1 Lezione del 28-10-25

Ritorniamo sull'argomento dell'*internetworking*.

Al livello network il livello superiore è il transport: vorremo prendere i *segmenti* transport e trasportarli in *datagrammi*.

- Il **trasmettitore** dovrà quindi prendere i segmenti dal livello superiore e incapsularli in datagrammi;
- Il **ricevitore** dovrà invece prendere i datagrammi, estrarne i segmenti e fornirli al livello superiore.

Questo protocollo è implementato in tutti i dispositivi a livello network: questo significa **host** e **router** (gli switch non sono dispositivi a livello network, ma a livello datalink).

Il compito dei *router* è quindi chiaro a livello network: questi leggono i campi header dei *datagrammi IP* che ricevono, e inoltrano (*instradano*, da *routing*) tali datagrammi verso la destinazione finale (attraverso la porta di uscita giusta) consultando una *routing table*.

1.0.1 Funzioni livello network

Il livello network implementa due funzioni fondamentali:

- **Forwarding**: spostare pacchetti da un *link di ingresso* del router ad un appropiato *link di uscita* (quella che prima abbiamo chiamato *porta di uscita*);
- **Routing**: determinare il percorso fatto dai pacchetti dalla sorgente alla destinazione. In questo caso dobbiamo dotarci di appositi *algoritmi di routing*.

1.0.2 Data plane e control plane

Abbiamo quindi che i router lavorano fondamentalmente su due *piani*: **piano dati** (*data plane*) e **piano di controllo** (*control plane*).

- Il **piano dati** è una funzione *locale*, che coinvolge il singolo router, e che riguarda lo spostamento del datagramma da porta di ingresso del router a porta di uscita;
- Il **piano di controllo** è invece una funzione *globale* alla rete, condivisa fra i router, che determina come il datagramma deve essere instradato fra i router nel percorso (**route**) da host mittente e host destinatario.

  Esistono 2 modi per realizzare il piano di controllo:

  - Adottare degli algoritmi di routing *tradizionali*, implementati direttamente nel router.

    Chiaramente, questi algoritmi di routing dovranno essere distribuiti, cioè eseguiti da tutti i router che lavorano insieme per assicurare il corretto instradamento dei pacchetti (aggiornando le forwarding table di ogni router);
  - Sfruttando il *Software Defined Networking* (**SDN**), cioè implementando gli algoritmi di routing nei server e facendo semplicemente eseguire al router il percorso trovato.

    In questo caso si dovrà prevedere un controllore remoto che si occupi di installare le forwarding table in ogni router.

1.0.3 Modello di servizio

Possiamo interrogarci quali sono le caratteristiche del servizio che Internet è capace di offrire attraverso il protocollo IP (quello che nell'Internet moderno implementa il livello network).

Abbiamo che IP non garantisce nulla riguardo a *bitrate*, *perdita di dati*, *ordinamento dei dati* o *temporizzazione*.

In questo rappresenta un servizio **best effort**:

- Non si garantisce la consegna assicurata dei pacchetti;
- Non si assicurano costanti minime di temporizzazione o l'ordinamento dei pacchetti;
- Non si assicura la larghezza di banda massima al flusso da host a host.

Il protcollo best effort potrebbe sembrare inizialmente svantaggioso, ma per Internet la fortuna è stata proprio rendere la rete *"stupida"*, e concentrare l'intelligenza all'*edge* (agli host, si pensi ai protocolli di trasporto, ecc...).

Questo rende infatti molto più semplice ed economico realizzare l'infrastruttura di rete, e allocando abbastanza risorse in un secondo momento si riesce ad ottenere un servizio *"abbastanza buono"* anche per applicazioni intensive sul bitrate come le videochiamate, ecc...

1.1 Router

Un **router** è in maniera estremamente generica un computer che ha delle **porte di entrata** e delle **porte di uscita**.

Fra porte di ingresso e porte di uscita deve essere posta una qualche **rete di switching** estremamente veloce che permetta la commutazione da ingresso a uscita. Questa implementa il *data plane*, e quindi il *forwarding*.

Questa dovrà quindi essere governata da un **processore di routing**. Questo implementa il *control plane*, e quindi il *routing*.

