1 Lezione del 17-10-25

1.0.1 Pseudocodice del selective repeat

Continuiamo la discussione del selective repeat. Per farlo, non usiamo più un approccio FSM, ma descriviamolo in *pseudocodice*.

Vediamo allora il trasmettitore:

Algoritmo 1 Trasmettitore *selective repeat*

if ci sono dati dall'alto **then**
  Se il prossimo numero di sequenza è nella finestra, invia il pacchetto
end if
if timeout(n) **then**
  Reinvia il pacchetto n e riavvia il timer
end if
if ACK(n) nella finestra **then**
  Marchia il pacchetto n come *ricevuto*
  Se era il pacchetto non ricevuto con numero di sequenza minore, avanza la finestra fino al prossimo pacchetto non ricevuto
end if

e quindi il ricevitore:

Algoritmo 2 Ricevitore *selective repeat*

if ricevuto pacchetto n in finestra **then**
  invia ACK(n)
  se è fuori ordine, bufferizzalo
  se è in ordine, consegna il pacchetto (e quindi i successivi bufferizzati), e avanza la finestra al prossimo pacchetto non ancora ricevuto
end if
if ricevuto pacchetto n prima della finestra **then**
  invia ACK(n)
end if
if ricevuto pacchetto n dopo la finestra **then**
  ignoralo
end if

Notiamo quindi che sia il trasmettitore che il ricevitore mantengono le loro finestre, il primo per i pacchetti marchiati come da inviare, il secondo per i pacchetti che può bufferizzare.

1.1 Protocollo PPP

Vediamo quindi un protocollo reale di livello *datalink* che implementa le funzionalità che abbiamo descritto finora. Questo è il protocollo **PPP** (*Point-to-Point Protocol*), e viene oggi usato dai router per parlare direttamente l'uno con l'altro.

Le funzionalità del PPP sono:

- **Framing** pacchetti: provvede all'incapsulamento dei datagrammi livello network in frame livello datalink. Può portare dati livello network di qualsiasi protocollo di rete (non solo IP);
- **Trasparenza bit**: è capace di portare qualsiasi pattern di bit nel campo dati;
- **Rilevamento errori**: c'è ma non c'è correzione;
- **Verifica connessione**: permette di verificare e segnalare il fallimento del link al livello network;
- **Negoziamento indirizzi** al livello network: gli host possono imparare e configurare gli indirizzi di rete altrui.

Vediamo che non è presente quanto abbiamo detto sul *trasferimento affidabile* di dati: questo lo rende effettivamente un link **inaffidabile** (abbiamo visto che questa funzionalità viene reintrodotta a livello *transport*).

In particolare, PPP non è provvisto di:

- Correzione errori;
- Recupero da errori;
- Controllo di flusso;

- Consegna fuori ordine;
- Comunicazioni *point-multipoint*.

Ci si aspetta invece che queste funzionalità vengano delegate a protocolli di livello superiore come *TCP*.

1.1.1 Pacchetto PPP

Iniziamo a vedere la struttura di un pacchetto PPP:

```
1 byte    1 byte    1 byte    1 byte     variabile  2 o 4 byte  1 byte
<flag>    <address> <control> <protocol> <info>     <check>     <flag>
```

- I campi **flag** fanno da delimitatori per il framing;
- Il campo **address** è effettivamente inutile: il PPP non permette comunicazine *point-multipoint* (più destinatari) e quindi è sempre impostato tutto a 1 (l'indirizzo di *broadcast*);
- Il campo **control** è ugualmente inutile;
- Il campo **protocol** stabilisce il protocollo di livello superiore da usare (ad esempio TCP);
- Il campo **info** contiene i dati veri e propri, ed è a dimensione variabile da 0 a 1500 byte (la dimensione massima può essere negoziata);
- Il campo **check** supporta il CRC per il rilevamento errori.

Notiamo che garantire la *trasparenza bit* non è immediato: se si spedisse un campo info che contiene una copia esatta del marcatore di flag terminale che ci aspettiamo, come dovrebbe fare il ricevitore a capire che tale campo va spedito e non è effettivamente il terminatore?

