martedì 3 ottobre 2017

Tcp

- Richiede più risorse perché ci sono i controlli, serve infatti per trasmissioni che devono arrivare in modo preciso
- Anche questo è nelle varie RFC
- Comunica in modo sincrono, cioè invio e conferma di ricezione, aspetto la risposta
- In caso di ricezione se non c'è buffer abbastanza si scartano i pacchetti

Le informazioni di controllo che il processo deve passare al TCP comprendono:

- **source address**: indirizzo completo del mittente (network+host+port);
- **destination address**: indirizzo completo del destinatario (network+host+port);
- **next packet sequence number**: il numero di sequenza che TCP deve assegnare al prossimo pacchetto che trasmetterà da quella porta;
- **current buffer size**: la dimensione del buffer del mittente;
- **next write position**: indirizzo dell'area del buffer in cui il processo pone i nuovi dati da trasmettere;
- **next read position**: indirizzo dell'area del buffer da cui TCP deve leggere i dati per costruire il prossimo segmento da inviare;
- **timeout/flag**: indica il tempo trascorso il quale i dati non riscontrati (*un-acknowledged*) devono essere ritrasmessi; il flag è usato per sincronizzare TCP e processo (per esempio tramite l'uso di semafori), per segnalazioni di stato ecc.

- Normalmente si bufferizza a meno che l'app richiede l'immediata trasmissione (imposta l'uscita a 1)
  - Altrimenti quando urg è impostato a 1, perché anche qua tutto viene spedito subito, se arriva qualcosa con urg=1 viene letta subito

I campi del segmento TCP (figura 3) sono:

- **source port number** (16 bit): numero di porta sull'host del mittente;
- **destination port number** (16 bit): numero di porta sull'host del destinatario;
- **sequence number** (32 bit): numero di sequenza progressivo del primo byte di dati contenuto nel segmento;
- **acknowledgment number** (32 bit): numero di riscontro, ha significato solo se il flag ACK è impostato a 1, conferma la ricezione di una parte del flusso di dati indicando il valore del prossimo *Sequence number* atteso (implicitamente si conferma che i byte precedenti sono stati ricevuti correttamente);
- **header length** (4 bit): indica la lunghezza (in word da 32 bit) dell'header del segmento TCP; tale lunghezza può variare da 5 word (20 byte) a 15 word (60 byte) a seconda della presenza e della dimensione del campo facoltativo Options. Serve quindi a indicare l'inizio dei dati del segmento;
- **not used** (4 bit): bit attualmente non utilizzati; devono essere impostati a zero;
- **flags** (8 bit): bit utilizzati per il controllo del protocollo:
  - **CWR** (Congestion Window Reduced): se impostato a 1 indica che l'host sorgente ha ricevuto un segmento TCP con il flag ECE impostato a 1 e ha di conseguenza abbassato la sua velocità di trasmissione per ridurre la congestione ed evitare la perdita di pacchetti;
  - **ECE** (ECN-Echo): se impostato a 1 indica che l'host supporta ECN (Explicit Congestion Notification) durante il 3-way handshake;
  - **URG**: se impostato a 1 indica che nel flusso sono presenti *dati urgenti* e quindi si deve leggere il campo *Urgent pointer*;
  - **ACK**: se impostato a 1 indica che il segmento TCP in questione è in risposta ad un altro ricevuto che conteneva dati di conseguenza indica che il campo *Acknowledgment number* è valido e si devono leggere le informazioni in esso contenute;
  - **PSH** (push): se impostato a 1 indica che i dati in arrivo non devono essere bufferizzati ma passati subito ai livelli superiori dell'applicazione;

