Livello di trasporto

Un protocollo di trasporto sfrutta il modello end-to-end. Il termine "end-to-end" si riferisce al fatto che il protocollo di trasporto opera direttamente tra gli host (cioè le estremità) che vogliono comunicare tra loro, senza che nessun altro dispositivo di rete intervenga nella gestione della comunicazione. Il livello di trasporto svolge un ruolo importante nella gestione delle connessioni di rete. Grazie ai suoi meccanismi di controllo del flusso e di controllo della congestione, TCP garantisce una trasmissione affidabile e ordinata dei dati su una rete IP. Tuttavia, in alcune situazioni, come quelle in cui è necessaria una bassa latenza, UDP può essere preferibile per la sua maggiore leggerezza e velocità.

UDP (User Datagram Protocol) — > RFC 768 (request for comment)

UDP è un protocollo di trasporto non orientato alla connessione, per tanto non vi è nessuna procedura di handshake tra gli host comunicanti. In oltre esso è inaffidabile : invia i dati sotto forma di pacchetti noti come datagrammi, che possono essere trasmessi sulla rete senza alcuna garanzia di consegna o di sequenza. Ciò significa che non vi è alcuna garanzia che il destinatario riceva tutti i dati inviati, né che li riceva nell'ordine corretto. Questo protocollo non include nessun meccanismo di controllo congestione, per tanto spedisce pacchetti ad una velocità molto più alta di TCP.

UDP è utilizzato principalmente in applicazioni che richiedono una trasmissione veloce dei dati o applicazioni che richiedono una bassa latenza di rete e che riescono ad ammettere inaffidabilità nella trasmissione di dati (loss-tollerant).

Un pacchetto UDP, anche noto come datagramma UDP, è costituito da un'intestazione (header) e dai dati da trasmettere.

L'intestazione UDP è composta da 4 campi di 2 byte ciascuno, per un totale di 8 byte. I campi sono i seguenti:

- Porta di origine: indica la porta di origine del mittente del pacchetto UDP.
- Porta di destinazione: indica la porta di destinazione del destinatario del pacchetto UDP.
- Lunghezza: indica la lunghezza totale del datagramma UDP, espressa in byte, incluso l'intestazione.
- Checksum: è un valore di controllo di 2 byte calcolato sui dati e sull'intestazione del pacchetto, utilizzato per verificare l'integrità del datagramma.

Checksum

I controlli sulla correttezza di ciò che arriva vengono fatti a livello molto più basso. Si sommano i valori dei campi di intestazione e contenuto dei dati. Non è raro che una macchina ignori il checksum.
Il checksum non assicura che il risultato sia giusto ma serve solo a vedere in che condizioni è sicuramente sbagliato
Tra richiesta e risposta non cambia nulla, vengono solo invertiti i numeri di porta tra sorgente e destinazione.

Per la realizzazione di un canale affidabile è essenziale numerare e mantenere ordinato l'invio dei pacchetti. Si ha a disposizione soltanto un canale inaffidabile che in buona percentuale viene reso affidabile grazie a degli accorgimenti. La situazione viene complicata dal fatto che gli host comunicanti non si vedono .
Come si ottiene un canale affidabile a partire da qualcosa di inaffidabile ?

Sfruttiamo gli automi a stati finiti e lo stato viene definito da alcune "variabili di stato".

RDT(Reliable Data Transfer)

RDT è un protocollo che viene utilizzato per garantire la consegna affidabile dei dati tra due entità di una rete di calcolatori, anche in presenza di errori di trasmissione.
Il protocollo RDT utilizza diverse tecniche per garantire la consegna affidabile dei dati, come l'acknowledgement (ACK) e il retransmission timeout (RTO). In particolare, il mittente invia i dati al destinatario e attende un ACK di conferma. Se l'ACK non arriva entro un certo intervallo di tempo (RTO), il mittente reinvia i dati. In questo modo, il protocollo RDT garantisce che i dati vengano consegnati in modo affidabile, anche in presenza di errori di trasmissione.

