RDT - Continuo

Problemi dell'RDT

Questo protocollo presenta però dei problemi: cosa accade se il risconto ricevuto è anch'esso corrotto? Potremmo avere quindi un'alterazione dei bit dei pacchetti contenenti i **riscontri** che viaggiano dal ricevente al mittente. Cosa facciamo nel caso in cui si presenta un'anomalia?

Tenendo presente che può esserci una violazione anche sui riscontri, dobbiamo prevedere una checksum anche sui riscontri.

Nel pacchetto contenente il riscontro deve quindi essere presente una checksum, per consentire al mittente (a chi riceve il riscontro) di verificare se il riscontro ricevuto è corretto oppure no.

Se il riscontro non è corretto, il mittente chiede la rispedizione del pacchetto. Nel caso in cui il riscontro è ricevuto in maniera alterata, il mittente assume che il ricevente abbia inviato un riscontro negativo, quindi calcoliamo il caso peggiore.

La rispedizione del pacchetto, però, non è così semplice da gestire: è possibile che il pacchetto fosse stato spedito correttamente, ma siccome il riscontro (e non il pacchetto) è stato alterato durante il transito in rete, potrebbe crearsi un duplicato; questo perchè il ricevente (che dava per scontato che il pacchetto fosse stato inviato correttamente) interpreta il pacchetto come se fosse un nuovo pacchetto, e da una nuova risposta.

Per capire se il pacchetto è un duplicato o meno, è il mittente che deve inserire un'informazione che consente al ricevente che si tratta di un duplicato.

Gestione dei duplicati

Il problema appena introdotto, ovvero che si alteri un riscontro, e la successiva risoluzione attravero la rispedizione del pacchetto, richiede che nel pacchetto sia aggiunto un **numero di sequenza**. Questo numero di sequenza ci permette di capire se il pacchetto è un nuovo pacchetto, o un pacchetto già spedito in precedenza.

Un pacchetto viene flaggato come duplicato nel momento in cui il suo numero di sequenza è identico ad un numero di sequenza di un pacchetto già spedito in precedenza; ci aspettiamo quindi che i pacchetti spediti in successione abbiano dei numeri sequenza diversi.

Quando siamo nello stato inziale (alto a sinistra) e sono disponibili dei dati da spedire, viene costruito un pacchetto e viene inserito il numero di sequenza che corrisponde a quello stato; successivamente il pacchetto viene spedito.

Nel nuovo stato (stato di attesa per un riscontro positivo/negativo legato alla spedizione del pacchetto) si può passare al secondo stato (basso a dx).

In questo stato (basso a sx) possiamo poi ricevere altri dati, costruire un pacchetto contenente il numero di sequenza 1 e passare ad un riscontro:

Se arriva il riscontro ed è integro (ACK), si passa ad uno stato associato al numero di sequenza zero, quindi è possibile riutilizzare il numero di sequenza zero. Poichè dobbiamo capire se nei pacchetti in successione ricevuti sono pacchetti diversi, o lo stesso pacchetto, sono sufficienti solo due numeri di sequenza: 0 ed 1.

La transizione che si richiude sullo stesso stato di riscontro, e quindi è una transizione che viene realizzata nel momento in cui viene ricevuto qualcosa diverso da un ACK, abbiamo diverse possibilità: se il pacchetto ricevuto è alterato o un NACK, l'azione corrispondente è la rispedizione del pacchetto:

Il pacchetto che viene spedito in questo caso è diverso dal pacchetto che viene spedito nel seguente caso:

che è la medesima azione legata alla transizione che si richiude sullo stato di attesa di un riscontro per la spedizione precedente di un pacchetto contenente il numero di sequenza 1.

NAK-free protocol

Possiamo pensare di non considerare in maniera esplicita il riscontro negativo, ovvero rimuovere i riscontri negativi; non prevediamo i riscontri negativi ma possiamo usare un approccio diverso:

Possiamo considerare un ACK con numero di sequenza diverso da quello attesa; questa variante dell'RDT2.2 (protocollo che consente di superare i problemi legati all'integrità dei pacchetti), invece di usare un NACK costruiamo un riscontro con un numero di sequenza diverso rispetto a quello che deve essere previsto per questo stato.

