1| Spiegare il controllo di congestione nel TCP basandosi sulla seguente figura

Slow start:

1. quando si stabilisce una connessione TCP, il valore cwnd ovvero la finestra di congestione, viene inizializzato 1 MSS (velocità iniziale di invio = MSS/RTT)
2. si incrementa di 1 MSS ogni volta che un segmento trasmesso riceve un ACK , incrementa la finestra di congestione (cwnd) di 1 MSS e invia due segmenti di dimensione massima
3. La velocità cresce esponenzialmente finchè non avviene una congestione, ovvero perdita di pacchetto/ACK, viene salvato il valore della finestra in sshthresh a cwnd/2: metà del valore che aveva la finestra di congestione quando la congestione è stata rilevata. Quindi si entra in modalità

Congestion avoidance

4. Invece di raddoppiare il valore in ogni RTT, TCP incrementa di 1 cwnd di 1 MSS ogni RTT.
5. Ogni volta che riceve ACK l'incremento del mittente della propria cwnd è di MSS*(MSS/cwnd)
6. Quando si verifica una congestione, cwnd viene ancora impostato a 1, però una perdita può essere il risultato della ricezione di tre ACK duplicati, ed entra nello stato di

Fast Recovery

7. Il valore cwnd è incrementato di 1 MSS per ogni ACK duplicato ricevuto relativamente al segmento perso che ha causato l'entrata di TCP in questo stato.
8. Quando arriva ACK del segmento perso, entra in *Congestion avoidance* dopo aver ridotto cwnd
9. Se invece si verifica timeout,si passa a *Slow Start* dopo aver impostato cwnd a 1 MSS e sshtresh a metà del valore cwnd

Slow start e Congestion Avoidance sono componenti obbligatorie in TCP mentre Fast Recovery è suggerita

2 Spiegare il concetto di tunneling tra IPv4 e IPv6

Visto che non tutti i router supportano IPv6, è necessario applicare la tecnica di **tunneling, ovvero un pacchetto IPv6 viene incapsulato come payload di un pacchetto IPv4 attraverso i router IPv4.**

Si mette nel payload di un datagramma IPv4 tutto il datagramma IPv6, così che quando l'host riceve questo datagram IPv4 sa che nel payload c'è un datagramma IPv6.

IPv6 è compatibile con IPv4 ma non viceversa

3 Spiegare i principi di base del protocollo Go-Back-N (GBN) e Selective Repeat (SR)

-Go-Back-N (GBN)

Il mittente può trasferire più pacchetti senza dover attendere l'ACK, ma non può avere più di un numero massimo consentito di N pacchetti in attesa di ACK.

Quindi si crea un finestra di dimensione N su l'intervallo dei numeri in sequenza, questa finestra trasla in avanti ogni volta che viene ricevuto l'ACK del pacchetto con N più basso.

Ogni volta che c'è spazio sulla finestra viene creato e inviato un pacchetto, nel caso sia piena lo restituisce al livello superiore

Quando viene ricevuto un ACK, il numero di sequenza di esso significa che tutti i pacchetti con un N inferiore ad esso sono stati ricevuti correttamente.
Si usa un timer per ritrasmettere un pacchetto nel caso in cui non venga ricevuto un ACK in un determinato tempo.
Nel caso in cui il destinatario riceve un pacchetto non in ordine, lo scarta e manda un ACK con il numero di sequenza dell'ultimo pacchetto consegnato con l'ordine giusto.

-Selective Repeat (SR)
Visto che ritrasmettendo pacchetti che sono arrivati correttamente al destinatario solamente perchè non è arrivato uno con un numero di sequenza inferiore ad essi, questo protocollo fa in modo che avvengano ritrasmissioni inutili ritrasmettendo solo i pacchetti che vengono persi o su cui esiste errore. Quindi **il destinatario deve mandare ACk specifici in modo che il mittente capisca quali pacchetti sono da ritrasmettere**
Quindi in questo caso il destinatario dovrà utilizzare anch'esso un buffer dove immagazzinare i pacchetti ricevuti correttamente e nel caso in cui riceva un pacchetto presente nel buffer venga scartato.

