1 Lezione del 26-09-25

1.1 Livelli di protocolli

Analizzando l'architettura di Internet abbiamo introdotto il concetto di **protocollo**. Studiando il *core* della rete abbiamo poi visto che la rete è molto complicata e costruita su più *astrazioni*: host inviano pacchetti di applicazioni attraverso router su vari link, ecc...

Possiamo decidere di organizzare i protocolli che governano tali operazioni attraverso una struttura a *layer* o **livelli**. Ogni livello implementa un dato **servizio**, utilizzando le sue azioni interne, e basandosi sui servizi offerti dal livello sottostante.

Questo procedimento non è fuori luogo in sistemi complessi composti da diversi componenti in relazione fra di loro: permette infatti di gestire ogni componente singolarmente, senza doverci preoccupare di rompere la compatibilità con gli altri componenti (basta basarsi sullo stesso servizio sottostante e offrire lo stesso servizio al livello superiore).

Lo stack di protocolli internet che studiamo è quindi il seguente, a 5 livelli:

1. **Application**: il livello che supporta le applicazioni in rete. Protocolli *IMAP*, *SMTP*, *HTTP*;
2. **Transport**: il livello che supporta il trasferimento dati da processo a processo assicurando sicurezza e astrazione sul modello a pacchetto. Protocolli *TCP* e *UDP*;
3. **Network**: il livello che implementa il trasferimento di *datagrammi* da sorgenti a destinazioni. Protocolli *IP*, *routing*;
4. **Link**: il livello che implementa la trasmissione di dati tra elementi di rete fra di loro "vicini". Protocolli *Ethernet*, *802.11 (WiFi)*, *PPP*;
5. **Physical**: il livello fisico rappresentato dai bit sul mezzo di comunicazione.

Un diverso stack è quello presentato dal modello di riferimento **ISO/OSI**. Questo è diverso dal modello presentato prima in quanto presenta qualche livello in più:

1. **Application**: come sopra;
2. **Presentation**: il livello che permette alle applicazioni di interpetare il significato dei dati. Protocolli di *compressione*, *crittografia*, ecc...;
3. **Session**: il livello che permette *sincronizzazione*, *checkpoint*, *recupero* di dati, ecc...;
4. **Transport**: come sopra;
5. **Network**: come sopra;
6. **Link**: come sopra;
7. **Physical**: come sopra;

Queste funzioni, se strettamente necessarie, dovranno essere implementate nel livello application.

1.1.1 Incapsulamento di protocolli

Adesso che abbiamo stabilito una gerarchia di protocolli che permettono il collegamento internet, possiamo definire in maniera più precisa cosa accade quando ci colleghiamo, ad esempio, ad un werver HTTP per richiedere una pagina Web.

Dal nostro calcolatore, e in particolare dall'applicazione *browser* in esecuzione sul nostro calcolatore, attraversiamo tutti i livelli (application, transport, network, link, e physical) per arrivare ad un router (probabilmente quello della rete di accesso). Man di mano che scendiamo in livelli più bassi corrediamo il messaggio inviato dall'applicazione con altre informazioni di controllo, utili ai livelli più bassi. A questo punto il router ottiene il messaggio attraverso, magari solo il livello link e physical: questo procedimento si ripete, finché non raggiungiamo il server. Quando il messaggio raggiunge la macchina server, risaliamo tutti i livelli (physical, link, network, transport, application) per arrivare all'applicazione server vera e propria, perdendo nel frattempo le informazioni aggiunte dai livelli sottostanti in fase di trasmissione. A questo punto il server può interpretare il messaggio ed eventualmente rispondere.

Con questa sezione abbiamo concluso l'introduzione al funzionamento (ad alto livello) di Internet, visto sia dall'*edge* della rete (cioè dal punto di vista degli *host*) che dal *core* della rete (cioè dal punto di vista dell'*infrastruttura*) di rete.

1.2 Applicazioni in rete

Veniamo quindi allo sviluppo di **applicazioni** in rete, con l'obiettivo di tornare alle specifiche delle reti in un secondo momento.

In questa sezione vedremo i principi delle applicazioni Web *client-server* e *peer-to-peer* (**P2P**), in particolare approfondendo i protocolli Web (HTTP), e-mail (SMTP, IMAP), il sistema DNS, nonché la programmazione con l'API dei *socket* coi protocolli TCP e UDP.

Nostro focus sarà quindi il livello applicazione (e in minor parte di trasporto), e gli aspetti concettuali e di implementazione di applicazioni in rete.

Creare un'applicazione in rete significa scrivere programmi che:

- Girano su diversi sistemi;
- Comunicano via la rete.

Le applicazioni vengono scritte per gli **host**: i router non eseguono applicazioni utente, e anzi *non eseguono* nemmeno lo stack protocollare completo (si limitano al livello link e al massimo network). Sviluppare applicazioni per i sistemi all'*edge* della rete permette invece la facile e rapida propagazione delle stesse.

1.2.1 Paradigma client-server

Il paradigma **client-server** è il più comune per le applicazioni in rete. In questo caso individuiamo due agenti principali:

- Il **server** è un host sempre attivo, con indirizzo IP permanente, solitamente distribuito su data center, per permettere scalabilità (qui si sfruttano tecnologie come *load balancer*, ecc...);
- Il **client** è un host che stabilisce contatto intermittente col sever, che può avere indirizzo IP variabile, e che non interagisce mai direttamente con altri client.

Abbiamo quindi che l'identità di client e server è *forte*, la relazione fra i due è asimmetrica e ben definita. In termini di risorse, il client non dovrà avere particolari risorse computazionali, mentre il server dovrà essere capace di gestire le richieste di tutti i client.

Esempi di protocolli client-server sono il protocollo HTTP, i protocolli di posta IMAP e il protocollo di trasferimento file FTP.

1.2.2 Paradigma peer-to-peer

Nel paradigma **peer-to-peer** non esiste un singolo server always-on, ma ogni peer può comportarsi in modalità intermittente sia da client che da server. Questo significa che i peer ricevono servizi da altri peer, fornendo in cambio altri servizi.

Questa caratteristica permettte l'*auto-scalabilità*: la rete P2P si sviluppa autonomamente man di mano che si aggiungono peer.

I peer hanno un identità molto più labile rispetto a quelle di client e server tradizionali: l'indirizzo IP può essere dinamico, non sono sempre online, il loro servizio potrebbe essere intermittente, ecc...

1.2.3 Socket

I **socket** sono l'API che implementa la connettività di rete per le applicazioni che scriveremo. Sono offerti dal sistema operativo e rappresentano un'astrazione per la connessione di rete (come i file rappresentano un'astrazione per il disco).

L'analogia tipica del socket è quella di una *porta*. I messaggi entrano dalla porta ed escono dalla porta: quello che sta dietro alla porta è parte dell'infrastruttura implementata prima dal sistema operativo (che implementa i livelli protocollari) e poi dalla rete Internet in sé per sé, ed è astratto via dal programmatore.