🏁 0:20

RDT 3.0 - Canali con errori e perdite

🏁 00:30

Finora abbiamo considerato soluzioni al problema del canale che può alterare i bit di un pacchetto in transito, ma cosa accade se il canale **cancella** un pacchetto durante la trasmissione?

Ad esempio una componente intermedia, ad esempio un router, scarta il pacchetto perchè non ha spazio. **Cosa facciamo?**

Domanda: come gestiscono delle parole perse in una conversazione telefonica sender-to-receiver gli umani? Effettivamente ci accorgiamo che il ricevente non ha ricevuto nulla perchè non otteniamo alcuna risposta.

L'approccio: possiamo considerare un'attesa temporizzata; chi spedisce si aspetta di ricevere **per ciascun pacchetto che ha spedito** un riscontro. Il mittente, quindi, aspetta per un tempo ragionevole in attesa di un ACK. Se questo non viene ricevuto nell'intervallo temporale, il mittente ipotizza che il ricevente non ha ricevuto il pacchetto.

Se l'attesa da parte del mittente non è corretto (stimiamo un intervallo temporale piccolo), il riscontro potrebbe si arrivare, ma dopo l'intervallo stabilito dal mittente. Di conseguenza, il mittente, pensando che il ricevente non ha ricevuto il pacchetto, rispedisce il pacchetto precedente (con un approccio conservativo) ma se il riscontro è stato semplicemente ritardato (o l'intervallo di attesa è stato sottostimato) il pacchetto sarà un **duplicato**.

Questo non è un gran problema, proprio perchè in ricezione il controllo sui **numeri di sequenza** ci permettono di capire se il pacchetto è un duplicato oppure no.

Ad ogni modo, in questa variante nella quale consideriamo un canale che può perdere pacchetti, aggiungiamo un ulteriore elemento: il **timer**.
Ogni pacchetto che viene spedito fa partire un timer, ed associato ad un timer è presente un intervallo allo scadere del quale si verifica un **nuovo evento: timeout**.

Cosa accade nell'automa?

1. Quando il pacchetto viene spedito abbiamo un start_timer, che farà scattare un timeout dopo l'intervallo prefissato.

2. Nello stato di attesa di riscontro con numero di sequenza zero se viene ricevuto un pacchetto (non corrotto e che contiene il riscontro con numero di sequenza zero) il timer viene interrotto.

3. Si passa nello stato simile a quello iniziale, ma è assegnato al numero di seq 1.

4. In questo stato è possibile che venga ricevuto un riscontro negativo, che si avrà solo nel momento in cui il ricevente ha ricevuto il pacchetto spedito in precedenza:
    1. potremmo avere come evento la ricezione di un riscontro negativo; quindi il ricevente ha ricevuto il pacchetto ma questo non è integro. **Questo avviene nel momento in cui l'ACK ricevuto come riscontro è diverso da quello con cui il pacchetto è partito**
    2. il pacchetto spedito potrebbe non essere stato consegnato affatto; dobbiamo quindi considerare l'evento di **timeout** . Se si presenta l'evento di timeout, il mittente ipotizza che il pacchetto spedito non sia stato per nulla ricevuto dal ricevente, e quindi lo rispedisce avviando nuovamente un timer.
    Riamiamo quindi allo stesso stato dell'automa finchè il pacchetto non viene ricevuto (integro o meno).

Vediamo il caso in cui venga perso un pacchetto - Versione grafica - Primo scenario

Durante l'invio di un pacchetto questo viene perso; siccome il ricevente non ha ricevuto nulal non può spedire alcun riscontro, quindi il mittente non può fare altro che attendere fine all'evento del timeout, e poi rispedire il pacchetto:

Secondo scenario

Se il pacchetto venisse sempre perso vuol dire che la rete ha un problema di discontinuità permanente, il numero di tentativi è **limitato**; tipicamente un pacchetto viene rispedito solo per un numero n di volte.

In questo caso viene perso non il pacchetto, ma il riscontro (ACK1), il pacchetto è quindi arrivato al destinatario ma il mittente lo rispedisce comunque:

Per il ricevente questo pacchetto è un duplicato, ma l'introduzione dei numeri di sequenza permettono al ricevente di capire che il pacchetto è duplicato; questo eprchè non possiamo avere due pacchetti continui **con lo stesso numero di sequenza**.

