1 Lezione del 10-10-25

Torniamo sull'argomento dello streaming video.

1.1 Tecniche avanzate per lo streaming

Avevamo introdotto l'idea del **playout delay**, dove bufferizzaviamo i dati video in ingresso per "spostare in avanti" in qualche modo la zona di dati compromessa dal *jitter* (irregolarità) della rete.

Visto che il jitter può essere massiccio, e la quantità di dati da inviare non indifferente, esistono alcune strategie più effiaci come ad esempio **DASH**.

1.1.1 DASH

DASH (*Dynamic, Adaptive Streaming over HTTP*) è una tecnologia che prevede di memorizzare i file video in più chunk, ogninuo memorizzato con ridondaza a diversi bitrate. Un *manifest file* fornirà gli URL di tutti i chunk, individuati quindi da indice nel video e livello di compressione.

Il *client* dovrà a questo punto misurare la velocità di trasmissione del server, e quindi consultare il manifest file: sulla base delle misurazioni fatte richiede il chunk di qualità massima sostenibile alla largezza di banda corrente. In questo modo può variare la qualità del video nel tempo, in modo da rispondere in maniera abbastanza elastica alle irregolarità della rete.

Questo richiede *"intelligenza"* dal client: questo deve sapere:

- *Quando* richiedere i chunk in modo da non incorrere né in starvation (restare senza chunk) né in overflow (non avere più spazio per i chunk);
- *A che bitrate* richiedere i chunk sulla base della banda disponibile;
- *Dove* richiedere i chunk: se i server sono duplicati dovrà scegliere il più efficiente.

1.2 CDN

Vediamo un'altra tecnologia, questa volta riguardante la *scalabilità* dei server che forniscono gigantesche quantità di contenuti video a numeri altrettanto grandi di utenti, simultaneamente, in tutto il mondo. Questo è il problema che viene affrontato dai **CDN** (*Content Distribution Network*).

L'idea è di copiare e servire diverse copie dei video su diversi server in siti geograficamente distribuiti. Esistono due approcci per la dislocazione di tali server:

- **Enter deep**: usare server CDN vicini agli access network: l'idea è di spostarsi il più vicino possibile agli utenti;
- **Bring home**: meno cluster, più grandi, distribuiti in **POP** (*Points Of Presence*) *vicini* ma non *sulla* rete di accesso degli utenti.

Facciamo un'esempio pratico, guardando al processo che un servizio di distribuzione segue per fornire un contenuto a un utente. Abbiamo che il meccanismo principale di redirezione è realizzato sfruttando il DNS:

- Il CDN (diciamo `kingcdn.com`) mantiene più copie del contenuto su diversi *nodi* CDN (magari `kingcdn.com/<slug-contenuto-1>`, `kingcdn.com/<slug-contenuto-2>`, ecc...).

- Il servizio di distribuzione rende disponibile il contenuto a `video.netcinema.com/<slug-contenuto>`;
- L'utente richiede un contenuto dal CDN, collegandosi a `netcinema.com` e navigando fino al contenuto richiesto a `video.netcinema.com/<slug-contenuto>` (l'opzione più probabile è che questo contenuto si trovi in un blocco HTML di tipo `<video>`);
- A questo punto, il DNS locale dell'utente chiede l'IP di `video.netcinema.com` ai server autoritativi di `netcinema.com`, che risponde con un record CNAME corrispondente all'IP del server autoritativo di `kingcdn.com`;
- L'utente viene quindi rediretto verso il nodo più vicino (potrebbe anche scegliere un'altro percorso se si verificassero variazioni (in peggio) del servizio). Da qui in poi il video è trasmesso dal nodo del CDN su HTTP.

1.3 Programmazione socket

Abbiamo introdotto il concetto di **socket** (dall'inglese per *presa*) come astrazione della comunicazione in rete per i processi in esecuzione su un S/O.

Lato pratico, il S/O renderà disponibili chiamate a sistema per effettuare almeno alcune operazioni base:

- Apertura di un socket: questo potrà essere **TCP** o **UDP**;
- Invio di dati al socket (o scrittura sul socket);
- Ricezione di dati al socket (o lettura dal socket).

Ricordiamo che il TCP è un protocollo orientato alla connessione, sicuro e affidabile, mentre UDP è un protocollo semplice per il trasferimento diretto ma non affidabile di dati.

1.4 Reti in connessione diretta

Finite le applicazioni, riprendiamo il discorso dal *basso*. Se avevamo discusso come creare applicazioni che sfruttassero la rete, adesso vogliamo discutere come la rete riesce in primo luogo a trasmettere bit fra più processi, o anche solo fra più calcolatori.

