Principi del trasferimento dati affidabile - 2

domenica 13 agosto 2023 16:45

Il protocollo rdt 3.0 ha un problema di prestazioni: **è un protocollo stop-and-wait**, ovvero il mittente manda il pacchetto ed aspetta che questo arrivi al destinatario, quest'ultimo una volta ricevuto, manda l'ACK al mittente di conferma, solo da quel momento il mittente manda un secondo pacchetto.

Calcoliamo praticamente le sue prestazioni

Dati:

RTT (ritardo di propagazione) totale: 30 ms (15ms andata, 15 ms ritorno);
Due host connessi ad una **velocità di trasmissione** pari a 1Gbps (10^9 bits/sec);
Pacchetti di dimensione L = 1000 byte (8000 bits).

Premessa:

Il tempo richiesto per trasmettere tutti i bit del pacchetto sul collegamento è $\frac{L}{R}$ quindi:

$$dt = \frac{L}{R} = \frac{8000bits}{10^9\ bits/s} = 8\ microsecondi$$

Il pacchetto effettua un viaggio dall'host A all'host B con un ritardo di propagazione di andata pari a 15 millisecondi, quindi devo aggiungere a questi 8 microsecondi un ritardo di propagazione di 15 ms, di conseguenza l'ultimo bit giunge al destinatario all'istante t = 15ms + 0.008 millisecondi = 15,008 ms.
Assumiamo che i pacchetti ACK siano molto piccoli quindi trascurabili; il destinatario può spedire un ACK non appena ricevuto l'ultimo bit di un pacchetto dati, abbiamo quindi che l'ACK giunge al mittente all'istante t = RTT (ritardo di propagazione andata + ritardo di propagazione ritorno) + $\frac{L}{R}$ = 30,008 ms.
Solo a questo punto dopo aver ricevuto l'ACK il mittente può mandare il messaggio successivo.
Definiamo l'utilizzo (**utilization**) del mittente inteso come frazione di tempo cui lui è occupato con il pacchetto, viene restituita la percentuale di tempo che utilizziamo nella rete rispetto alle sue potenzialità.
Il protocollo stop-and-wait presenta un utilizzo del mittente pari a:

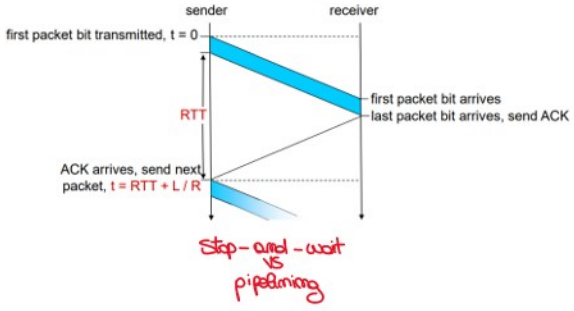
$$U_{mittente} = \frac{\frac{L}{R}}{RTT+\frac{L}{R}} = \frac{0,008}{30,008} = 0,00027$$

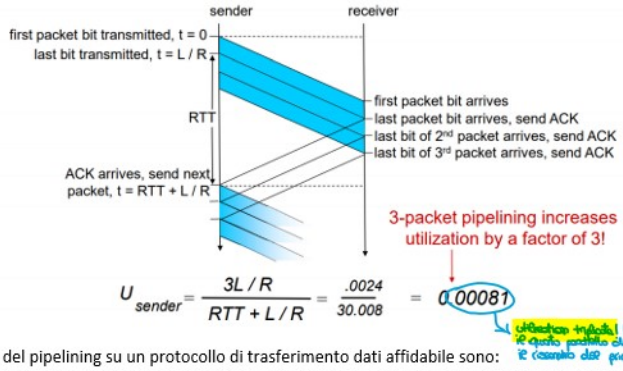
L'utente è attivo per 2,7 centesimi del 1% del tempo, quando avrebbe potuto inviare altri pacchetti.
Lo stop-and-wait va bene per le reti in cui il ritardo è molto piccolo, ad esempio nel wi-fi, dove i ritardi sono molto piccoli.

Introduciamo un nuova versione del protocollo rdt 3.0..

Protocollo rdt 3.0 con pipelining

Per aumentare l'**utilization** possiamo consentire al mittente di inviare più pacchetti senza dover attendere per ognuno di essi i corrispettivi ACK..





