max. di byte che può trasmettere il minimo tra i due valori delle finestre

L'idea di base di questi algoritmi è di diminuire la finestra di congestione quando un pacchetto è scartato dalla rete e aumentarla quando un pacchetto è riscontrato.

Dai 4 algoritmi descritti nell'RFC 2581, si prenderanno in esame soltanto 2, ovvero **slow start** e **congestion avoidance**, che vengono usati solitamente insieme in molte implementazioni di TCP. L'idea alla base è di iniziare a trasmettere lentamente, *esplorando* la rete, per poi accelerare finché non c'è una perdita di dati che comporta un rallentamento nell'invio di nuovi byte.

I punti seguenti spiegano come lavorano i due algoritmi, iniziando con *slow start*:

- al momento della creazione della connessione TCP tra due host, il mittente imposta la finestra di congestione alla massima quantità di byte che la rete può spedire
- ogni volta che viene inviato un segmento TCP, e ricevuto l'ACK di conferma ricezione, il valore della finestra di congestione viene raddoppiato (max. 64 KB)
- da questo in poi la dimensione della finestra di congestione è regolata dall'algoritmo *congestion avoidance* che effettua incrementi di tipo lineari, infatti viene di volta in volta sommato il valore assegnamento inizialmente alla finestra di congestione
- quando si genera un timeout (che per questa versione di TCP significa congestione) si effettuano le seguenti operazioni:
  - la soglia viene impostata alla metà del valore della finestra di congestione
  - la finestra di congestione viene riportata al suo valore iniziale

4.5.4 Fasi di comunicazione TCP

La comunicazione tra mittente e destinatario (tipicamente *client* e *server*) a livello TCP è di tipo connection-oriented, quindi sono previste 3 fasi:

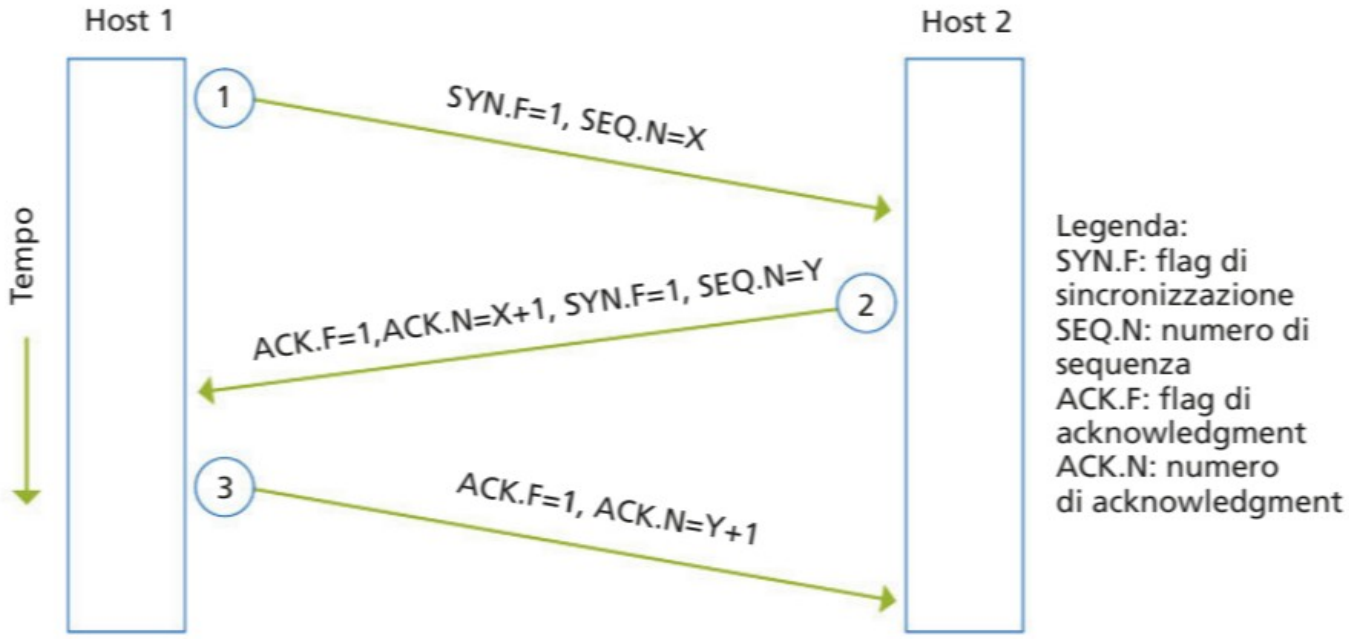
- instaurazione della sessione TCP
- trasmissione dati
- chiusura della sessione TCP

Nella tabella di seguito vengono mostrate le **primitive** usate per implementare i servizi del TCP. Mittente e destinatario comunicano mediante messaggi detti **TPDU** (***T**ransport **P**rotocol **D**ata **U**nit*), anche chiamati **segmenti**.

