1 Lezione del 21-10-25

Riprendiamo la discussione dei protocolli MAC.

1.0.1 Protocollo FDMA

Avevamo parlato del protocollo *TDMA*. Adesso vediamone la versione in partizionamento di stretto: l'**FDMA** (*Frequency Division Multiple Access*). non l' ha fatto

1.1 MAC ad acesso casuale

Veniamo quindi a discutere i protocolli ad **accesso casuale**. In questo caso ipotizziamo che:

- Quando un nodo ha un pacchetto, vuole trasmetterlo al bitrate massimo R;
- Non c'è alcun tipo di coordinazione *a priori* dei nodi.

In questo caso, chiaramente si incorre nel problema delle *collisioni*.
I protocolli **MAC random access** specificano come:

- Rilevare collisioni;
- Recuperare dalle collisioni (ad esempio ritrasmettendo con ritardi casuali).

Esempi di protcolli MAC random access sono **ALOHA** (originariamente acronimo *Additive Links On-line Hawaii Area*) e **CSMA** (*Carrier Sense Multiple Access*).

1.1.1 Protocollo ALOHA slotted

Il protocollo **ALOHA** (*Additive Links On-Line Hawaii Area*) nasce da un progetto di Norman Abramson, per coordinare le comunicazioni fra le sedi dell'università delle Hawaii.

Vediamo la versione di ALOHA detta **ALOHA slotted** (tempo diviso in *slot*). Assumiamo che:

- Tutti i frame hanno la solita dimensione prefissata;
- Il tempo è diviso in slot di dimensioni uguali (corrispondenti al tempo necessario a trasmettere 1 frame);
- I nodi iniziano a trasmettere solo all'inizio degli slot;
- I nodi sono sincronizzati;
- Se 2 o più nodi iniziano a trasmettere all'inizio di uno slot, tutti gli slot rilevano una collisione.

La modalità di operazione dei nodi in ALOHA è quindi la seguente:

- Quando un nodo ottiene un frame da trasmettere, lo trasmette al prossimo slot;
- Se non c'è collisione, la trasmissione va a buon fine e il frame viene trasmesso;
- Se c'è collisione, il nodo ritrasmette il frame in ogni slot seguente con un certa probabilità p finché la trasmissione non riesce.

Le collisioni vengono verificate controllando se si riceve un ACK al termine della trasmissione: notiamo che questo rende identici (dal punto di vista del trasmettitore) gli errori di trasmissione di ACK e le collisioni. I trasmettitori in ALOHA assumono che ci sia comunque bisogno di ritrasmettere.

I pro dell'approccio sono che ogni nodo può trasmettere al bitrate R completo (1), il sistema è altamente decentralizzato (3) (assunta sincronizzazione), ed è semplice (4).

I contro sono che le collisioni riducono il bitrate effettivo, sprecando slot e quindi impedendo che M nodi possano trasmettere a capacità R/M (3). Inoltre, si possono verificare slot fermi. Infine, notiamo che abbiamo bisogno di sincronizzazione di clock o comunque una sincronia che permetta ai nodi di capire quando gli slot iniziano.

1.1.2 Efficienza di ALOHA slotted

Facciamo alcune considerazioni quantitative sull'efficienza del protocollo appena visto. Assumiamo che N nodi vogliano trasmettere frame negli slot con probabilità p.

La probabilità che ogni nodo trasmetta con successo in uno slot sarà:

$$p_{\text{nodo}} = p(1-p)^{N-1}$$

per cui la probabilità che un *qualsiasi* nodo trasmetta con successo in uno slot sarà:

$$p_{\text{nodi}} = Np(1-p)^{N-1}$$

Vogliamo quindi trovare p^* che massimizza p_{nodi}. Derivando p_{nodi} su p si ha:

$$\frac{d}{dp}p_{\text{nodi}} =$$

che imponiamo uguale a zero: finisci

$$\frac{d}{dp}p_{\text{nodi}} = 0 \implies$$

1.1.3 Protocollo ALOHA pure

Eliminiamo l'ipotesi della sincronizzazione dal protocollo ALOHA in versione slotted: in questo caso, appena arriva un frame, il nodo inizia subito a trasmettere. Chiamiamo questo protocollo **ALOHA pure**.

La probabilità delle collisioni in questo caso incrementa: ogni frame inviato al tempo t_0 collide con i frame inviati fra $[t_0-1, t_0+1]$ (dove l'unita corrisponde allo slot temporale dedicato ad un frame).

1.1.4 Efficienza di ALOHA pure

divertiti coi calcoli

Abbiamo quindi trovato un protocollo che è molto più semplice e decentralizzato (non richiede sincronizzazione), ma ha un utilizzazione del canale di comuniazione che è di molto minore della versione slotted.

1.1.5 Protocollo CSMA

Il protocollo **CSMA** (*Carrier Sense Multiple Access*) prevede di *ascoltare* prima di trasmettere: in questo caso si può rilevare la condizione del canale condiviso prima di provare ad accedervi. In particolare, dopo aver ascoltato:

- Se il canale è rilevato fermo, si trasmette l'intero frame;
- Se il canae è rilevato attivo, si differisce la trasmissione ad un secondo momento.

Questo è in qualche modo analogo al modo in cui gli umani usano mezzi condivisi (ad esempio l'etere quando parlano a voce): prima si ascolta cosa dicono gli altri, e poi si parla. La regola fondamentale è *"non interrompere gli altri!"*.