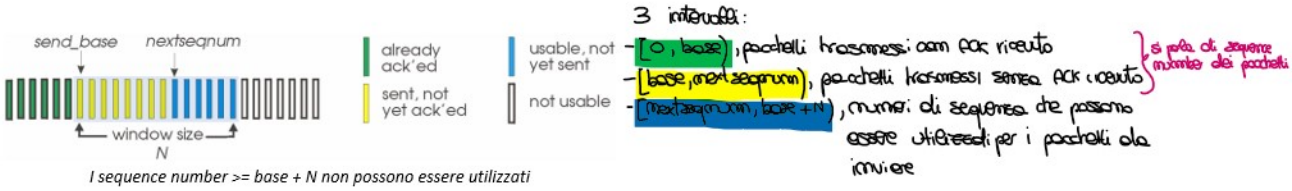
Le conseguenze dell'uso del pipelining su un protocollo di trasferimento dati affidabile sono:

- L'intervallo dei numeri di sequenza disponibili deve essere incrementato, questo perchè ci possono essere più pacchetti in transito in attesa di ACK e ogni pacchetto, in transito, deve presentare un sequence number univoco (per poter essere identificato);
- Il lato mittente e il lato ricevente possono dover memorizzare in un buffer più di un pacchetto: il mittente dovrà memorizzare i pacchetti trasmessi ma il cui ack non è ancora stato ricevuto.

La quantità di numeri di sequenza necessari e i requisiti di buffer dipendono dal modo in cui il protocollo di trasferimento dati reagisce ai pacchetti smarriti, alterati o troppo in ritardo. Per la risoluzione degli errori con pipeline abbiamo due approcci: **Go-Back-N** e **Selective Repeat**.

Go-Back-N

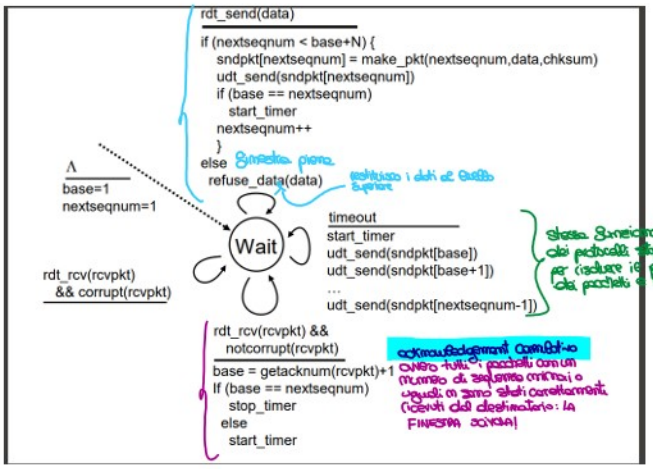
Non possono esserci più di N pacchetti consentiti in attesa di ACK. Definiamo ora "**base**" il sequence number del pacchetto che aspetta da più tempo un ACK e "**nextseqnum**" il più piccolo sequence number inutilizzato, *ovvero il numero di sequenza che avrà il prossimo pacchetto da inviare...*



I sequence number >= base + N non possono essere utilizzati

Quando il protocollo è in funzione, la finestra trasla lungo lo spazio dei numeri di sequenza. Per questo motivo GBN è detto **protocollo a finestra scorrevole.**
Nel protocollo GBN, se un pacchetto viene perso, grazie al timeout, io me ne accorgo, appena scatta il pacchetto perso e tutti gli altri senza risposta ACK vengono ritrasmessi.

Lato mittente

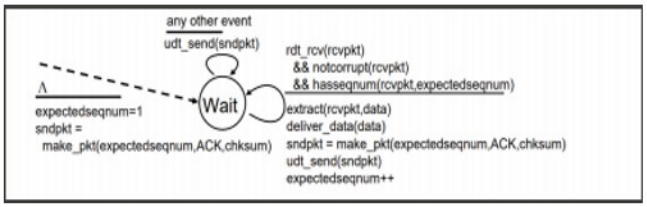


Il mittente GBN individua 3 tipi di eventi:

