Si necessita delle parti del protocollo TCP che servono per aprire e chiudere la connessione.
TCP offre un servizio orientato alla connessione : i dispositivi che vogliono comunicare instaurano prima una connessione mediante una procedura di handshake a tre vie con la quale si instaura una connessione TCP tra gli host che devono scambiarsi dei segmenti preliminari per stabilire come avverrà la trasmissione dei dati.

Si aveva inizialmente un handshake a 2 vie ma si ha un problema in quanto la risposta al mittente è standard e non viene data in funzione di quanto ricevuto. Si ha un handshake a tre vie per garantire che il messaggio di risposta sia conseguenza del messaggio di richiesta e poi si invia un messaggio al mittente che risponderà anch'esso in funzione di quanto richiesto.
Nella connessione TCP si ha un numero di sequenza e un ACK . Il numero di sequenza dei pacchetti parte da un numero pseudo-randomico per garantire che la risposta sia in funzione della richiesta : di solito si usano le cifre meno significative della traduzione binaria dell'ora corrente.

Concetto di sfida

Host 1 vuole aprire una connessione con Host 2. E Host 2 risponde. Dato che Host 1 non vede l'Host 2 e Host 2 non vede l'Host 1, ma semplicemente possono scambiarsi il messaggio; chi c'è dall'altra parte? Ci sono messaggi vecchi provenienti da Host 2 che si sono persi? Che spuntano improvvisamente senza motivo? Oppure c'è un reale tentativo di aprire la connessione? Dato che non si vedono, la sfida è "io ti mando un numero, se tu ora sei presente, sei online, allora mi devi rispondere con un valore legato al numero che ti sto mandando". Poi la sfida viene invertita. L'Host 2 sfida l'Host 1 a fare la stessa operazione. Qui non è un problema di sicurezza, è un problema solo per garantire che vecchi messaggi che circolano nella rete per qualunque motivo, che non ci interessa, non vadano ad aprire connessioni, quindi a occupare risorse in maniera inutile. Host 2 apre la connessione solamente se la sua sfida, quella che manda verso Host 1, ha successo.

Apertura connessione TCP

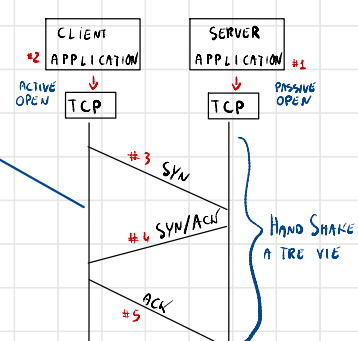
L'apertura di una connessione TCP prevede che il dispositivo client invii un pacchetto di richiesta di connessione al dispositivo server. Questo pacchetto viene chiamato "SYN" (synchronize) e contiene informazioni sulla porta di origine e di destinazione, oltre a un numero di sequenza iniziale. Il dispositivo server, se disponibile e pronto a stabilire una connessione, risponderà con un pacchetto "SYN-ACK" (synchronize-acknowledge) che conferma la disponibilità del server e invia un numero di sequenza iniziale. Il dispositivo client quindi risponderà con un pacchetto "ACK" (acknowledge) che conferma la ricezione del pacchetto SYN-ACK e completa la connessione. Il bit SYN viene posto a 1 per i primi due pacchetti inviati. Il mittente invia il primo pacchetto ed avvia un timer . Non avendo campioni su cui basarmi imposto un timer molto grande per essere sicuro di coprire il tempo di risposta.

Questo approccio ha un problema di sicurezza : vengono bloccate delle risorse che verranno istanziate per tutta la durata del timer. Se si inviano delle richieste continue al server esso si blocca in quanto si satura il buffer di richieste.

D-DOS (Distributed delay of service)

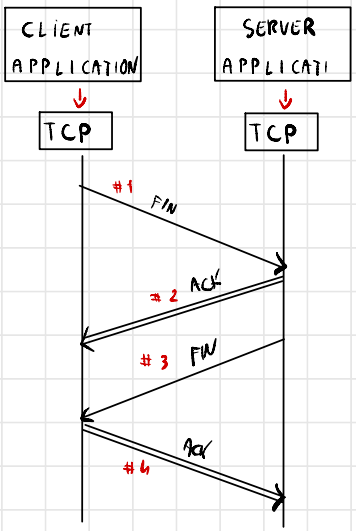
Il Delay of Service DOS è un attacco informatico che mira a rendere un servizio online inaccessibile ai suoi utenti legittimi, aumentando i tempi di risposta del servizio fino a renderlo inutilizzabile. Un approccio "man in the middle" prevede che si inserisca una macchina in mezzo alle due comunicanti che riceva le richieste senza inoltrarle direttamente al server.