Le porte del router implementano i primi 2 livelli dello stack protocollare, cioè:

1. Livello **fisico**: si occupa della ricezione bit a bit del datagramma;
2. Livello **link**: ad esempio di tipo Ethernet, si occupa del rilevamento errori e si libera del frame di livello link;
3. Una volta superato il livello link, i payload dei frame di livello link (cioè i datagrammi veri e propri) vengono inoltrati effettuando un lookup sul loro campo header.

   Il forwarding può essere implementato in 2 modi:

   - Forwarding **basato su destinazione**: basato solo sull'indirizzo IP estratto dall'header di datagramma.

     Abbiamo introdotto le *switch table* in 15.2.1. Le **forwarding table** usate dai router sono molto simili: si definiscono *range* di indirizzi che mappano a porte di uscita dei router.

     Quando i frame si sovrappongono, si usa il cosiddetto **longest prefix matching**, cioè si prova ad usare sempre l'indirizzo con prefisso di rete più lungo.

- Forwarding **generalizzato**: basato su tutti i campi dell'header di datagramma.

Si prevede solitamente una coda per i datagrammi in arrivo, che viene riempita quando i datagrammi arrivano più velocemente di quanto si riesca ad instradarli.

1.1.1 Rete di switching

La **rete di switching** (dette anche *switching fabric*) si occupa di trasferire i pacchetti dal link di ingresso al link di uscita. Definiamo **switching rate** la frequenza con cui i pacchetti vengono trasferiti da input a output. Solitamente viene esprressa come un multiplo del bitrate di linea input/output: chiaramente con N input si cerca di avere uno switching rate pari ad N volte il bitrate di linea.

Esistono 3 tipi principali di switching fabric:

1. Basato su **memoria**: si direbbe a *memoria condivisa*. Solitamente viene infatti implementato nei computer tradizionali che fanno switching attraverso la CPU.

   In questo caso si prendono i pacchetti in entrata e si copiano in memori asistema. Dalla memoria sistema si inoltrano quindi verso le porte di uscita.

   Questo chiaramente è limitato dal bitrate della memoria (bisogna passare dal bus 2 volte per datagramma, una volta in entrata dalla porta di ingresso alla memoria, una volta in uscita dalla memoria alla porta di uscita);

2. A **bus**: si usa un bus per collegare le porte di ingresso direttamente alle porte di uscita.

   In questo caso il problema principale è chiaremente la competizione sul bus, e il fatto che la velocità di trasferimento è limitata dal bitrate del bus;

3. A **rete di interconnessione**: in questo caso si sfruttano tecnologie sviluppate inizialmente per connettere i processori nei sistemi multiprocessore (*crossbar*, *clos networks*, ecc...).

   L'evoluzione naturale di questi sistemi è l'uso di **switch multistage**: switch $n \times n$ realizzate attraverso più stadi formati da switch più piccole. Questo approccio può sfruttare un certo grado di *parallelismo*: si divide il datagramma in più *celle* di lunghezza fissa, che vengono immesse nello switch e riassemblate in uscita.

Notiamo che ulteriore parallelismo può essere sfruttato prevedendo più strutture di commutazione parallele (se vogliamo, più *data plane* paralleli).

1.1.2 Accodamenti in ingresso

Se la rete di switching ha bitrate più lento del bitrate combinato delle porte di ingresso, si potrebbero formare delle **code** alle porte di ingresso. Questo porta a delay ed eventualmente anche perdita di dati.

- Questo accade ad esempio in caso di **competizione** su una singola porta di uscita: se 2 pacchetti destinati alla stessa porta di uscita arrivano contemporaneamente a porte di ingresso diverse, uno dei due dovrà necessariamente aspettare che il secondo venga instradato;
- I pacchetti successivi a blocchi per competizione su porte di uscita possono quindi subire blocking **HOL** (*Head Of Line*): in questo caso bisogna attendere che la porta di ingresso si liberi perché il pacchetto in arriva venga effettivamente processato.

1.1.3 Accodamenti in uscita

Anche le porta di uscita possono essere suscettibili a code: potremmo infatti richiedere del **buffering** in uscita quando la logica di commutazione è più veloce della logica di uscita (quella che implementa i livelli datalink e fisico, in quest'ordine).

1.1.4 Gestione di perdite

Anche in questo caso possono verificarsi delay ed eventualmente **perdita di dati**. Alcune politiche di gestione del buffer che potremmo adottare in questo caso potrebbero essere:

- **Tail drop**: si scarta (*"drop"*) il pacchetto in arrivo sul buffer pieno;
- **Prioritaria**: si scartano i pacchetti su base prioritaria.