1. Ev. Invocazione dal livello superiore;
2. Ev. Ricezione di un ACK;
3. Ev. Timeout.

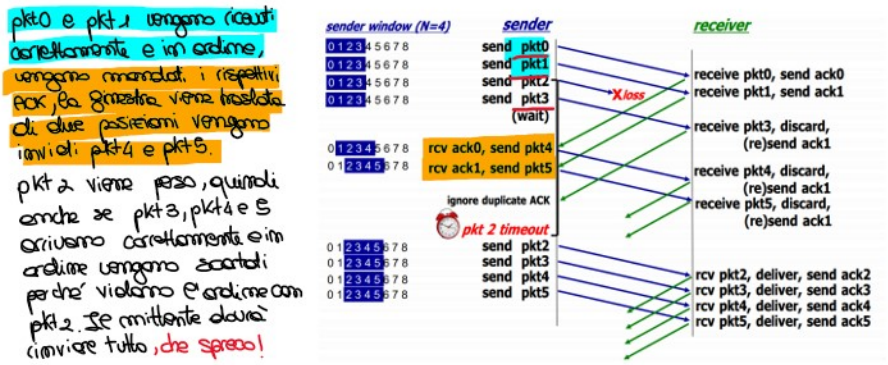
… Quando si verifica un timeout (tempo scaduto), il mittente invia nuovamente **TUTTI** i pacchetti spediti che ancora non hanno ricevuto un ACK.

Lato destinatario



Il destinatario GBN se gli arriva un pacchetto con un numero di sequenza ordinato e correttamente, manda l'ACK corrispondente di risposta, in tutti gli altri casi il pacchetto viene scartato e viene mandato l'ACK dell'ultimo pacchetto non scartato ricevuto.
La scelta di non memorizzare i pacchetti ricevuti fuori sequenza in un buffer, per poi consegnarli correttamente in ordine al livello superiore, è per pura semplicità.

Il mittente deve mantenere il limite superiore e inferiore della finestra e la posizione di **nextseqnum** all'interno di tale finestra (--> dove siamo arrivati), il destinatario invece deve solo memorizzare il numero di sequenza del successivo pacchetto (**expectedseqnum**) nell'ordine che gli deve arrivare (--> che gli serve per mantenere l'ordine dei pacchetti che gli servono).
In funzione…
Abbiamo una finestra di 4 pacchetti; il mittente può inviare quindi da 0 a 3 pacchetti, poi deve attendere l'ACK di uno di questi per inoltrare il successivo pacchetto…

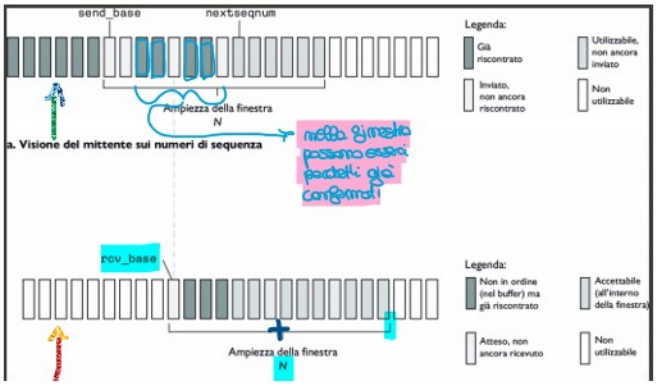


Si nota che GBN ha problemi di prestazioni quando l'ampiezza della finestra e il prodotto tra larghezza di banda e ritardo sono grandi! In questo caso nella pipeline si possono trovare tanti pacchetti, ne consegue che un errore (come la perdita di un pacchetto) su un singolo pacchetto comporta un numero elevato di ritrasmissioni; lo spreco di GBN è che i pacchetti fuori sequenza vengono scartati dal ricevente.

Selective repeat

I protocolli a ripetizione selettiva evitano le ritrasmissioni non necessarie, ovvero il mittente SR ritrasmetterà solo i pacchetti persi o alterati, quindi ne consegue che il destinatario SR invia un ACK sia per i pacchetti in ordine che per quelli fuori sequenza.
Infatti, questi pacchetti fuori sequenza vengono memorizzati in un buffer fino a quando non sono stati ricevuti tutti i pacchetti precedenti mancanti. Quando quest'ultimi sono stati ricevuti viene passato il blocco al livello superiore.

Finestra per il protocollo SR, di N elementi per evitare il numero di pacchetti privi di ACK nella pipeline



Nota: ***nextseqnum*** *è il puntatore del pacchetto con sequence number più basso che si può spedire.*
Nel protocollo SR i riscontri NON sono cumulativi, a differenza di GBN.