RDT 1.0

Trasferimento dati su un canale perfettamente affidabile (banale). Il Sender crea un pacchetto e lo spedisce al canale di comunicazione mentre il Receiver prende un pacchetto dal canale ed estrae i dati. Con un canale così affidabile non è necessario che le due macchine comunichino tra di loro in quanto nulla può andare storto.

RDT 2.0

Si verifica spesso, data l'inaffidabilità, che si perdono i pacchetti e le macchine lo rilevano attraverso un meccanismo di checksum. Quando viene inviato un pacchetto si sfrutta il meccanismo stop-and-wait . La macchina ricevente notifica di aver ricevuto correttamente il pacchetto attraverso un messaggio di ACK(dati ricevuti correttamente), se viene ricevuto con errore si chiede la ritrasmissione con un NAK(dati corrotti). Si necessita di aggiungere uno stato di stop and wait alla macchina Sender. Il Sender spedisce un pacchetto alla volta : se non riceve una notifica di ACK/NAK per quel pacchetto, non proseguirà con l'invio del pacchetto successivo. Se gli errori sono presenti anche sul canale di ritorno bisogna aggiungere un checksum anche sul ACK/NAK e ritrasmettere l'ultimo pacchetto a seguito della ricezione di un ACK/NAK corrotto. Tuttavia questo introduce il problema di pacchetti duplicati, in quanto non è detto che il pacchetto sia stato realmente perduto. L'unica cosa che si può fare è rispedire l'ultimo pacchetto distinguendo pacchetti pari e dispari .

RDT 2.1

La soluzione consiste nell'etichettare i messaggi inviati con un numero di sequenza (0/1) e al Sender sarà sufficiente valutare questo numero per capire se il pacchetto rappresenti una ritrasmissione o meno. Sender e Receiver introducono quindi di un nuovo stato che consentono di valutare il numero di sequenza del pacchetto (basta un solo bit per valutare lo stato 0/1).

RDT 2.2

Ci si rende conto che i NAK non sono necessari. Possono essere eliminati grazie all'invio sempre di segmenti di ACK con l'identificativo dell'ultimo pacchetto ricevuto correttamente. Se il Sender riceve due ACK con lo stesso identificativo (ACK duplicati) allora sa che il Receiver non ha ricevuto correttamente il pacchetto successivo e glielo ritrasmette. Il funzionamento del receiver viene quindi semplificato.

RDT 3.0

Se i pacchetti inviati vengono persi ?
Si necessita di un meccanismo di temporizzazzione (meccanismo di timeout) che risincronizza le operazioni.

RTO (RetransmissionTimeOut) è utilizzato dal protocollo RDT per garantire che i dati inviati dal mittente siano ricevuti dal destinatario in modo affidabile. In RDT, il mittente invia un segmento di dati al destinatario e attende una conferma di ricezione (ACK) dal destinatario. Se il mittente non riceve l'ACK entro un determinato intervallo di tempo RTO, assume che il segmento sia andato perso e lo ritrasmette. Nelle reali implementazioni si lancia un'eccezzione se il pacchetto viene perso un certo numero di volte per evitare di mandare in loop il sistema.

Si può causare un ritardo nell'invio dei segmenti di ACK e l'unica circostanza che fa fallire il protocollo è data dal tempo di viaggio dei pacchetti che non è costante : il sender riceve una risposta di ACK relativa ad un pacchetto vecchio che viene scambiata per il pacchetto attuale , quindi nessuna delle due macchine capisce che c'è stato un errore nella trasmissione.
Si necessita quindi di aumentare i bit usati per l'identificazione del pacchetto (solo un bit per 0/1) per evitare di confondere la risposta ad un pacchetto recente con quella di un pacchetto con lo stesso identificativo ma inviato precedentemente.
Bisogna modificare il timeout : non posso introdurre in meccanismo di temporizzazione costante perchè non tutti i pacchetti hanno la stessa strada da fare, alcuni hanno percorsi molto più brevi di altri.
Un valore troppo basso di RTO può causare una ritrasmissione eccessiva dei segmenti, riducendo le prestazioni della rete. D'altra parte, un valore troppo alto può causare un ritardo nella ritrasmissione dei segmenti persi, aumentando il tempo di consegna dei dati.