Risolviamo il problema sfruttando il **byte stuffing/unstuffing**, meccanismo per noi sostanzialmente analogo a quello delle *sequenze di escape*. Si antepone quindi al byte problematico un'altro specifico byte, di escape, che il ricevitore potrà poi interpretare come segnalatore (di prendere *"alla lettera"* il byte sucessivo), e quindi buttare via.

1.2 Link multipli

Veniamo ora a descrivere le reti formate da più **link di accesso**. La prima distinzione che dobbiamo fare è fra i due tipi di link che possiamo incontrare:

- Link **punto-punto**: sono del tipo che abbiamo visto finora, e collegano solo due dispositivi fra di loro;
- Link **broadcast** (detti anche a *mezzo condiviso*): sono tipici di tecnologie come il *vecchio Ethernet*, le reti mobili e reti wireless come 802.11 (*WiFi*). In questo tipo di rete i dispositivi possono parlare con tutti gli altri dispositivi, contemporaneamente.

Il problema delle reti broadcast è chiaramente l'**interferenza**: si possono verificare *collisioni* se un nodo riceve 2 o più segnali contemporaneamente.

1.2.1 Protocolli MAC

Dobbiamo quindi definire un protocollo di accesso multiplo, cioè un **algoritmo distribuito** che determini come i nodi condividono il canale (decida quale nodo può trasettere). Il problema è che tale protocollo deve sfruttare comunicazione sul canale condiviso stesso, in quanto non abbiamo sempre a disposizione un altro canale *fuori banda* da usare per la coordinazione delle comunicazioni.

Quello che desideriamo è, dato un canale ad accesso multiplo **MAC** (*Multiple Access Channel*) con capacità di R bit al secondo, che:

1. Quando un nodo vuole trasmettere, può farlo a capacità R;
2. Quando M nodi vogliono trasmettere, questi possono farlo a capacità R/M;
3. Il sistema sia completamente decentralizzato: non ci siano nodi speciali che coordinano le comunicazioni, né clock di sincronizzazione;
4. Il sistema sia *semplice*.

Chiameremo i protocolli che ci permettono di fare ciò protocolli **MAC** (acronimo già visto).

1.2.2 Tassonomia dei MAC

Esistono tre classi principali di protocolli MAC:

1. **Partizionamento canali**: si divide il canale in *"pezzi"* (slot temporali, bande di frequenza), e si alloca ogni *"pezzo"* all'uso esclusivo di un nodo. Un idea interessante è quella della sovrapposizione di **codice**: in questo caso ogni nodo parla usando una codifica particolare, e si riesce a distinguere fra nodi rilevanto tali codifiche;
2. **Accesso casuale**: il canale non viene diviso e si permettono le collisioni. In questo caso chiaramente l'approccio alla trasmissione sarà di:
   - Rilevare quando il canale è libero;
   - Aspettare un quanto temporale (sperabilmente piccolo e casuale, in modo da ridurre le collisioni);
   - Iniziare a trasmettere.

   In questo caso chiaramente bisogna dotarsi di un sistema per *recuperare* le collisioni, che inevitabilmente prima o poi accadranno;
3. **Turni**: i nodi si dividono a turni, ma i nodi con più dati da inviare possono prendere turni più lunghi.

1.2.3 Protocollo TDMA

Vediamo il primo protocollo MAC, il **TDMA** (*Time Division Multiple Access*). Questo fa parte della categoria (1) dei protocolli MAC, e quindi è a *partizionamento canali*.

Si fornisce l'accesso al canale di accesso in *round*. Ogni stazione otiene uno slot di accesso a lunghezza fissa in ogni round. Gli slot non utilizzati tengono il canale a riposo.

Con questo protocollo riusciamo a soddisfare i primi 2 requisiti che ci eravamo dati. La decentralizzazione non è immediata, in quanto bisogna capire quando i round iniziano e l'ordine di comunicazione ad ogni round. Immaginiamo che i nodi riescano comunque a coordinarsi automaticamente.