**RTO:** retransmission time-out.

**RTT:** round trip time.

**Dupacks:** ack duplicati.

**Controllo del flusso:** se il mittente trasmette più velocemente di quanto il ricevitore possa elaborare, si verifica un data overrun. I protocolli usano un controllo del flusso dei dati per permettere al ricevitore di controllare la velocità della trasmissione dati. I due meccanismi principali sono: stop-and-go e sliding window.

**Stop-and-Go:** il ricevitore invia piccoli pacchetti di controllo quando è pronto per il pacchetto successivo. Il mittente aspetta il pacchetto di controllo prima di inviare il pacchetto successivo. Può essere molto inefficiente se il tempo di invio e ricezione è lungo.

**Sliding Window:** permette al mittente di inviare più pacchetti prima di ricevere un riscontro. Il numero di pacchetti che può essere inviato è definito dal protocollo ed è chiamato window. Quando arrivano dei riscontri (ACK) dal ricevitore, la window si sposta in avanti per permettere il trasferimento di nuovi pacchetti.

**Advertised Window:** misura la memoria libera nel buffer del ricevitore e viene comunicata al trasmettitore.

**Congestion Window:** viene scelta dal trasmettitore in base al livello di congestione che riesce a percepire. Questo livello viene stimato attraverso gli algoritmi di Congestion Control: Slow Start, Congestion Avoidance(Additive Increase Multiplicative Decrease (AIMD)), Fast Retransmit, Fast Recovery.

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo\Protocollo** | **TCP Berkley** | **TCP Tahoe** | **TCP Reno** |
| Slow Start |  |  |  |
| Congestion Avoidance |  |  |  |
| Fast Retransmit |  |  |  |
| Fast Recovery |  |  |  |

**Slow Start**

* Tende ad evitare l’insorgere di congestione durante la fase di avvio di una connessione
* Regola l’emissione dei segmenti all’inizio di una connessione e ha lo scopo di raggiungere il ritmo di emissione a regime senza causare congestione
* Si definisce una Congestion Windows (cwnd) (misurata in segmenti) che tende ad aumentare esponenzialmente fino alla metà della dimensione massima della cwnd, poi aumenta linearmente fino a raggiungere il massimo
* La congestion window limita il valore della finestra fino a che questo non sia fissato dalla ricezione degli ACK

**Congestion Avoidance (AIMD)**

* Regola l’ampiezza della finestra in caso di congestione di rete che non permette di arrivare al valore di finestra indicato dal ricevitore
* Una procedura identica a quella di slow start è troppo aggressiva in caso di congestione
* Il meccanismo di congestion avoidance:
  + È innescato in caso di esaurimento del time-out e quindi di ritrasmissione di un segmento
  + Richiede la definizione di un parametro detto Slow Start Threshold Size (ssthresh)
* Procedura:
  + Lo scopo è quello di aumentare la finestra di un segmento ogni round trip time se non ci sono perdite (Aumento Additivo)
  + Altrimenti se c’è congestione la cwnd viene dimezzata ogni volta che c’è una perdita (Decremento Moltiplicativo)

**Fast Retransmit:** migliora le prestazioni in caso di perdita di un singolo segmento, la cwnd impostata ad 1 (si riparte con la slow start). Viene inviato il pacchetto mancante e l’ack che ritorna si riferisce all’ultimo pacchetto arrivato al reciver (considerando il sequence number).

* Velocizza la ritrasmissione del segmento perso
* Evita la ritrasmissione dei segmenti successivi
* Procedura:
  + Il ricevitore emette un ACK non appena rivela un fuori sequenza ed emette un ACK per ogni segmento successivo fuori sequenza
  + La ricezione di tre ACK duplicati è considerata sintomo di un segmento perso
    - La scelta di tre ACK tende ad evitare il caso in cui il segmento successivo a quello riscontrato abbia subito un ritardo così elevato da aver causato un fuori sequenza
  + La ritrasmissione del segmento inizia non appena si sono ricevuti quattro ACK del segmento precedente anche se il time-out non è scaduto

**Fast Recovery:** evita l’innesco della procedura standard di congestion avoidance associata alla procedura di fast retransmission, si pone ssthresh=cwnd/2 e si continua dopo l’ACK cumulativo con la Additive Increase dal valore della ssthresh.