Iniziamo col considerare come nodi **adiacenti** nella rete comunicano fra di loro, cioè come 2 host con apposite interfacce di rete comunicano attraverso una connessione diretta: il caso più semplice è quello di un link punto-punto fisico.

Ci sono già diversi problemi da analizzare: come rendere la comunicazione **affidabile** (*framing*, *rilevamento errori* e *correzione*, ecc...). Introdurremo quindi il protocollo **PPP** (*Point to Point Protocol*), i protcolli ad **accesso multiplo**, e le **LAN** (*Local Area Network*).

1.5 Link punto-punto fisici

Supponiamo di avere 2 host (2 macchine) che vogliono comunicare fra di loro. Questi saranno provvisti ciascuno di un interfaccia per una porta seriale che potranno comandare (alto o basso), e le loro porte seriali saranno collegate da un link di qualche tipo (rame, fibra ottica, ecc...).

Le porte seriali permettono la **codifica** di bit attraverso variazioni di stato del mezzo fisico del link.

L'interfaccia della macchina mittente è detta **trasmettitore**, mentre l'interfaccia della macchina destinatario è detta **ricevitore**: bisognerà stabilire un protocollo per la comunicazione fra questi.

In questo caso il protocollo è semplice il *trasmettitore* dovrà occuparsi di **codificare** sequenze di bit in variazioni di stato sul mezzo di link, mentre il *ricevitore* dovrà occuparsi di **decodificare** tali variazioni di stato riportandole in bit sulla macchina destinatario.

1.5.1 Data link

Una caratteristica fondamentale dei link è che sono *inaffidabili*: il canale di trasmissione non è mai ideale (degradazione del segnale, rumore, interferenze, ecc....) e quindi la sequenza codificata potrebbe arrivare al ricevitore radicalmente cambiata rispetto a quella iniziale.

Chiameremo *bit error rate* la frequenza di bit persi sul mezzo di comunicazione.

Per gestire l'inaffidabilità introdurremo un livello superiore a quello di **link fisico**, detto di **link dati** (*data link* o anche *livello logico*).

- La prima cosa che prevede il data link è il **framing**: si divide la comunicazione in *datagrammi* (insiemi di bit), e si incapsulano i datagrammi fra *header* e *trailer* (sostanzialmente intestazioni e terminazioni del datagramma). Nell'**header** potremmo inserire diverse informazioni: prima fra tutte l'indice del datagramma nell'intero blocco di dati da trasferire.

  Se la frequenza di errore è comparabile con la lunghezza dei datagrammi, avremo sicuramente perdite di dati: riducendo la lunghezza dei datagrammi potremmo avere più fortuna.

  Questa tecnica introduce chiaramente un certo *overhead* di trasmissione, dato dal dover introdurre header e trailer. Ci permette però di dividere la trasmissione in unità contenute, su cui è più semplice fare controllo degli errori;

- Altri servizi sono forniti per il controllo degli errori di trasmissione: l'**error detection** mira a rilevare gli errori, mentre l'**error correction** mira a rilevarli e correggerli. Questo è piuttosto semplice nel caso di trasmissioni binarie: assunto di avere un rilevamento dell'errore granulare al bit, correggerlo significherà semplicemente invertirlo.

  Una soluzione alternativa è comunque quella della **ritrasmissione** dei dati in errore, che implica però una comunicazione all'*indietro*, dal ricevitore al trasmettitore per la segnalazione dei frame corrotti.

- Ci sono poi altri servizi che potremmo voler fornire: ad esempio il **controllo di flusso** che permetta di moderare in qualche modo la frequenza di trasmissione. Ad esempio il ricevitore potrebbe voler segnalare al trasmettitore di dover ridurre il bitrate, magari per problemi di overflow.

- Infine, vogliamo distinguere se offriamo servizi **half-duplex** o **full-duplex**:

  - **Half-duplex**: entrambi i nodi possono trasmettere, ma solo uno per volta;
  - **Full-duplex**: entrambi i nodi possono trasmettere, in *parallelo* (quindi contemporaneamente).

  Configurazioni dove solo un nodo può trasmettere sono dette **simplex**.

1.5.2 NIC

Il livello *data link* è implementato nella cosiddetta **NIC** (*Network Interface Card*) all'interno di ogni host sulla rete. Questa implementa il livello data link e di conseguenza il livello fisico. Può essere Ethernet, WiFi, ecc...

La NIC è collegata al bus periferiche di un calcolatore (ad esempio bus PCI) e implementa una o più delle funzionalità offerte dal livello data link. Fa questo combinando *hardware*, *software* e *firmware*: l'importante è che sia capace di offrire al calcolatore i datagrammi inviati al livello data link (in questo sia il NIC che il calcolatore vedono il livello data link).