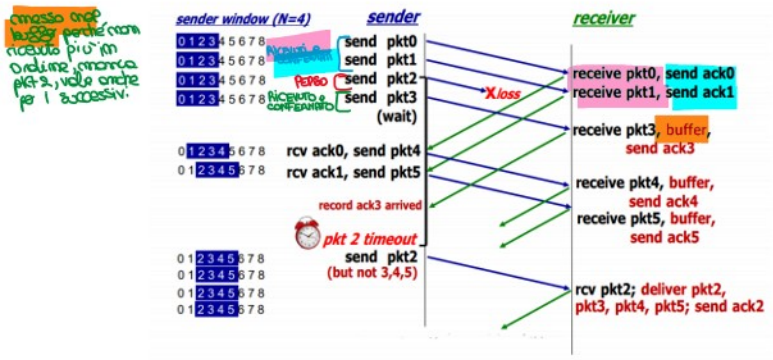
b. Visione del ricevente sui numeri di sequenza

I 3 eventi mittente:

1. **Ev. Dati ricevuti dall'alto** (si intende livello applicativo),il mittente controlla il successivo numero di sequenza disponibile per il pacchetto. Abbiamo due casi:
    i. Se il numero di sequenza è all'interno della finestra i dati vengono impacchettati e inviati;
    ii. Se il numero di sequenza è al di fuori della finestra i dati vengono salvati nei buffer o restituiti al livello superiore per una successiva ritrasmissione (proprio come accadeva in GBN);
2. **Ev. Timeout**, utilizziamo anche qua dei contatori per gestire la perdita di un pacchetto o di un ACK, in aggiunta ora ogni pacchetto deve avere il proprio timer logico, così che al timeout del pacchetto specifico sarà ritrasmesso solo quello;
3. **Ev. ACK ricevuto**, il mittente etichetta tale pacchetto come ricevuto, ammesso che sia nella finestra. Se il numero di sequenza del pacchetto è uguale a **send_base**, allora la base della finestra si muove verso il pacchetto che non ha ricevuto ACK con il più piccolo numero di sequenza.

I 3 eventi destinatario:

1. **Ev. Ricezione pacchetto con numero di sequenza nell'intervallo [rcv_base, rcv_base + (N - 1)]**, il pacchetto viene ricevuto correttamente e ricade nella finestra del ricevente, al mittente viene quindi restituito un **ACK selettivo**, di un dato pacchetto; se quest'ultimo non era già stato ricevuto viene inserito in ordine nel buffer;
2. **Ev. Ricezione pacchetto con un numero di sequenza nell'intervallo [rcv_base - N, rcv_base - 1]**, ovvero dall'inizio dei sequence number fino al numero di sequenza che precede il primo presente nella finestra; si tratta di un pacchetto già ricevuto, in questo caso viene comunque generato un ACK corrispondente: non ignoro i pacchetti già ricevuti come avveniva in precedenza ma viene ri-inviato nuovamente l'ACK. Questo avviene perché il mittente non ha ricevuto l'ACK precedente e non ha slittato la finestra al sequence number successivo, finchè non riceve l'ACK corrispondente lui continuerà ad inviare quel pacchetto senza avanzare;
3. Se non ricade nei casi precedenti il pacchetto viene ignorato.



Nota: dopo aver ricevuto l'ACK di 0 (rcv 0) io faccio traslare la finestra, perchè 0 era il più piccolo pacchetto di cui stavo aspettando il riscontro, una volta ricevuto traslo verso 1.
Supponiamo di avere un intervallo di 4 numeri di sequenza: 0, 1, 2, 3. *Ricorda che i numeri di sequenza assegnati sono ciclici, infatti, dopo che è stato assegnato il pacchetto il numero di sequenza 3, il prossimo sarà 0.*
Supponiamo inoltre di avere una ampiezza di finestra pari a 3 e che i pacchetti 0, 1, 2 vengano trasmessi e ricevuti correttamente e che il destinatario invia gli ACK corrispondenti per ognuno.
A questo punto la finestra del destinatario si sposta sul quarto, quinto e sesto pacchetto, i quali presentano rispettivamente i numeri di sequenza 3, 0 e 1. Abbiamo due scenari:

A. **Gli ACK dei primi 3 pacchetti vanno persi**, la finestra del destinatario si muove ma quella del mittente sta ferma. Il destinatario riceve un pacchetto con sequence number 0, lui pensa che contiene nuovi dati invece è un pacchetto ritrasmesso;
B. **Gli ACK dei primi 3 pacchetti vengono ricevuti correttamente**, le due finestre traslano e il mittente invia il pacchetto con sequence number 0 successivo che contiene nuovi dati.

Dal punto di vista del ricevente *le diverse azioni del mittente non vengono distinte*, quindi lui non sa se il pacchetto con sequence number 0 ricevuto per la seconda volta contiene nuovi dati oppure è un pacchetto ritrasmesso.
Di conseguenza un'ampiezza di finestra inferiore di 1 rispetto a quello dello spazio dei sequence number non funziona.