Abbiamo iniziato a parlare del protocollo TCP e visto la fase del Three way handshake per la realizzazione di un protocollo di comunicazione di tipo Stream. Abbiamo visto i due flag syn/ack e il seq# e l’ack#.

L’RFC 793 del 1981 è il protocollo ancora valido che definisce il funzionamento del protocollo TCP.

L’idea è quella di avere, nell’header, una sequenza di parole di 32 bit. La prima parola è costituita da due sottoinsiemi di 16 bit, che sono i numeri di porta (sorgente e destinazione). Si ricorda che siamo al livello 4 (di trasporto) dunque questo datagramma sarà incapsulato in un altro di livello 3 (che conterrà il protocollo IP con gli indirizzi sorgente e destinazione).

La seconda parola da 32 bit è il sequence number (seq#) e la terza è l’ack number (ack#). Dopodiché, nella quarta parola abbiamo un’altra suddivisione: alcuni bit rappresentano la lunghezza dell’header (una delle caratteristiche del TCP è che l’intestazione ha lunghezza variabile), poi ci sono una serie di bit non usati e infine ci sono i 6 flag (di cui noi abbiamo visto il flag ack e il flag syn).

Dal punto di vista della connessione abbiamo un collegamento CLIENT-SERVER, col Client che è attivo e il Server che è passivo. La connessione è resa affidabile da un meccanismo di temporizzazione e da un invio dei riscontri (dopo ogni invio di informazioni deve essere inviato un acknowledgement in senso contrario, che se non arriva entro una certa quantità di tempo si assume che qualcosa sia andato sorto nella comunicazione e il protocollo prevede che vada inviata un’altra copia del datagramma).

Possiamo cominciare a discutere un aspetto rilevante riguardo le prestazioni del sistema: cosa succede quando vengono persi i messaggi? Avviene la ritrasmissione, che però ha un costo: sia in termini di tempo (poiché la copia arriverà con un certo ritardo che farà perdere tempo), sia in termini di spazio (perché occupiamo il canale di comunicazione: dunque non possiamo inviare altri dati finché non abbiamo ricevuto l’acknowledgement per quelli precedenti -> è necessario l’utilizzo di una banda larga).

È quindi nel nostro interesse cercare di evitare la perdita dei messaggi: ma cosa la può causare? Immaginando che il sender e il reciever si trovano in degli host (computer / non router), questi avranno dei buffer che si possono riempire. Quando i buffer di ricezione del destinatario sono pieni (e questi non ha ancora letto i messaggi nel buffer) se il mittente vuole mandare un ulteriore messaggio (perché per gli altri ha ricevuto l’acknowledgement) quando poi lo fa questo non arriva a destinazione, essendo pieni i buffer di ricezione il messaggio viene cancellato.  
Per evitare questa perdita di informazioni si usa un ulteriore protocollo, che è il protocollo di controllo di flusso (Flow Control). Questo protocollo si basa su ulteriori campi di informazione contenuti nell’header TCP: in particolare fa riferimento ai 16 bit successivi ai flag, che vengono chiamati “finestra di ricezione”. Questi 16 bit ci dicono quanto è grande il buffer di ricezione del destinatario: infatti tra sender e reciever vengono mandati più datagrammi TCP (per via degli handshake, acknowledgement), dunque ad ogni datagramma inviato viene aggiornato il valore nel campo della finestra di ricezione (che si basa sulla quantità di byte disponibili).