Terzo scenario

Questo scenario si evince come questo schema sia vulnerabile alla scelta del **timeout**:

A causa di un rallentamento transitorio della rete, che aumenta il ritardo lungo il percorso, abbiamo che il pacchetto di riscontro spedito dal ricevente (ACK1) non arriva in tempo, quindi il mittente va comunque in timeout, ed il mittente rispedisce il pacchetto, anche se questo era arrivato correttamente al ricevente.

Di conseguenza, per via del ritardo, l'ack1 precedente viene comunque ricevuto dal mittente, anche se con un ritardo. Il mittente spedisce un nuovo pacchetto perchè crede che il riscontro ack1 (ricevuto con ritardo) sia il pacchetto di riscontro di un nuovo pacchetto:

Performance dell'RDT3.0 - Stop and Wait

La verisione 3 del protocollo non è perormante, come mai?

Un problema può essere sicuramente il fatto che abbiamo un'attesa, proprio perchè il mittente si ferma finchè non riceve un riscontro, non può quindi sfruttare il canale al 100%.
Il throughput è determinato dal round trip time; questo valore è dato dalla quantità di bit inviati diviso il tempo impiegato per farlo.

Supponiamo di avere un pacchetto di n bit; inviamo questo bit in un intervallo temporale che è dato dalla somma di due contributi: **tempo di trasmissione** (lunghezza del pacchetto / larghezza di banda del canale) e **round trip time**, quindi otteniamo

Abbiamo la spedizione di n bit in un tempo pari a `l/r + RTT`.

Se consideriamo un altro fattore, ovvero l'utilizzo del canale vediamo come il canale sia usato pochissimo, quando usiamo questo protocollo.

🏁 1:01

Esempio pratico

Facciamo finta di avere un canale di trasmissione con una larghezza di banda pari ad 1Gps, con un ritardo di propagazione pari a 15ms e dobbiamo spedire dei pacchetti dal peso totale di 8000 bytes.

-- Inserisci foto calcolo qui

Proviamo a vedere cosa accade quando inviamo i dati:

Nell'immagine notiamo cosa accade: abbiamo un intervallo temporale che rappresenta il tempo di trasmissione, ed è il rapporto l/r, ed abbiamo un altro intervallo di tempo rappresentato dal RTT.

Si vede facilmente graficamente che il canale è impegnato solo per l'intervallo temporale pari ad **l/r**, mentre tutto il tempo rappresentato dal RTT è tempo di attesa

-- inserisci secondo calcolo

Vediamo come il canale viene realmente usato davvero poco rispetto a quello che potrebbe; **come possiamo migliorare le prestazioni?**

Dobbiamo tenere presente che le macchine potrebbero essere ad una grande distanza l'una dall'altra, quindi il RTT potrebbe essere elevato; più è basso il RTT più aumenta l'uso del canale. Puttosto che inviare il pacchetto ed attendere, possiamo inviare più pacchetti ed aspettare il meno possibile.

Utilizzo del pipelining

Invece di aspettare inviamo insuccessione i segmenti, in modo da usare il canale il più possibile.

Abbiamo un miglioramento del coefficiente dell'utilizzo del canale che è tre volte maggiore rispetto al precedente; ad ogni modo il coefficiente rimane molto basso. Per arrivare ad un coefficiente pari ad 1 dovremmo spedire molti altri pacchetti, in modo da **riempire completamente l'intervallo temporale** RTT.

Abbiamo però un problema: con i riscontri facciamo si che il mittente sappia se il pacchetto è arrivato correttamente o meno. In questo caso, il mittente continua a spedire i pacchetti **senza aver però ricevuto i riscontri**; di conseguenza il mittente deve mantenere in memoria tutti i pacchetti che ha spedito.

Go-Back-N: sender

Questi protocolli che utilizzano il pipelining sono anche detti "**a finestra scorrevole**", perchè prevedono l'utilizzo di una finestra di dimensione N; questo valore ci dice che il mittente può spedire N pacchetti **senza aspettarne il riscontro** e deve memorizzare tutti i pacchetti spediti di cui non ha ancora ricevuto il riscontro.

Fine lezione 22