Quindi la principale differenza tra essi è che Go-Back-N fa in modo che se viene trasmesso per primo il pacchetto con numero di sequenza N ma viene ricevuto prima quello con numero di sequenza N+1, viene inviato un ACK con il numero di sequenza dell'ultimo pacchetto ricevuto nel corretto ordine. Quindi vengono ritrasmessi tutti i pacchetti dopo di esso.
Mentre in Selective Repeat, se viene consegnato prima il pacchetto con numero di sequenza N+1, NON viene scartato ma viene inviato un ACK con il suo numero di sequenza al mittente, esso capisce quali pacchetti sono andati persi e li ritrasmette, il destinatario aspetta a spostare la finestra finché non ha i pacchetti in ordine di numero di sequenza

4 A cosa serve la frazione La/R, dove L è la lunghezza dei pacchetti in bit, a è la velocità media di arrivo dei pacchetti, e R è la velocità di trasmissione.
Discutere i possibili scenari in base al valore della frazione
Il ritardo di accodamento La/R è la grandezza che incide maggiormente nel ritardo di un pacchetto, è una grandezza variabile in funzione della velocità di arrivo sulla coda, della velocità di trasmissione del collegamento e della natura del traffico per caratterizzarla si fa uso di misure statistiche quali il ritardo di accodamento medio, la varianza del ritardo di accodamento e la probabilità che il ritardo di accodamento supero un valore fissato

- **La/R>1** allora la velocità media di arrivo dei bit supera la velocità alla quale vengono ritrasmessi in uscita; In questa situazione la coda tenderà a crescere all'infinito e con essa il ritardo di accodamento
- **La/R<=1** la velocità in uscite è più alta di quella in entrata e sul ritardo di accodamento influisce la natura del traffico. Se l'intensità di traffico è vicina a zero, allora gli arrivi dei pacchetti sono pochi e piuttosto distanziati mentre se è vicino ad 1 il tasso di arrivo dei pacchetti è molto alto e si forma facilmente una coda

5 Commentare il codice

```
rset=allset
[0,1,0,0,1]
```

Ad ogni tabella corrisponde un socket descriptor
Voglio vedere se c'è attività nel socket descriptor 1 e 4.
Per vedere se più socket descriptor hanno avuto attivita si usa una system call select.
Deve sostanzialmente ascoltare i 2 socket descriptor.

Si ha in input un rset che è il nostro array di bit la select può ascoltare in fase di lettura tramite DHP capire se ci sono stati errori oppure ci sono dei socket pronti a scrivere, noi abbiamo utilizzato quella più semplice che è quella della lettura, se questo socket descriptor (quello listen) è pronto in lettura, vuol dire che qualcuno ha fatto una connect

Se invece c'è qualcosa di pronto in lettura nel socket del client vuol dire che il client ci ha parlato per esempio ci ha chiesto di calcolare il massimo, gli rset che vengono passati alla select vengono passati per riferimento infatti c'è la & il che vuol dire che se la passo alla select e la select scatta può succedere che l'array può passare a una situazione del genere [0,0,0,1] perché viene passata da riferimento e la select l'ha modificato, se l'ha modificato così vuol dire che io ero in ascolto su 2 socket descriptor ma solo uno è diventato pronto in modalità ascolto

if(ready=select(maxd+1,&rset,NULL,NULL,NULL)<0)

tramite la FD-ISSET guardo quali di questi bit ha avuto attività

if(FD_ISSET(sockfd,&rset))

Dopo di che io devo rimettermi in ascolto sull'rset quindi devo rifare la select il problema è che se noi non abbiamo memorizzato la struttura iniziale rset cioè salvare i bit in cui dovevamo monitorare il socket descriptor su ci abbiamo avuto la prima attività

Quindi cosa facciamo ?

Dobbiamo memorizzare da qualche parte questa struttura dati che abbiamo chiamato allset dove all sta per dire tutti quelli che io voglio controllare nell'attività

Qua addirittura perderemmo la listen socket perchè se viene monitorata attività sul socket descriptor del client perdo quello della listen e quindi non farò più nulla se qualcun'altro mi interpella

Quando c'è attività sulla listen socket io faccio un fdset per aggiornare il valore del socket desc descriptor.

In allset ho tutti i bit da monitorare e rset è la vittima in questo caso.

6 Che differenza c’è tra listen socket e connection socket che si gestiscono in una classica comunicazione client/server ?