Pertanto, il flow control si occupa di tenere conto della dimensione della finestra di ricezione e di far sì che vengano inviati al più (se sono di meno tanto meglio) tanti byte quanto quelli necessari per riempire il buffer e eventualmente aspettare per mandare i byte rimanenti (se ce ne sono). Successivamente, se il reciever manderà un acknowledgement o un altro messaggio che non c’entra, con il datagramma arriverà una finestra di ricezione aggiornata, il cui valore potrebbe quindi non essere più 0 (fino ad allora si assume infatti che sia stata riempita, se abbiamo mandato abbastanza byte di informazioni).  
Questo sistema funziona particolarmente bene quando il flusso di informazioni è bidirezionale, perché così la finestra di comunicazione viene continuamente aggiornata. Se il flusso di comunicazione è monodirezionale, una volta finito lo spazio nel buffer di ricezione bisogna per forza aspettare l’acknowledgement (perché è l’unica risposta che si può ricevere). Ma c’è la possibilità che il ricevente non abbia ancora letto il messaggio quando decide di inviare l’acknowledgement, inserendo quindi nella finestra di ricezione il valore 0, e che svuoti il buffer di ricezione solo in un secondo momento, senza comunicarlo (perché la connessione è monodirezionale, o comunque l’altro non deve inviare messaggi e sta solo in ascolto).

Per questo motivo c’è un’accortezza nel protocollo di controllo di flusso: ogni tanto (se non si ricevono messaggi) si inviano pochi byte (1 solo in realtà di payload, oltre all’intestazione), per limitare i danni in caso di scarto del messaggio e per ricevere un acknowledgement con la finestra di ricezione (recieve window) aggiornata in caso di successo nell’arrivo del messaggio.

--Si ricorda che in tutto ciò i dati sono contenuti all’interno dei buffer di trasmissione e dei buffer di ricezione (finchè non vengono inviati al destinatario i dati rimangono nel buffer di trasmissione, e poi rimangono in quello di ricezione finchè non vengono letti)—

Attraverso il sistema del three way handshake, della ritrasmissione e del controllo di flusso si ha un trasporto efficiente e sicuro delle informazioni, indipendentemente dalla velocità della macchina ricevente.

Oltre ai primi 4 valori da 32 bit dell’header che abbiamo visto, abbiamo quindi i 16 bit di quello che viene chiamato il check sum e il puntatore ai dati urgenti.

Il check sum è un controllo di integrità, serve per il controllo dell’integrità da parte del ricevente (si fanno una serie di calcoli il cui risultato deve essere contenuto nel check sum, se il risultato non combacia allora il messaggio viene scartato, perché è fallito il controllo di integrità: tuttavia questo accade molto, molto, raramente).

Tralasciamo il puntatore ai dati urgenti.

Quello che segue è una parte opzionale (e qui entra in ballo la lunghezza dell’header che è presente prima, perché la presenza o meno della parte opzionale è definita da quella).

Dopodiché c’è il payload.

--Nota: nel protocollo TCP, c’è un sistema per cui prima di inviare l’acknowledgement il ricevente aspetta un po’ di tempo per verificare di non dover inviare dati indietro, in tal caso invia l’acknowledgement e il payload dei nuovi dati assieme, altrimenti finita la quantità di tempo invia l’acknowledgement con payload vuoto.--

Possiamo passare a vedere l’altro protocollo a livello di trasporto, il protocollo UDP.

Il Protocollo UTP è molto più semplice: ha anche lui un header e anche lui contiene le porte sorgente e destinazione. Non ha l’idea di “connessione”, dunque non ci sono i flag, l’ack# il seq#, non c’è la recieve window, non c’è controllo di flusso ecc.

C’è però la funzionalità di multiplexing e demultiplexing dei messaggi (dunque ha le stesse caratteristiche, dal punto di vista dell’affidabilità, del protocollo IP, su cui si basa). Come livello di trasporto è quasi come se non ci fosse.

Passiamo rapidamente a vedere il protocollo di Livello 3 di Rete (Network).

Cominciamo parlando della versione 4 del protocollo IP (c’è anche la versione 6).

Il protocollo di rete assegna gli indirizzi ai vari host e si occupa di effettuare l’instradamento dei messaggi in funzione dell’indirizzo di destinazione. L’header del protocollo IPv4 è costituito in questo modo:

è composto da parole/righe di 32 bit, come descritto dall’RFC 791, (analogamente ai protocolli di livello 4).