Con “troughput” intendiamo la quantità di lavoro svolta nell’unità di tempo. Utilizzando lo stop and wait il tempo totale che impiega il datagramma per viaggiare nella rete, arrivare al destinatario ed essere elaborato sarà dato dalla seguente formula :

$$t = T\_frame + RTT + T\_elaboration + T\_ack$$

- **T_frame** : tempo necessario per trasmettere un frame di dati.
- **RTT** (Round Trip Time) : tempo che impiega un pacchetto per andare dal mittente al destinatario e tornare indietro.
- **T_elaboration** : tempo che impiega il ricevente per elaborare il frame ricevuto, che può includere la decodifica del frame, la verifica degli errori e l'eventuale invio di un messaggio di errore al mittente.
- **T_ack** : il tempo necessario per trasmettere un pacchetto di ACK dal destinatario al mittente per indicare che il frame è stato ricevuto correttamente.

In una comunicazione "stop and wait", il mittente trasmette un singolo frame e poi attende la conferma del destinatario prima di trasmettere il successivo. Pertanto, il tempo totale di trasmissione è la somma di tutti i tempi sopra elencati

RDT 3.0 risulta affidabile ma inefficiente in termini di prestazioni. Il problema risiede nella strategia stop and wait. Il mittente non potrà mai sfruttare tutta la banda disponibile dato che prima di spedire un pacchetto, deve attendere l’ACK del pacchetto precedente, inoltre non va bene in quanto introduce un grosso dispendio di tempo per le distanze lunghe.

Pipelining

Si introduce un meccanismo di Pipelining che prevede che al posto della fase di wait successiva all’invio del primo pacchetto si inviano altri pacchetti in quella finestra di spedizione che tiene un certo numero di pacchetti fino al primo ACK. Si ha un sistema che mi permette di minimizzare il numero di tempi morti, ovvero i momenti in cui il sender rimane in attesta di una risposta senza inviare pacchetti, infatti se la finestra di spedizione è di 3 pacchetti, allora avrò migliorato le prestazioni in quanto 3 pacchetti accodati occuperanno il canale e non attendo che si sia ricevuto il pacchetto per spedire l’altro (avrò triplicato il Troughput). Si può implementare il pipelining attraverso gli approcci Go-Back-N e Ripetizione Selettiva.

GBN (Go Back N)

RDT che sfrutta solo stop and wait va bene solo per le distanze brevi. Go back N prevede una finestra di trasmissione ma anche di ricezione. In caso di perdita di un pacchetto si ritorna indietro dal punto in cui si era interrotta la sincronizzazione , di cui si tiene conto tramite il segmento di ACK cumulativo . Indica che fino al pacchetto n, sono stati ricevuti correttamente tutti i pacchetti precedenti, il re-invio dei pacchetti riguarda quindi tutti quelli da n+1 a seguire. Il problema è che si reinviano anche tutti i pacchetti che seguono il pacchetto perso che erano stati ricevuti correttttamente. Si basa sulla supposizione che i pacchetti successivi al primo rovintato, probabilmente siano rovinati anch’essi.

Ripetizione selettiva

Quando la finestra è grande, nella pipeline si possono trovare numerosi pacchetti. Se un pacchetto viene ricevuto in maniera errata o perso, verranno ritrasmessi anche tutti quelli che seguono, anche se per la maggior parte di questi è stata ricevuta correttamente. Se le perdite sono sporadiche si reinviano solo i pacchetti su cui vi è un sospetto di errore. Infatti il destinatario invia un riscontro per i pacchetti ricevuti fuori sequenza.