Primitive	TPDU inviata	Descrizione
accept()	nessuna	blocca finché non arrivano le richieste di connessione da parte dei processi
connect()	Connection Request	prova a stabilire una connessione
send()	Data	invio di informazioni
receive()	nessuna	blocca finché non si riceve una Data TPDU
disconnect()	Disconnection Request	rilascio della connessione da un lato

- Fase di instaurazione di una sessione TCP

Affinché si possa instaurare una sessione TCP tra host 1 e host 2 occorre che quest'ultimo acconsenta mediante una sequenza che prevede 3 passi e che viene chiamata **3 way handshake**.



- L'host 1 invia un segmento TCP con il **flag SYN impostato a 1**. Invia inoltre un numero di sequenza, scelto in modo casuale, che diventa il suo sequence number (X) (primitiva: *connect()*)

- L'host 2 acconsente, risponde con una conferma mediante il flag ACK impostato a 1 e l'**acknowledgment number impostato al valore ricevuto del sequence number + 1** (X+1). Inoltre, per stabilire la connessione nella direzione inversa (da host 2 a host 1) host 2 imposta il proprio SYN a 1 e genera un suo numero di sequenza (Y), scelto in modo casuale, da inviare all'host 1 (primitiva: *send()*)
- L'host 1 risponde con un'ulteriore conferma mediante il flag ACK impostato a 1 e l'**acknowledgment number impostato al valore**, **ricevuto dall'host 2**, **del sequence number + 1** (Y+1) (primitiva: *send()*)

Se sull'host 2 non c'è nessun processo in ascolto sulla porta specificata nel campo destination port number, il secondo passo non viene eseguito e l'host 2 invia un segmento di risposta con flag RST impostato a 1 per rifiutare la connessione.

Nella fase di connessione è quindi importante che ogni host conosca il sequence number dell'altro. Altra informazione che si scambiano è la **dimensione massima del segmento** (*MSS* = Maximum Segment Size) che ogni host invierà all'altro. Verrà scelta la dimensione minore e questo dato sarà particolarmente utile per il controllo della congestione. Infine si scambiano la dimensione della finestra che fornisce indicazioni sulla dimensione del buffer utilizzato per memorizzare i segmenti ricevuti.

MSS = min(MTU, MRU) – 20 byte

In caso di mancanza di informazioni viene utilizzato come valore di default 536 byte, ottenuti nel seguente modo:

576 byte (default IP) – 20 byte (header IP) – 20 byte (header TCP) = 536 byte

MTU (***M**aximum **T**ransfer **U**nit*) è la dimensione massima del campo dati nel frame a livello data link; è un valore che caratterizza la rete di trasmissione utilizzata (per esempio le reti Ethernet MTU = 1500). Ogni volta che IP deve inviare un pacchetto più grande della MTU è costretto a frammentare. TCP tiene conto di questo e, per avere maggiori prestazioni, fa coincidere la dimensione massima del segmento con la MTU.

MRU (***M**aximum **R**eceive **U**nit*), è la MTU del destinatario.

- Fase di trasmissione dati

TCP gestisce il **controllo di flusso** e degli **errori di trasmissione** attraverso il protocollo *sliding windows*.