I primi bit sono il numero di versione (contengono o il numero 4 per l’IPv4 o il 6 per l’IPv6), poi abbiamo una serie di bit che contengono la lunghezza dell’header (come nel caso del TCP l’intestazione è di lunghezza variabile perché ha una parte opzionale). Dopodiché, abbiamo l’indicazione del tipo di servizio. I successivi 16 bit contengono la lunghezza del datagramma (compreso l’header); la dimensione del datagramma può arrivare fino a 2^16-1 byte. I 16 bit successivi sono un identificatore, poi abbiamo dei flag e poi abbiamo una roba complicata specificata su 13 bit che è legata alla possibilità di frammentare messaggi. Il campo successivo è di 8 bit e viene chiamato “time to live”, poi c’è un altro campo da 8 bit chiamato “next level” protocol e poi abbiamo un check sum (che però si applica solo all’intestazione e non alla parte dati). Poi, Finalmente, abbiamo gli indirizzi: prima c’è quello sorgente e poi quello di destinazione (entrambi di 32 bit).  
Poi c’è la parte facoltativa di ulteriori opzioni.  
Poi c’è il payload (che tipicamente sarà o di tipo TCP o di tipo UDP).

Il next level protocol contiene un codice numerico che indica qual è il protocollo del livello 4: il codice potrà significare o TCP o UDP (leggendo lì si può sapere che protocollo è contenuto nel payload).   
Il check sum è un controllo di integrità, ma può rilevare errori solo per l’header (d’altra parte se il payload è di tipo TCP questo avrà un check sum tutto suo per il controllo di integrità).

Per comprendere Il campo time\_to\_live immaginiamo di avere l’host trasmittente, una serie di routers e poi l’host ricevente: sarà compito dell’algoritmo di instradamento dei singoli router quello di indirizzare il datagramma verso destinazione. Il time\_to\_live è un valore massimo che si impone a priori sulla lunghezza del percorso che può essere realizzato all’interno della rete (in particolare si riferisce al numero di hop che possono essere fatti prima di arrivare a destinazione).  
Quindi se per esempio un primo router trova un TTL di 3, lo riduce di 1, sceglie la direzione e invia il messaggio con un TTL di 2. Quando il secondo router vede che il TTL è ancora maggiore di 0, lo decrementa e inoltra il messaggio. Quando il messaggio arriva al terzo router e lo decrementa, scopre che il valore del TTL è diventato 0 e a questo punto il datagramma non viene più inviato, ma viene scartato.

Si introduce questo meccanismo “limitante” per evitare che a causa di errori nella configurazioni delle tabelle di instradamento il nostro datagramma possa tornare indietro su un router già visitato in precedenza (andando a creare un loop infinito).

Abbiamo un controllo di integrità dell’intestazione del messaggio, questo tiene conto dei valori iniziali del datagramma, quindi quando un router deve decrementare il valore del time to live, per poter mantenere valido il risultato del check sum questo va ricalcolato: ogni volta che si decrementa il TTL il check sum viene ricalcolato dal router in questione e il nuovo valore aggiorna quello precedente.

Approfondiamo il discorso degli indirizzi.

Gli indirizzi IPv4 sono codificati su 32 bit e ogni host deve avere un indirizzo diverso: dunque possiamo avere al massimo 2^32 host diversi connessi alla nostra rete (2^32 = poco più di 4 \* 10^9 = 4 miliardi). Ma se ci sono 8 miliardi di persone nel mondo siamo ben oltre le capacità di indirizzamento del protocollo IPv4. Questo è il motivo principale per cui si è passati dalla versione 4 alla versione 6 (che ha 128 bit per l’indirizzamento -> 2^128 indirizzi possibili). Noi per il momento continueremo a considerare l’IPv4.