- **window size** (16 bit): è usato dall'host ricevente per dire al mittente quanti dati può ricevere in quel momento (*finestra di ricezione*), cioè il numero di byte che il mittente può spedire a partire dal byte confermato (specificato dall'*acknowledgment number*). Il valore 0 indica di non inviare altri dati per il momento; quando il ricevente sarà di nuovo in grado di ricevere dati invierà al mittente un segmento con *window size* diverso da 0 ma con lo stesso dall'*acknowledgment number*;
- **checksum** (16 bit): è utilizzato per la verifica della validità del segmento. L'algoritmo di calcolo è definito nell'RFC ed è del tutto simile a quello di UDP, unica differenza è che per TCP il campo è obbligatorio. Se dal calcolo della checksum risultasse che il segmento TCP è stato danneggiato, non verrebbe riscontrata la sua ricezione e quindi il mittente lo dovrebbe ritrasmettere;
- **urgent pointer** (16 bit): ha significato solo se il flag URG = 1, contiene il numero che deve essere sommato (*offset*) al *Sequence number* per ottenere il numero dell'ultimo byte urgente nel campo dati del segmento. Questo campo consente all'applicazione di usare messaggi che interrompono la normale elaborazione;
- **options**: campo facoltativo che può avere dimensione variabile da 0 a 10 word (word = 32 bit). L'opzione più importante è quella che consente ad un host di specificare la dimensione massima del segmento che è in grado di accettare. Il default è 536 byte di dati e 20 byte di header (quindi ogni host deve poter gestire segmenti di almeno 556 byte);
- **data**: contiene i dati da trasmettere provenienti dal livello superiore o, nell'altra direzione, i dati ricevuti dal livello inferiore.

- Il tcp considera la rete come una lan, quindi una rete fisica, quindi non considera le eventuali tratte radio
- TCP si basa quindi su deduzioni
- Le primitive son funzioni di sistema operativo tipo interfaccia

---

Fase di instaurazione di una sessione TCP

Handshake a tre vie

1) HOST 1 Invia segmento TCP con San impostato a 1 + numero casuale che diventa sequence number (x)
2) Se host 2 acconsente, risponde con un ack.f impostato a 1 e l'ack number impostato al x+1
    1) Inoltre per stabilire la connessione imposta il proprio sync a 1 e genera un altro numero di frequenza (y)
3) 1 risponde con una altra conferma ace impostato a 1, quindi l'ace è impostato al valore ricevuto di y+1

Si cerca di far si che non si superino 1500 byte per semplificare il livello datalink

Ogni host deve conoscere il sequence number iniziale dell'altro

---

Fase di trasmissione dati

- Gestisce il controllo di flusso e trasmissione con Sliding Windows ed è simile al datalink MA

- in TCP il puntatore nella finestra è al singolo byte, mentre a livello Data Link è al frame;
- in TCP la dimensione della finestra è variabile mentre a livello Data Link è fissa.

- Se si riempie il buffer e non viene svuotato, il ricevente può sospendere la trasmissione anche se è già iniziata
- In datalink la soluzione è hardware mentre in tcp è software
- Quando viene restituito l'ace viene inviata la grandezza rimanente del buffer
- Il mittente non invia un numero di byte superiore a quello

- Il ricevente se dice che ne ha 10, 10 ne riceve, per forza
- Se la finestra è 700, ti invio 500, tu mi devi rispondere 200
- Qua non c'è il Nack ma c'è un timer
- Ogni volta quindi gli dice dove è arrivato
- Si presuppone che ogni volta che si conferma un numero, tutto quel che c'era prima, è stato ricevuto
- Un doppio Ack viene interpretato come un Nack

Nell'handshake modificato

Si imposta FIN anziché Syn a 1

- **RST** (reset): se impostato a 1 indica che la connessione non è valida; viene utilizzato in caso di grave errore; a volte utilizzato insieme al flag ACK, per la chiusura di una connessione;
- **SYN** (SYNchronize sequence numbers): è usato nella fase di instaurazione di una connessione; se impostato a 1 indica che l'host mittente del segmento vuole aprire una connessione TCP con l'host destinatario e specifica nel campo *Sequence number* il valore dell'*Initial Sequence Number* (ISN) per sincronizzare i numeri di sequenza dei due host. L'host ricevente invia poi la risposta SYN-ACK con SYN=1. Questo flag deve essere usato solo in questi due casi; SETUP
- **FIN** (final): se impostato a 1 indica che l'host mittente del segmento non ha più dati da inviare e vuole *chiudere la connessione TCP*. Il ricevente invia la conferma di chiusura con un FIN-ACK. A questo punto la connessione è ritenuta chiusa in un verso: l'host che ha inviato il FIN non potrà più inviare dati, mentre l'altro host ha il canale di comunicazione ancora disponibile. Quando anche l'altro host invierà il pacchetto con FIN=1 la connessione, dopo il relativo FIN-ACK, sarà considerata completamente chiusa;