* L’arrivo di ACK multipli assicura che i segmenti ricevuti sono stati ricevuti e quindi la congestione è stata superata
* Rispetto all procedura standard di congestion avoidance
  + Il valore iniziale di cwnd è maggiore
  + L’incremento di cwnd è sempre lineare
  + Si evita la fase iniziale di aumento esponenziale di cwnd
* La procedura è la seguente:
  + Quando sono stati ricevuti tre ACK duplicati
    - Si pone ssthresh=FlightSize/2  
      (Flight Size: bytes trasmessi ma non ancora confermati)
    - Viene ritrasmesso il segmento perduto (se consentito dal valore della finestra)
    - Per tener conto dei segmenti già ricevuti si pone cwnd=ssthresh+3
  + Ogni volta che arriva un ACK non duplicato, il valore di cwnd viene incrementato di uno e trasmesso (se possibile) un segmento
  + Quando viene ricevuto un ACK duplicato (riscontro cumulativo)
    - Si pone cwnd=ssthresh
    - La finestra è aggiornata come nella procedura di congestion avoidance

**Politiche per il controllo della congestione:** gli eventi che indicano al sender la perdita di dati trasmessi sono:

* ACK non riscontrati allo scadere del timer (time-out)
* ACK duplicati 3 volte

**TCP Tahoe:** il caso dei 3 ACKS duplicati e quello del timeout sono trattati allo stesso modo. La ssthresh viene settata come la metà della cwnd, la cwnd viene reimpostata ad 1 MSS e si riparte con la slow start.

* tale protocollo prevede che ogni qual volta si verifichi un evento perdita (o evento di congestione) di qualsiasi tipo, la finestra di congestione venga dimezzata e il nuovo valore memorizzato nella variabile soglia SSTHRESH
* Fatto questo la trasmissione dei dati ricomincia impostando il valore iniziale della congestion window corrente pari a 1 MSS (massima dimensione di un segmento TCP)
* Si ha quindi la slow start, ovvero la crescita della congestion window avviene progressivamente (seguendo un trend esponenziale nel tempo) fino a raggiungere il valore di soglia prima determinato. Oltre questo valore la crescita avviene linearmente nel tempo (in tal caso si parla di fase di congestion avoidance) fino a quando non si verifica nuovamente un evento perdita e l’algoritmo viene rieseguito
* È importante sottolineare come la riduzione della congestion window ad 1 MSS comporti una repentina riduzione della velocità di trasmissionedei dati nella connessione TCP. Questo effetto, da un lato de-congestiona la rete, ma dall’altro limita temporaneamente (ma fortemente) la velocità di trasmissione/ricezione dei dati.
* La crescita esponenziale fino al livello soglia consente alla connessione TCP di recuperare prontamente (ma parzialmente) una parte della banda ormai persa in seguito all’evento di congestione.

**TCP Reno:** distingue l’evento di perdita dovuto al timeout da quello dovuto dalla ricezione di 3 ACK duplicati.

* In caso di perdita dovuto al timeout del timer, viene applicato l’algoritmo di Tahoe, poiché si assume che la rete sia talmente congestionata da non essere in grado di far passare nessun altro pacchetto, quindi:
  + Si fa ripartire la trasmissione impostando la finestra corrente al valore minimo di 1 MMS
  + Si ricomincia con la partenza lenta con crescita esponenziale
* Quando invece l’evento perdita è generato dalla ricezione di 3 ACK duplicati il TCP Reno assume che la rete è ancora in grado di trasferire qualcosa. In tal caso si entra nella cosiddetta fase di Fast Recovery. Essa consiste in una meno drastica riduzione della congestion window:
  + Il valore soglia viene impostato alla metà del valore della finestra di congestione al momento della ricezione di tre ACK duplicati (cwnd=cwnd/2)
  + La trasmissione riparte impostando il valore della finestra corrente pari al valore di soglia e proseguendo nell’invio con un incremento lineare di 1 MSS ad ogni RTT il valore della finestra di congestione (ssthresh =cwnd, e continua con additive increase dal nuovo valore di ssthresh)