TCP / IP v4 (transmission control protocol)

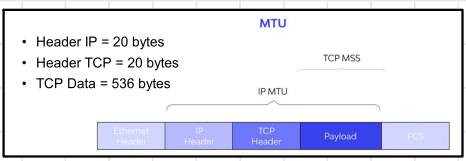
TCP è un protocollo a livello di trasporto che offre alle applicazioni un servizio di comunicazione affidabile e orientato alla connessione che va in esecuzione solo su sistemi periferici, i router intermedi sono ignari della connessione TCP.

TCP offre una connessione di tipo full-duplex in quanto i dati possono fluire da mittente a destinatario e viceversa. Questo protocollo implementa il modello end-to-end, ossia la comunicazione ha luogo esclusivamente fra due host (multicast non ottenibile tramite TCP). TCP frutta l'architettura peer-to-peer in quanto gli host comunicanti sono alla pari (non c'è un dispositivo master e uno slave)

L' MSS (Maximum segment size) definisce la massima quantità di dati inseriti in un segmento. Questo valore sommato all'intestazione TCP (20 byte) deve essere minore dell'unità trasmissiva massima (MTU) ovvero la grandezza massima di un frame che può essere inviato in un collegamento.

Struttura di un segmento TCP

Un pacchetto TCP (Transmission Control Protocol) è composto da un payload (al massimo MSS) e da un header :



- Porta di origine (2 byte): il numero di porta del mittente.
- Porta di destinazione (2 byte): il numero di porta del destinatario.
- Numero di sequenza (4 byte): il numero di sequenza del primo byte di dati contenuti nel pacchetto.
- Numero di conferma (4 byte): il numero di sequenza del prossimo byte di dati attesi dal mittente.
- Lunghezza dell'header (4 bit): la lunghezza dell'header TCP in parole da 4 byte.
- Bit di controllo (6 bit): contiene vari flag di controllo, tra cui il flag SYN, ACK, FIN e altri.
- Window size (2 byte): il numero di byte di dati che il mittente può accettare dal destinatario.
- Checksum (2 byte): un valore di controllo che viene utilizzato per rilevare eventuali errori nel pacchetto.
- Puntatore di urgente (2 byte): se il flag di urgente è impostato, il puntatore di urgente viene utilizzato per indicare il byte di dati successivo che richiede un'attenzione immediata.
- Opzioni (variabile): opzioni aggiuntive, come ad esempio il massimo numero di segmenti di dati che possono essere inviati prima di ricevere una conferma.

Il payload TCP contiene i dati veri e propri che vengono trasmessi dal mittente al destinatario. La dimensione del payload è variabile e dipende dalla quantità di dati da trasmettere.

TCP indica il numero del primo byte del segmento, per esempio se un host deve inviare segmenti di 1000 byte ciascuno , allora il primo ha un numero di sequenza 0, il secondo 1000, il terzo 2000 ecc..

Timeout TCP

TCP usa un timer per far fronte alla possibile perdita di pacchetti. Definiamo RTT il tempo che intercorre tra l'invio di un segmento e la ricezione del suo ACK di verifica. Per calcolare il RTT in TCP, si utilizza spesso la media ponderata **EWMA** (Exponentially Weighted Moving Average). Questa tecnica calcola una media mobile dei tempi di risposta precedenti, assegnando maggior peso ai tempi di risposta più recenti. L'equazione per il calcolo del RTT EWMA è la seguente:

$$RTT = (1 - alpha) * RTT + alpha * SampleRTT$$

- RTT è il valore attuale del RTT calcolato;
- alpha è un fattore di peso (alpha = 0.125) che indica quanto peso assegnare al SampleRTT più recente rispetto ai valori precedenti;
- SampleRTT è il tempo di risposta campionato per il pacchetto più recente.