SYN : porta origine e destinazione + numero sequenza
SYN/ACK : conferma la disponibilità + numero sequenza sequenza
ACK : completa la connessione



Chiusura di una connessione TCP

La chiusura in TCP è unidirezionale . Se una macchina chiude la propria connessione non può inviare più nulla all'altra ma deve continuare a rispondere ai messaggi con degli ACK, per tanto bisogna che entrambi gli host chiudano la loro connessione TCP. La chiusura di una connessione TCP prevede che uno dei dispositivi invii un pacchetto "FIN" (finish) per indicare che la connessione deve essere chiusa. L'altro dispositivo risponderà con un pacchetto "ACK" per confermare la ricezione del pacchetto FIN. Una volta ricevuto il pacchetto "FIN" di conferma, entrambi i dispositivi completeranno la chiusura della connessione e rilasceranno le risorse associate ad essa. Se un segmento di ACK si perde , l'altra macchina non vede alcuna risposta e chiude il canale di comunicazione.



L'ultimo ACK era necessario? C'è un continuo scambio di pacchetti di ACK dovuto al fatto che si deve confermare la ricezione dell'ACK precedente.

Congestione

La congestione si verifica quando il numero di pacchetti trasmessi attraverso la rete è superiore alla capacità di trasmissione della rete stessa .In un canale i pacchetti viaggiano sempre alla velocità massima. La congestione è creata dai router e non dal canale. Non si può saturare un canale e non è il canale a causare la congestione. Quando la congestione in rete risulta molto alta, è possibile che si verifichino degli overflow nei router causando perdita di pacchetti. La congestione spesso è dovuta alla presenza in rete di copie di copie di pacchetti ritrasmessi perché il collegamento ha generato del ritardo sui router o perché andati persi. Sostanzialmente si arriva ad un situazione in cui, la maggior parte dei pacchetti in rete sono copie che non hanno alcuna utilità ed intasano la rete.

La finestra di congestione TCP si riferisce alla quantità di dati che un mittente può inviare prima di ricevere un'acknowledgement (ACK) dal destinatario.

Fase AIMD

La fase AIMD (Additive Increase Multiplicative Decrease) è un algoritmo di controllo della congestione utilizzato dai protocolli di trasporto come TCP (Transmission Control Protocol) per regolare la velocità di invio dei dati in una rete. Durante la fase AIMD, il protocollo TCP aumenta gradualmente la sua finestra di congestione in modo additivo, ovvero incrementando la quantità di dati inviati di un certo valore a ogni round trip time (RTT), fino a quando non si verifica una perdita di pacchetti o un timeout. A questo punto, la finestra di congestione viene ridotta drasticamente in modo moltiplicativo per alleviare la congestione nella rete. Questo processo viene ripetuto continuamente durante la trasmissione dei dati.

Slow start TCP

Lo Slow Start, d'altra parte, è utilizzato all'avvio della connessione TCP o quando la connessione subisce una riinizializzazione. In questa fase, il protocollo aumenta esponenzialmente il tasso di trasmissione di dati finché non viene rilevata una perdita di pacchetti, a quel punto il protocollo passa alla fase AIMD per regolare il flusso di dati.

La fase AIMD e lo Slow Start sono due algoritmi di controllo della congestione utilizzati in TCP per regolare il flusso di dati sulla rete. La fase AIMD regola il flusso di dati quando la rete è congestionata, mentre lo Slow Start viene utilizzato all'inizio della connessione o in caso di riinizializzazione.

Politica TCP Tahoe (Slow start , AIMD, fast retransmitt)

Congestione -> $\frac{1}{2}$ -> salvo in una Soglia SSTresh
Valore iniziale posto a 1MSS e passo allo slow start fino al raggiungimento del valore prima determinato
Oltre questo valore la crescita sarà lineare fino alla rilevazione di perdita.

Tahoe prevede che ogni qual volta si verifichi un evento perdita (o evento di congestione) di qualsiasi tipo, la finestra di congestione venga dimezzata e questo nuovo valore viene memorizzato nella soglia SSTresh. Fatto questo la trasmissione dei dati ricomincia impostando il valore iniziale della finestra di congestione corrente pari ad 1MSS. Si riinizializza la trasmissione con *Slow Start*, ovvero la crescita della finestra di congestione avviene esponenzialmente fino a raggiungere il valore di soglia prima determinato. Oltre questo valore la crescita avviene linearmente nel tempo secondo la modalità *AIMD*, fino a quando non si verifica nuovamente un evento perdita e si riparte dalla fase *Slow Start*.