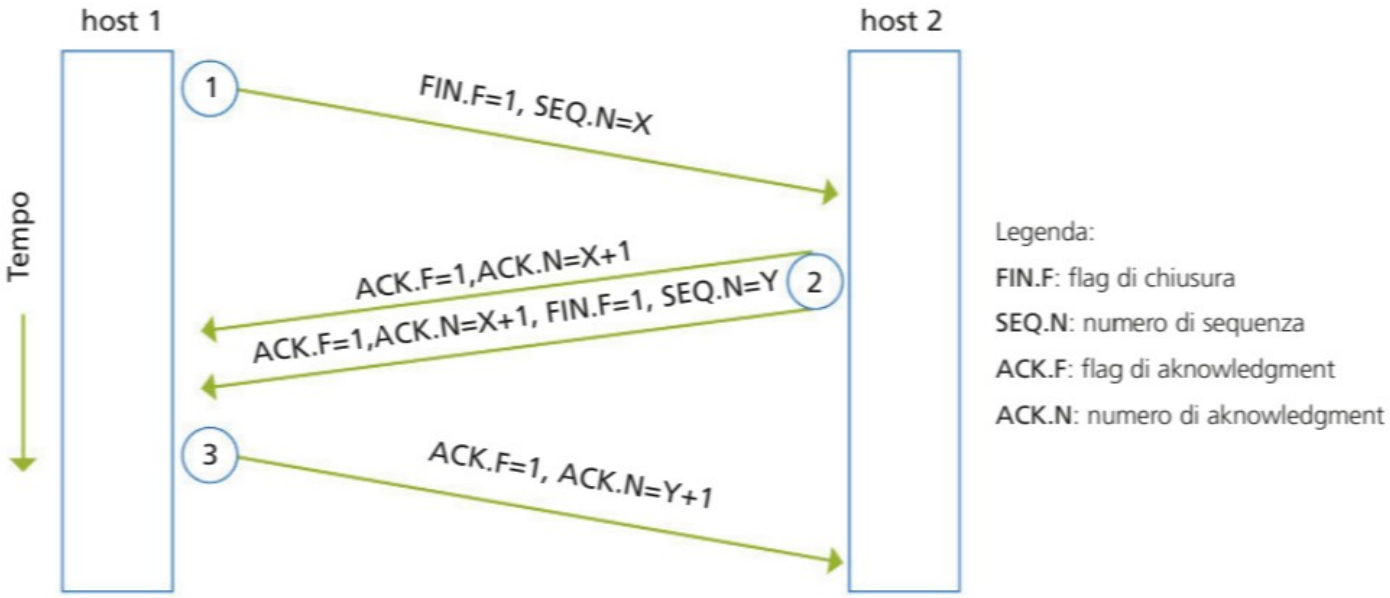
Il protocollo utilizzato è simile a quello usato a livello Data link. Ci sono, però, due differenze:

- in TCP il puntatore nella finestra è al signolo byte, mentre a livello Data link è al frame
- in TCP la dimensione della finestra è variabile, mentre al Datalink è fissa

- Fase di chiusura connessione

Quando l'host 1 non ha più dati da inviare, comunica al TCP di chiudere la connessione e, dopo aver inviato gli eventuali dati rimasti, inizia la procedura di chiusura della connessione.

La chiusura avviene in entrambe le direzioni e mediante una sequenza di 3 passi chiamata **3 way handshake modificato**.



Il processo è molto simile all'instaurazione della connesione, con alcune differenze:

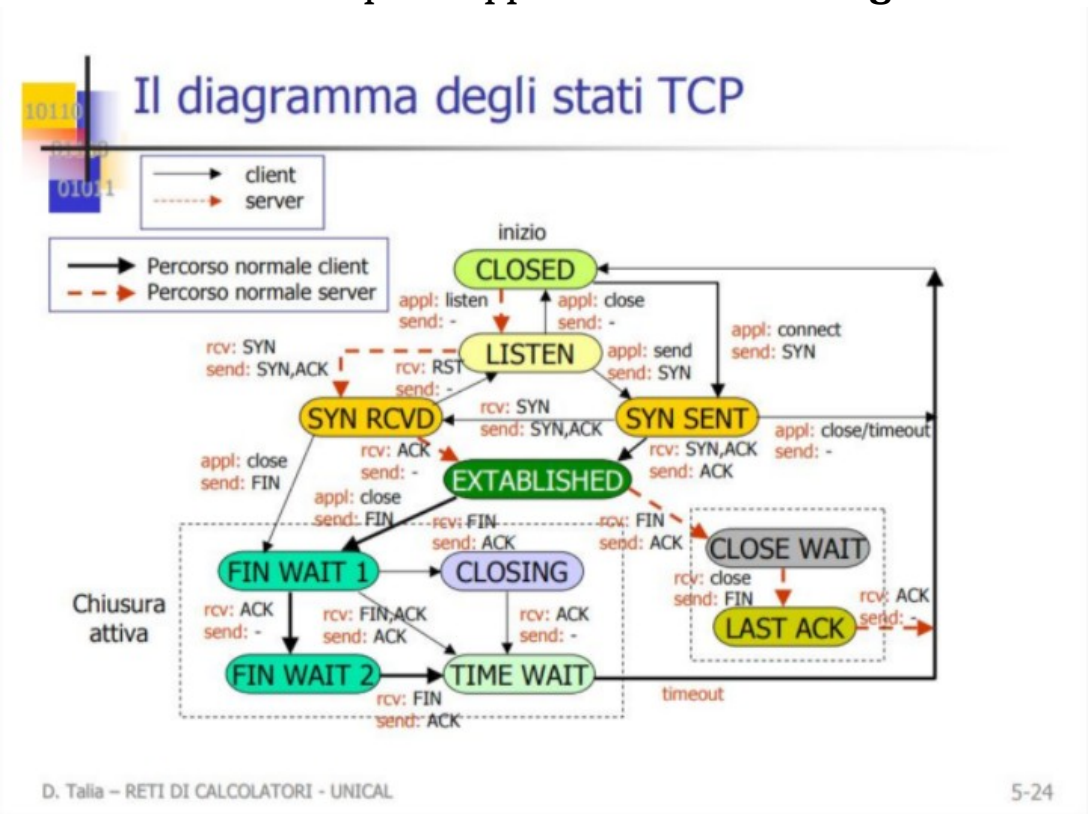
- l'utilizzo del flag **FIN** (al posto del flag SYN) impostato a 1 dall'host 1 per comunicare l'intenzione di chiudere la sessione