**TCP New Reno**

* TCP Reno risolve il problema di perdite non dovute a congestione solo quando le perdite non sono fortemente correlate tra loro, cioè quando si perde al massimo un pacchetto all’interno di ogni finestra. Questo comportamento è problematico nelle situazione in cui si perdono interi burst di pacchetti (situazione frequente nei collegamenti wireless). Infatti, in questi casi TCP Reno potrebbe ridurre il valore della finestra di congestione più volte consecutivamente (ovvero tante volte quanti sono i pacchetti persi) causando un drastico peggioramento della velocità di trasmissione della connessione TCP
* TCP New Reno cerca di aggirare il problema basandosi sul sistema degli ACK parziali
* Vengono considerati ACK parziali gli ACK che confermano pacchetti intermedi, e non gli ultimi pacchetti che necessiterebbero conferma, dopo che è stata già iniziata la fase di Fast Recovery in seguito all’arrivo dei tre ACK duplicati
* Quando uno di questi ACK si presenta durante la fase di Fast Recovery (cioè in seguito alla ricezione di 3 ACK duplicati), TCP New Reno si mantiene in tale fase continuando a reinviare i pacchetti via via richiesti finchè non viene riscontrato l’ultimo pacchetto inviato nella fase precedente all’ingresso in Fast Recovery

**TCP SACK**: impiega uno schema basato sui cumulative ACK, si informa il mittente su quali segmenti sono giunti a destinazione e quali no (selective acknowledgment (sack)).

* I segmenti ricevuti che non si trovano all’estremo sinistro della finestra di ricezione non sono acknowledged
* Ciò forza il TCP sender a:
  + Attendere un round trip time prima di venire a conoscenza di ciascun pacchetto perduto
  + Oppure a ritrasmettere inutilmente dei segmenti che viceversa sono stati correttamente ricevuti
* Inoltre con lo schema cumulative ACK, perdite multiple di segmenti provocano la perdita della proprietà self-clocking intrinseca al TCP 🡪 diminuzione del throughput
* Vantaggi:
  + Questi comportamenti indesiderati vengono corretti dalla strategia Selective Acknowledgment (SACK), particolarmente efficace in presenza di perdite multiple di segmenti nella singola finestra
  + In cosa consiste
    - Il ricevitore informa il trasmettitore su tutti i segmenti che sono stati consegnati correttamente, così che vengano ritrasmessi solo i segmenti davvero perduti
* Svantaggi:
  + L’header di opzioni di 40byte costituisce un overhead che può risultare pesante in situazioni di bandwidth limitata e se pienamente utilizzato non permette di implementare altri meccanismi per il miglioramento delle performance
  + Non permette di adottare tecniche di compressione degli header, né di distinguere tra i vari tipi di perdita (errore, congestione, handoff, fading)

**TCP Vegas:** parte da un approccio diverso rispetto a TCP Reno e New Reno, cerca di stimare il livello di congestione prima che accada, e quindi di evitarla. Mantiene le regole di equità nei confronti degli altri utilizzatori della rete correggendo in maniera moltiplicativa la frequenza di invio quando si rileva una congestione : applica slow start quando si rileva un timeout.