Facendo un passo indietro, qual è lo scopo della rete internet? Lo scopo originario era di connettere alla rete le sotto reti locali (quindi avere una rete geografica globale che connettesse tra di loro tante reti locali). Quindi ci siamo sviluppati verso questa implementazione.   
Per tenere conto di questo aspetto l’indirizzo viene diviso in due parti: la prima parte la si può chiamare l’indirizzo di rete e la parte rimanente viene chiamata numero di host. A seconda si come si ripartiscono queste due parti si possono avere più host all’interno delle reti locali. La combinazione dell’indirizzo di rete punta quindi a una LAN particolare, mentre l’host specifico all’interno della rete viene specificato nella seconda parte dell’indirizzo.

All’inizio questo metodo era stato implementato in maniera abbastanza brutale, definendo tre classi di rete (che corrispondevano ad altrettante ripartizioni dei 32 bit di indirizzo). La prima classe era di 8 bit per la rete, 24 per l’host; un’altra classe era 16 bit per la rete e 16 per l’host; la terza classe era di dedicare 24 bit al numero di rete e 8 bit al numero di host (le tre classi erano dette A, B e C).

Gli indirizzi di varia classe venivano quindi distribuiti da un’organizzazione globale.

Con un solo indirizzo di rete di classe A era possibile connettere 16 milioni di computer all’interno della rete locale (e infatti era quella che costava di più). La maggior parte degli indirizzi è stata diffusa come di classe B e C. Poiché la rete internet è nata come progetto di ricerca universitario la maggior parte degli indirizzi di classe A e B sono stati distribuiti alle grosse università americane (che però non li hanno mai saturati, neanche lontanamente).

La maggior parte degli indirizzi di rete distribuiti è stato quindi di classe C, che permettono di avere fino a 256 host.

Quando internet si è diffuso anche in maniera commerciale la divisione in classe è stata rimpiazzata da un’organizzazione che consiste nell’associare ad un indirizzo IP da 32 bit quella che viene chiamata una Netmask, sempre da 32 bit. L’idea di questa Netmask è quella di avere nella parte più a sinistra una certa quantità di bit a 1 e i restanti a 0. La Netmask serve a distinguere la parte indirizzo di rete da quella di numero di host (non c’è più il vincolo a 8, 16, 24 ecc. possiamo metterci anche 30 bit a 1 nella nostra Netmask). In questo modo è possibile assegnare indirizzi in maniera efficiente anche per le reti particolarmente piccole.

Quindi il proprio indirizzo IP assumerà un valore che dipende dai primi x bit dall’agenzia che mi ha fornito l’accesso a internet e gli ultimi 32-x bit da come voglio io (dove x è il numero di bit a 1 nella netmask).

Riassumendo: Una parte dell’indirizzo localizza la rete (LAN) e l’altra un host particolare all’interno della rete, le dimensioni di tali “parti” sono decise dalla netmask. Se tutti i bit di netmask sono al valore 1 la LAN potrà contenere un solo host (con indirizzo locale 0).

Sapendo tutto ciò può risultare abbastanza evidente per quale motivo l’algoritmo di instradamento dei routers non tengano conto di tutti i bit di indirizzo IP, ma solo di una parte iniziale detta “prefisso”: tale prefisso è un sottoinsieme dei 32 bit più significativi che serve a decidere a quale router inviare il messaggio al prossimo hop. Quanti bit formano il prefisso sta a deciderlo il router, secondo l’algoritmo con cui funziona, poiché noi non sappiamo a priori quale sia la netmask utilizzata nella rete locale del ricevente (sappiamo solo che comprenderà i bit più significativi): se la conoscessimo, basterebbe prendere sempre i bit dell’indirizzo che hanno controparte 1 nella netmask della rete del ricevente. Infatti nella fase finale dell’instradamento conta più che altro riuscire ad avvicinarsi alla destinazione, è solo nella parte finale che si deve tener conto di tutti i bit della netmask (quando bisogna prendere la rete tra le poche rimanenti) -> il “prefisso” cresce gradualmente durante gli hop.