CSMA ha una versione detta **CSMA/CD**, cioè *CSMA with Collision Detection*. In questo caso le collisioni vengono rilevate entro qualche tempo limitato. Le trasmissioni in collisione vengono abortite, riducendo lo spreco del canale. Questo è piuttosto facile per i mezzi cablati, più difficile per i mezzi wireless.

Il CSMA/CD è necessario in quanto le collisioni possono comunque accadere dopo l'ascolto del canale (*carrier sensing*): i tempi di propagazione significano infatti che due nodi potrebbero non sentire le trasmissioni appena iniziate l'uno dell'altro.

1.1.6 Rilevamento collisioni CSMA

Per rilevare le collisioni si potrebbe pensare di mettere a comparatore il segnale che il nodo sta trasmettendo e quello che sta ricevendo: se il delta è maggiore di qualche soglia, dev'essere che c'è un altra sorgente di segnale e quindi siamo in collisione.

In verità l'approccio effettivamente usato è più semplice: invece di parlare di *segnali*, si parla di *potenze* rilevate sul mezzo di trasmissione. L'approccio è comunque funzionale: se il trasmettitore si aspetta di poter erogare una potenza P, rilevando potenze $P^* >> P$ sul mezzo potremo concludere con un certo grado di sicurezza di essere in collisione.

1.1.7 CSMA su Ethernet

Vediamo quindi l'implementazione di CSMA sul mezzo Ethernet. Vediamo cosa fa la **NIC** (*Network Interface Card*) quando vuole trasmettere un frame.

1. La NIC riceve il datagramma dal livello network, e ne crea un frame (siamo a livello datalink);
2. La NIC ascolta il mezzo (*channel sensing*):
   - Se il mezzo è rilevato fermo, inizia con la trasmissione del frame;
   - Se il mezzo è rilevato attivo, si aspetta finché non è fermo, e quindi si trasmette.
3. Se la NIC riesce a trasmettere l'intero frame senza collisioni, abbiamo finito;
4. Se si verificano altre trasmissioni mentre si trasmette, cioè una *collisione* (vedi sezione sopra), si abortisce la trasmissione e invia il cosiddetto segnale di **jam**: questo è un segnale a potenza più alta della media che ha lo scopo di avvisare gli altri trasmettitori che una collisione si è verificata;
5. Dopo aver abortito, la NIC entra in un **backoff esponenziale**:

- Dopo la m-esima collisione, sceglie un K casuale fra $\{0, 1, 2, ..., 2^m - 1\}$ (binario). Quindi il NIC aspetta per $K \times 512$ tempi bit (per *tempo bit* intendiamo il tempo necessario a trasmettere un bit con bitrate R di mezzo), e quindi torna al passo (2);
- Il risultato è che più collisioni si hanno, più lungo è in media l'intervallo di backoff (attesa).

Chiaramente, per implementare tale politica dovremo dotarci di un contatore per le m collisioni rilevate. Inoltre, sarà utile prevedere una dimensione massima per la finestra dove scegliere K, cioè un numero di collisioni oltre cui gli intervalli di backoff non continuano ad aumentare (in Ethernet questo numero è 10, per cui il tempo massimo è $2^10 - 1 = 1023$ tempi bit).

Vediamo cosa abbiamo ottenuto con CSMA su Ethernet: il requisito di completa decentralizzazione è soddisfatto (3), il sistema è effettivamente abbastanza semplice (4), e le frequenze di trasmissione sono perlopià soddisfatte, salvo collisioni da recuperare (requisiti (1) e (2)). Il problema rimasto è quello degli alti carichi: il backoff esponenziale implica che il sistema può rallentare fino a troughput nulli se si verificano collisioni particolarmente gravi (quindi inevitabilmente quando ci sono molti nodi).

1.2 MAC a turni

Veniamo quindi ai protocolli MAC basati sui turni.

1.2.1 Polling

Come primo esempio vediamo il meccanismo del **polling**. Prevediamo un nodo, detto *master*, che invita gli altri nodi (detti *slave*) a trasmettere a turno.

Questo approccio è usato spesso per dispositivi *"stupidi"* (è usato ad esempio in Bluetooth).

Per quando riguarda i nostri requisiti, non è assolutamente decentralizzato (3), è abbastanza efficiente (requisiti (1) e (2)) ed è sempre abbastanza semplice (4).

Si comporta bene agli alti carichi (il nodo centralizzato governa gli altri assicurando throughput massimo) ma non ai bassi (si sprecano molti turni), proprio come TDMA (sezione 12.3.1).

I problemi sono poi l'*overhead* dato dal polling, la *latenza* introdotta e il fatto che il master rappresenta un *single point of failure*. Quest'ultimo problema potrebbe essere risolto prevedendo un algoritmo di *rielezione* da mettere in esecuzione al momento della morte del master. Questo, però, va chiaramente in contro al requisito (4) (semplicità).

1.2.2 Passaggio di token

Un altro modo per implementare comunicazioni a turni è il **token passing**. Questo token (di controllo) rappresenta un qualche segnalatore che chi vuole trasmettere deve possedere per poterlo fare. Il token viene passato da un nodo all'altro, sequenzialmente, in modo che tutti i nodi possano parlare.

I problemi principali saranno quindi la *latenza*, la *gestione* del token stesso, nonché il fatto che questo rappresenta nuovamente un *single point of failure* per l'intero sistema (se il token va perso, chi può parlare?).

La latenza si ha dal fatto che il token deve essere passato, e questo rappresenta un *overhead* (il tempo passato a passarsi il token non è passato a trasmettere).