* TCP Vegas si basa sull’unione di tre tecniche:
  + Meccanismo di ritrasmissione avente lo scopo di reagire più tempestivamente alla perdita di un segmento
  + Meccanismo per anticipare la congestione e aggiustare di conseguenza la frequenza di invio dei dati
  + Procedura di slow start modificata in modo da evitare possibili perdite nella fase iniziale
* Estende il meccanismo di ritrasmissione di TCP Reno tenendo traccia dei tempi di arrivo degli ACK e stimando in questo modo i RTT. All’arrivo di un dupack , se la differenza tra il tempo corrente ed il timestamp del pacchetto risulta maggiore del tempo di timeout, allora il segmento mancante viene immediatamente ritrasmesso senza dover attendere altri dupack. Il tempo che intercorre tra l’invio di un pacchetto e la ricezione dell’ACK viene verificato anche per l’arrivo dei primi due nuovi ACK successivi all’invio dei dati nel caso che si tratti di una ritrasmissione. Se questo intervallo di tempo è più ampio del valore di timeout, allora un altro segmento viene ritrasmesso. Con questo nuovo meccanismo si riesce a ridurre il tempo di reazione del protocollo di trasporto alle perdite di segmenti. Inoltre se si verifica una perdita multipla con una o più invocazioni di Fast Retransmit, la cwnd viene ridotta solo per la prima di queste.
* Il secondo meccanismo implementato da TCP Vegas è quello che gli consente di prevenire le congestioni senza dover aspettare di rilevare delle perdite nei segmenti inviati. L’idea di base è quella di avere la bandwidth pienamente utilizzata e allo stesso tempo tenere sotto controllo la quantità di occupazione dei buffer nei router lungo il percorso dal mittente al ricevente. Infatti buffer troppo pieni provocherebbero perdite di segmenti ma, allo stesso tempo, buffer troppo vuoti non permetterebbero tempestivi adattamenti della frequenza trasmissiva in situazioni di aumenti improvvisi e brevi della bandwidth disponibile.
* Una ulteriore operazione effettuata è quella di calcolare il throughput reale una volta ogni RTT, rilevando il tempo trascorso tra l’invio di un segmento contenente un certo ammontare di dati ed il ricevimento del corrispondente ACK, e dividendo poi il tempo impiegato per i byte trasmessi. Si calcola quindi la differenza tra throughput atteso e quello reale:  
   *diff = Throughput\_Atteso – Throughput\_Reale*Questa differenza, tramite opportuni aggiustamenti lineari della frequenza reale di invio dei segmenti, deve rimanere sempre all’interno di un intervallo prestabilito:  
  α *< Diff <* βGli estremi αe β sono due costanti che devono rappresentare rispettivamente la situazione di avere troppi oppure troppo pochi pacchetti nei buffer dei router lunfo il precorso verso il ricevente.
* L’ultimo meccanismoriguarda alcune modifiche da apportare alla procedura di slow start per renderla in grado di riconoscere una congestione imminente e di evitarla. La crescita della finestra di invio in maniera esponenziale viene effettuata solo una volta per ogni RTT, nel restante periodo rimane fissa. In questo modo il protocollo di trasporto riesce a stimare quanti pacchetti inviare sul canale. Si cerca così di evitare perdite nella fase iniziale di slow start.
* Le versioni, Reno e New Reno del TCP inviano pacchetti sulla connessione con una frequenza sempre più elevata, attendendo di rilevare una perdita per comprendere quando la rete è divenuta congestionata e ridurre così la frequenza di invio. Si può quindi dire che questi protocolli provochino loro stessi delle perdite. TCP Vegas invece utilizza un approccio proattivo che simulazioni hanno dimostrato produrre un throughput anche del 71% migliore rispetto alla versione Reno.
* TCP Vegas non funziona bene in situazioni di asimmetria del percorso: se infatti la capacità del canale dal mittente al ricevente è diversa da quella inversa, non è detto che la misura del RTT costituisca un preciso indicatore del livello di congestione del percorso in avanti.   
  TCP Vegas non è in grado inoltre di reagire adeguatamente a perdite di tipo diverso da quelle dovute a congestione, situazioni nelle quali viene ugualmente invocata la procedura di slow start.

**Snoop Protocols**

[da fare]

By DeNa