Può essere utile anche **marchiare** (*"marking"*) quali pacchetti conviene scartare per primi in caso di congestione: ad esempio, i pacchetti ICMP saranno i primi ad essere scartati.

1.1.5 Scheduling in uscita

Anche su come si effettua lo **scheduling** dei pacchetti in uscita si possono fare delle considerazioni. Dovremo infatti prevedere un *server* (di livello link) che si occupa di accedere alla coda dei pacchetti in uscita, e immettere nel livello fisico i pacchetti.

Questi possono essere selezionati (*schedulati*) secondo più modalità:

- **FCFS** (*First Come First Served*): il primo pacchetto viene inoltrato;
- **Prioritaria**, cioè si assegna una qualche priorità assegnata dall'applicazione o al router al pacchetto;
- **RR** (*Round Robin*): si servono ciclicamente tutti i pacchetti;
- **Weighted fair queueing**: si cerca di assegnare probabilità eque di trasmissione ad ogni pacchetto.

Approcci alternativi al FCFS possono essere utili in quanto non tutte le applicazioni che girano in rete sono uguali: alcune sono più importanti di altri (l'esempio precedente, pensiamo ICMP contro streaming ad alta velocità).

- Un buon approccio ibrido può essere quello a **code multiple** FCFS. Si prevedono ad esempio due code, una **prioritaria** e una **non prioritaria**, gestite entrambe internamente come FCFS.

  Il traffico in entrata viene classificato fra queste due code, e quindi il traffico prioritario viene spedito per primo.

  Un problema apparente di questo approccio è la *starvation* del traffico non prioritario.

- Per risolvere il problema della starvation potremmo prevedere lo stesso meccanismo di classificazione, e distinguere fra le code non in maniera prioritaria ma usando il **WFQ** (*Weighted Fair Queueing*).

Questo non è altro che un approccio round robin generalizzato dove ogni classe i ha peso w_i e ottiene un livello di servizio (magari in probabilità) ad ogni ciclo di:

$$w_g = \frac{w_i}{\sum_j w_j}$$

Questo approccio è quello che viene più usato nei moderni router.

1.2 Protocollo IP

Il **protocollo IP** è oggi il protocollo a fondamento del livello network. Ne esistono di altri, ma in Internet si usa principalmente IP.

Il livello network, e quindi il protocollo IP, ha il compito di implementare algoritmi di **path-selection** (protocolli di routing) sulla base di date *forwarding table*.

Il protocollo IP in praticolare si occupa di definire:

- Il **formato** dei datagrammi;
- Le modalità di **indirizzamento** a livello network;
- **Convenzioni** per la gestione dei pacchetti.

Il protcollo IP si affianca all'**ICMP** (*Internet Control Message Protocol*), inizialmente pensato per la notifica degli errori e la diagnostica, oggi usato perlopiù per diagnostica.

Datagramma IP Il **datagramma IP** ha una struttura più complessa di quella che avevamo visto ad esempio per i frame Ethernet.

Si hanno quindi almeno 5 parole da 32 bit di *header*, e una sezione di *payload* di lunghezza variabile:

```
16 bit                                  16 bit
<ver> <header length> <service type>    <length>                    % header
<identifier>                            <flag>  <fragment offset>
<time to live>        <upper layer>     <checksum>
<source address (IP)>
<destination address (IP)>
<options>
<payload> (lunghezza variabile)                                     % payload
```

- Il campo **version** è su 4 bit, ed è seguito dalla **lunghezza dell'header**, il **tipo** di servizio offerto (inutilizzato, qui si metteva ad esempio il *marking* di congestione) e la **lunghezza dell'intero datagramma**;
- La lunghezza massima di un datagramma IP è di 64 Kb, anche se solitamente si rimane sui 1500 bytes o meno;
- Il campo **time to live** determina gli *hop* di rete (trasferimenti da router a router) che il pacchetto può ancora avere prima di essere scartato. Assumiamo che ogni router decrementi questo campo, appunto, ad ogni hop effettuato;
- Il campo **upper layer** contiene informazioni sul tipo di protocollo di trasporto usato (TCP o UDP);
- Si ha quindi un campo di **checksum** per il controllo di errori;
- Seguono i campi di **indirizzo** (sorgente e destinazione IP);

- Si ha un campo opzionale di **options** specifiche al pacchetto;
- Infine c'è il **payload** vero e proprio.