È importante sottolineare come la riduzione della finestra di congestione ad 1 MSS comporti una repentina riduzione della velocità di trasmissione dei dati nella connessione TCP. Questo effetto, da un lato, decongestiona la rete, ma, dall'altro, limita temporaneamente (ma fortemente) la velocità di trasmissione/ricezione dei dati. Pertanto, la crescita esponenziale fino al livello di soglia consente alla connessioneTCP di recuperare prontamente (ma parzialmente) una parte della banda ormai persa in seguito all'evento di congestione. Una volta raggiunto il livello di soglia, la crescita avviene lentamente per **sondare** il livello di banda effettivamente disponibile in rete e cercare di raggiungere nuovamente il livello di congestione il più lentamente possibile.

Politica TCP Reno (Slow start , AIMD, fast retransmitt, fast recovery)

TCP Reno reagisce in maniera diversa ai due eventi che segnalano la possibile perdita di un pacchetto dati. L'evento timeout scaduto viene considerato un indicatore "forte" di congestione, e pertanto, TCP Reno, così come TCP Tahoe, fa ripartire la fase di Slow Start da un valore della finestra di congestione pari ad 1 MSS , dimezza la SSTresh e continua in SlowStart. L'evento tre ACK duplicati, invece, viene considerato un indicatore "debole" di congestione, perché sintomatico del fatto che comunque la rete ha consegnato ulteriori pacchetti, dopo un pacchetto probabilmente perso. Pertanto, in TCP Reno questa circostanza produce la ritrasmissione veloce (Fast Retransmit) del segmento che contiene i dati già trasmessi per i quali non è ancora arrivato il riscontro, senza attendere lo scadere del relativo timeout, si riparte con la fase AIMD (incremento lineare) senza riportare a 1MSS la finestra di congestione (Fast Recovery).

1) RTO Scaduto Indicatore forte -> dimezza -> salvo in SSTresh e poi incrementa esponenzialmente, raggiunge il valore e AIMD

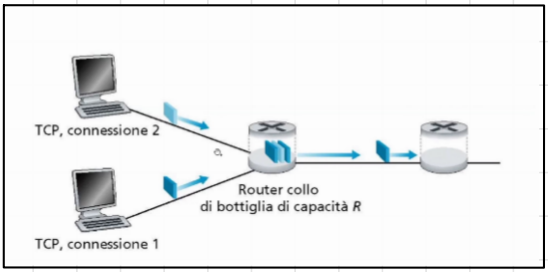
2) ACK Duplicati -> Si è perso un pacchetto, si ritrasmette con Fast Retransmit e si incrementa gradualmente la finestra senza perdita a 1MSS (FAST RECOVERY)

Le due politiche iniziano l'invio dei pacchetti sfruttando lo slow start (aumento esponenziale) e proseguono con la fase AIMD (aumento lineare) se si rileva una perdita tramite triplo ACK duplicato. Se si utilizzasse prima AIMD (incremento lineare del numero di pacchetti trasmessi) e dopo Slow Start (incremento esponenziale) la Fairness non sarebbe garantita (la curva divergerebbe).

Proprietà Fairness

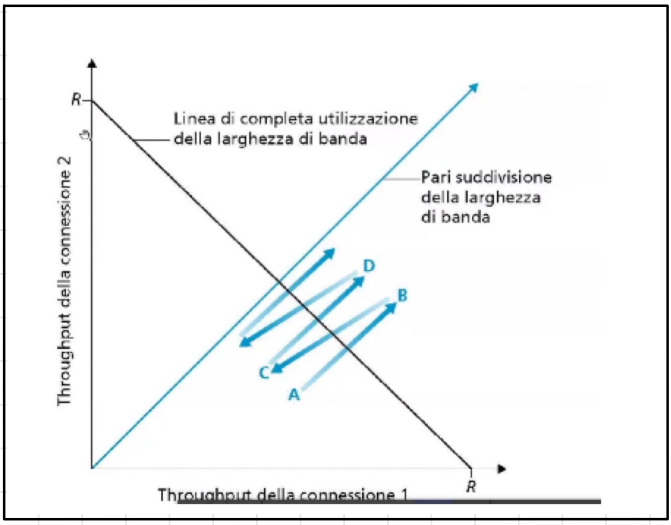
Molteplici flussi di dati possono competere per la larghezza di banda disponibile, e senza una regolamentazione adeguata, un flusso di dati più aggressivo potrebbe saturare la rete impedendo ad altri flussi di ricevere una quota equa di larghezza di banda. Il concetto di fairness si riferisce alla capacità di gestire equamente le richieste di larghezza di banda da parte di più flussi di dati, in modo che nessun flusso abbia un vantaggio ingiusto rispetto agli altri. Ciò significa che TCP deve essere in grado di allocare equamente la banda disponibile tra tutti i flussi di dati che condividono una connessione di rete.