- Il server riceve il segmento e deve immediatamente mandare un ACK
- Il client aspetta l'ACK e, quando lo riceve, aspetta un altro segmento dall'host 2 con FIN = 1. Quando lo riceve, manda un ACK all'host 2 che conferma la chiusura della connessione

Se una connessione non può essere rilasciata secondo la procedura normale, TCP prevede una **procedura di reset**: si invia un segmento con il bit **RST** impostato 1, che comporta la chiusura immediata della connessione, senza ulteriori scambi di segmenti.

4.5.5 Diagramma degli stati TCP

Lo stato di una connessione si puo' rappresentare da un **diagramma a stati finiti**.



Closed:

-Stato iniziale, il protocollo non è attivo

-Per uscire da questo stato si deve effettuare un'azione di open (passiva o attiva):

- passiva non manda nulla e passa allo stato di listen
- attiva spedisce un messaggio SYN e passa allo stato SYN SENT

Listen:

In questo stato il protocollo è attivo ed è in ascolto su una porta. Quando riceve un SYN risponde con un SYN+ACK e passa allo stato SYN RCVD.

Se l'applicazione richiede di inviare dati, invia un SYN e passa allo stato SYN SENT.

SYN SENT:

Stato in cui si è mandato il SYN e si attende l'ACK corrispondente:

- raggiunto da closed con un'open attiva
- raggiunto da Listen dopo un'operazione di send

Attende una risposta al SYN per un certo tempo:

- se riceve un SYN con ACK passa allo stato Established e manda a sua volta un ACK
- se riceve un SYN senza ACK (open simultanea) manda un SYN+ACK e passa allo stato SYN RCVD
- se non riceve risposta effettua una close o una reset

SYN RCVD:

Stato in cui si è ricevuto un SYN:

- se lo rifiuta, ritorna allo stato Listen con RST
- se accetta, passa allo stato Established e manda un l'ACK

Established:

Stato in cui si è stabilita la connessione ed è possibile iniziare il trasferimento dati:

- è stata completata la 3-way handshake

- Se l'applicazione decide di chiudere la connessione manda un FIN e passa allo stato di FIN WAIT 1
- Se riceve un FIN risponde con un ACK e passa allo stato Close Wait (close passiva)

Close Wait:

Stato in cui si è ricevuto un messaggio FIN e si attende che l'applicazione chiuda la connessione.

Quando l'applicazione decide di chiudere la connessione manda un messaggio FIN e passa allo stato LAST ACK

LAST ACK:

Stato in cui si è ricevuto il FIN dall'altro end-point e si è risposto con un FIN:

- il protocollo attende l'ACK al suo FIN
- Quando riceve l'ACK risponde con l'ultimo ACK e chiude la connessione

FIN WAIT 1:

Stato in cui si è inviato un messaggio FIN e si attende che l'altro end-point chiuda la connessione.

Se riceve un FIN+ACK manda l'ACK e passa allo stato TIME WAIT.

Se riceve solo un FIN (close simultanea) manda l'ACK e passa allo stato Closing.

Se riceve un ACK passa allo stato FIN WAIT 2.

Closing:

Stato in cui entrami gli end-point hanno mando un FIN contemporaneamente. Manda l'ACK e passa allo stato TIME WAIT.

FIN WAIT 2:

Stato in cui si è inviato un messaggio FIN per il quale è stato ricevuto l'ACK e si attende il FIN dell'altro end-point (half close). Quando riceve un FIN manda l'ACK e passa allo stato TIME WAIT.

TIME WAIT:

Attende un tempo pari a 2*MSL (Maximum Segment Lifetime) prima di chiudere la connessione per attendere eventuali richieste di ritrasmissione dell'ultimo ACK:

- la durata dipende dall'implementazione
- Per tutto questo intervallo di tempo la porta dell'end-point non è riutilizzabile