La DevRTT viene utilizzata per calcolare una stima della variazione del RTT nel tempo, cosa di cui non teneva conto la EWMA. Questa informazione può essere utilizzata da applicazioni di rete per adattare il loro comportamento alle attuali condizioni di rete. La formula per calcolare la deviazione media ponderata EWMA del RTT (DevRTT) è la seguente:

$$DevRTT = (1 - beta) * DevRTT\_precedente + beta * |RTT - Estimated\_RTT|$$

- DevRTT è la deviazione media ponderata EWMA del RTT calcolata in un determinato momento
- DevRTT_precedente è il valore di DevRTT calcolato al momento precedente
- beta (beta = 0.25) è un parametro di smoothing che determina il peso relativo dei valori passati e presenti di RTT nella media mobile.
- RTT è il tempo impiegato per inviare un pacchetto di dati da un computer a un altro e ricevere una risposta (Round-Trip Time)
- Estimated_RTT è il valore di RTT medio calcolato al momento precedente

l tempo massimo di attesa per una risposta, RTO, viene calcolato come la somma dell'Estimated RTT (RTT stimato) e di un valore di DevRTT moltiplicato per un fattore di 4.

$$RTO = Estimated\ RTT + 4 * DevRTT$$

Questo valore di DevRTT moltiplicato per 4 rappresenta un'ampiezza di banda di sicurezza per la finestra di trasmissione. In pratica, il RTO viene impostato in modo che la probabilità di perdere un pacchetto sia inferiore al 1%.

ACK Cumulativi

significa che il destinatario di un flusso di dati TCP conferma la ricezione di tutti i dati fino a un certo punto in una sola volta, invece di confermare la ricezione di ciascun pacchetto singolarmente. L'uso di ACK cumulativi in TCP consente al protocollo di gestire in modo efficiente la ritrasmissione dei dati mancanti, evitando di inviare pacchetti duplicati e migliorando l'affidabilità della trasmissione dei dati.

Fast Restransmitt

Al fine di diminuire i tempi di attesa nella ritrasmissione, TCP metta a disposizione un ulteriore servizio. Se il mittente riceve tre ACK duplicati per un segmento, allora egli considera il segmento che lo segue perduto e quindi lo ritrasmette, ancor prima che il timer per il segmento smarrito, scada. Ad esempio potrebbe accadere che il pacchetto n venga perso ma il destinatario riceva i pacchetti n+1, n+2 ed n+3 prima che scada il timer. Come conseguenza, conserva i pacchetti nel buffer ed invia un ACK duplicato per il pacchetto atteso, n appunto. Quando il mittente TCP riceve i tre ACK duplicati capisce che il pacchetto n si è perso e quindi lo ritrasmette.

Controllo di flusso

TCP offre un servizio di controllo di flusso affinché il mittente non saturi il buffer del ricevente.

NB : non bisogna confondere controllo di flusso con controllo di congestione. La differenza fondamentale è che il controllo del flusso si concentra sul flusso di dati tra due dispositivi di comunicazione, mentre il controllo della congestione si concentra sulla gestione del traffico in una rete di comunicazione più ampia.

Il receiver comunica al sender quale è la capienza di byte che può ricevere e il sender invia nel RcvBuffer dei pacchetti della capienza consentita. Mi serve un meccanismo che verifica se accumulare pacchetti(bufferizzare) ed inivare o se spedire pacchetti poco alla volta.

Algoritmo di Nagle

L'algoritmo di Nagle è un algoritmo utilizzato nei protocolli di trasmissione a livello di rete come il TCP ed è stato progettato per migliorare l'efficienza della rete riducendo il numero di pacchetti trasmessi.
Esso funziona nel seguente modo:

1. Il mittente deve accumulare dati sufficienti prima di inviare un pacchetto. In altre parole, un pacchetto viene inviato solo quando il mittente ha una quantità di dati significativa da trasmettere, invece di inviare un pacchetto ogni volta che ha un byte di dati da inviare.

2. Il mittente invierà un pacchetto immediatamente se il pacchetto precedente è stato confermato dal destinatario.

3. Se il pacchetto precedente non è stato ancora confermato, il mittente attenderà prima di inviare il nuovo pacchetto. Questa attesa viene chiamata "ritardo di Nagle" e dura circa 200 millisecondi.

L'algoritmo di Nagle aiuta a ridurre il traffico sulla rete, poiché riduce il numero di pacchetti inviati, ma può anche aumentare la latenza, in quanto i pacchetti vengono inviati con un certo ritardo.