Fairness



Abbiamo due connessioni TCP, un router centrale che fa da collo di bottiglia e ha una certa capacità R.
L'obiettivo finale è che la banda tra queste due connessioni venga equamente divisa in questo link.
Le due connessioni non sanno che l'altra esiste. Il router non può fare niente: il suo unico compito è di inoltrare i pacchetti fuori dal suo buffer con la sua politica FIFO e scartare i pacchetti che arrivano quando il buffer è pieno.

Dimostriamo che la proprietà della fairness esiste con una "dimostrazione grafica":



Andiamo a diagrammare il throughput delle due connessioni. Abbiamo un punto con delle coordinate (x,y) che rappresenta il throughput delle due connessioni.

Dato che R rappresenta la massima larghezza di banda disponibile in uscita, individuiamo altri due punti: R1 = (R,0) e R2 = (0,R), che indicano rispettivamente che la connessione 1 o la connessione 2 hanno preso tutta la banda disponibile. Infine, il punto di coordinate (R/2 , R/2) indica l'equa suddivisione della banda (al centro del grafico).
Tutta la bisettrice infatti è fatta da punti che hanno le coordinate (x,y) uguali, cioè i throughput sono uguali.

La fairness vuol dire che il throughput sta sulla bisettrice, e l'ideale è stare fermi sul punto a R/2.

Il segmento disegnato tra i punti (0,R) e (R,0) ha tutti i punti di coordinate (x,y) tali che x+y = R, perché tracciando una retta che passa per i punti R1 e R2 avrà coordinate:

$y = -x + R \Rightarrow y + x = R$

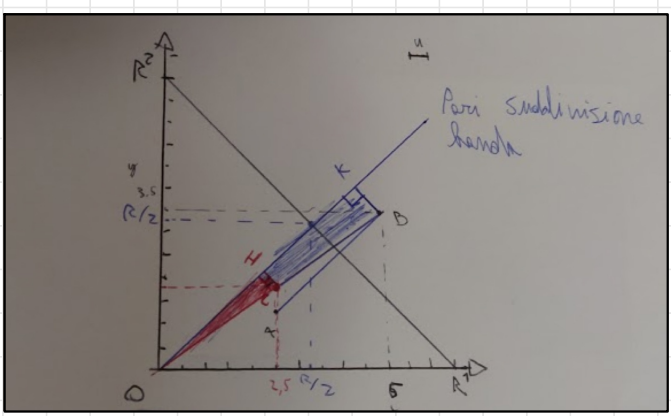
Quindi tutti i punti su questo segmento rispetteranno l'equazione y = -x + R. -1 è quindi il coefficiente angolare di questa retta (che estende il segmento) e una retta con coefficiente -1 è perpendicolare alla bisettrice (che ha coefficiente 1 e equazione y = x ovviamente, senza termine noto perché passa per l'origine).

Quindi è importante dire che il segmento tra i punti R1 e R2 è un segmento e non una curva o un'iperbole.

Tutti i punti che sono nel triangolo rettangolo di vertici OR1R2 sono punti tali che la somma delle coordinate è minore o uguale a R. Quindi il segmento indica la Linea di completa utilizzazione possibile della banda come dice il grafico. Va da sè dire che all'origine le connessioni sono ferme.
Dato che la somma è al di sotto di R il collegamento non è sfruttato al massimo.

Possiamo avere un throughput maggiore di R quindi? Teoricamente no, ma... invece sì. Come?

Consideriamo il fatto che ci sono dei buffer di ingresso che possono accumulare pacchetti, allora può capitare che ci sia un istante in cui il throughput è maggiore di R, semplicemente per il fatto che i pacchetti si stanno accumulando.



Quindi può capitare di arrivare nella zona oltre il segmento R1 R2. Però arrivare in questa zona del grafico, e quindi di avere un punto in questa zona oltre il max throughput, vuol dire che i pacchetti si stanno accumulando e quindi dato che si accumulano le code si riempiono e prima o poi si arriva alla perdita di qualche pacchetto per timeout o triplo ACK duplicato.