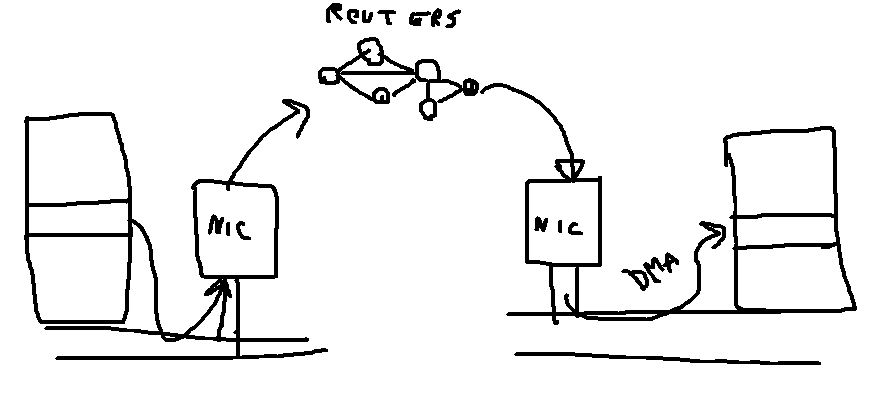
# RETI lezione 1

Il Sistema contiene i già visti dispositivi CPU, memoria RAM, Cache, ecc. e infine il dispositivo di interconnessione di rete (NIC).

Il NIC invia e riceve messaggi grazie a un trasferimento dma tra zone di memoria ram: i buffer di comunicazione. Quando si vuole inviare un messaggio le sequenze di byte vengono trasferite un byte alla volta all’interno della NIC per poi essere serializzate nel canale di trasmissione. Analogamente e in direzione inversa opera sempre il dma che copia nel buffer di ricezione i messaggi ricevuti dalla NIC.

La rete che permette alla NIC di fare ciò è costituita dai routers. Essi sono collegati tra loro e formano la rete. Al giorno d’oggi tali routers sono dei veri e propri computer costruiti al solo scopo di effettuare questi collegamenti: infatti i routers non hanno una sola unità NIC, ma ne hanno di più; cosa che permette loro di avere più collegamenti in contemporanea (si forma una “maglia” di collegamenti).

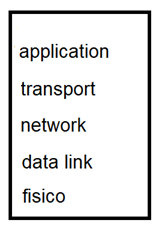


La Modalità di trasferimento viene chiamato Store & Forward: si riconduce alla comunicazione tra coppie di macchine. In un primo momento l’utente si connette al router a cui è fisicamente connesso e tutto il messaggio viene copiato nel buffer di memoria del primo router. Una volta ricevuto il messaggio, il router va a vedere a chi è destinato il messaggio: vi è quindi la necessità di attribuire degli indirizzi alle varie macchine connesse nella rete e tali indirizzi devono essere univoci. Dunque, una parte di bit del messaggio dovrà specificare l’indirizzo del destinatario; il router leggerà tali bit e deciderà il miglior percorso da seguire per trasferire i dati da un router all’altro fino al destinatario. Questi trasferimenti sono detti “Salti”.

Le telecomunicazioni possono avvenire tramite cavi o altri canali fisici (come per la fibra ottica) oppure attraverso la trasmissione via radio (e segnali elettromagnetici), come il wi-fi.

Introduciamo la Stratificazione in Livelli (Protocol Stack). Un modello teorico è il modello iso/osi, basato su una stratificazione in 7 livelli, ma che non è mai stata implementata nella pratica.

La stratificazione dei protocolli internet avviene quindi su 5 livelli. Ciascun livello si occupa di un aspetto particolare dell’implementazione dei protocolli di comunicazione.



Il livello più basso è il livello fisico (le implementazioni a livello hardware: come l’utilizzo di un particolare cavo elettrico e il livello di tensione necessaria per far funzionare i dispositivi) e permette di realizzare fisicamente il dispositivo. Il secondo livello è il Data Link, questo permette di specificare il formato dei bit che vengono trasferiti, indica come devono essere fatti i messaggi, di quanti byte possono essere composti, come viene controllato l’indirizzo del destinatario, come controllare eventuali errori nel messaggio ed eventuali specifiche nell’utilizzo del dispositivo. Fin qui stiamo ragionando a livello di un singolo canale di comunicazione (quindi di effettuazione di un singolo HOP). Al terzo livello abbiamo il livello di Rete (Network), esso consiste nell’instradamento del messaggio da sorgente a destinazione. È quello che dice ai router quale strada percorrere per portare un messaggio da una parte del mondo all’altra. Il quarto livello è il livello di Trasporto, mentre il quinto è quello Applicativo (Transport and Application).

A noi programmatori che vogliamo usare la rete interessa principalmente il livello Applicativo (poiché gli altri servono a realizzare la rete in sé). Il Livello di Trasporto, al contrario di come suona, permette di interfacciarsi col sistema operativo. Per capire perché sia così basti pensare che storicamente le reti di calcolatori e i sistemi operativi si sono evoluti separatamente (c’erano ingegneri che provavano a realizzare reti ma che non sapeva niente di computer e gli ingegneri di computer che non sapevano nulla di reti, l’unica eccezione era chi progettava la NIC).

Inizialmente c’era un supporto di telecomunicazione analogico (come la rete telefonica) e un livello di traduzione/codifica ulteriore per inviare anche informazioni digitali, ma che utilizzavano lo stesso mezzo di trasmissione (per cui si dovevano utilizzare frequenze diverse, infatti nei vecchi telefoni i numeri facevano rumori diversi per via della diversa frequenza a cui venivano codificati).

Al giorno d’oggi si fa la cosa opposta: è più efficiente trasformare i segnali analogici in digitali e trasmetterli così, i modem si sono quindi trasformati in routers, veri e propri computer. Le macchine usate dagli utenti invece hanno preso il nome id Host.   
Quello che cambia tra router e host è che i routers in genere possono non avere un’unità a disco, inoltre diverse sono le capacità del processore e la quantità di RAM, poiché il loro scopo è semplicemente trasferire informazioni.

Al giorno d’oggi non c’è separazione tra Reti e Sistemi Operativi/Computer, tuttavia, poiché i protocolli che ormai sono diventati standard (poiché per poter comunicare a livello mondiale è necessario che tutti usino lo stesso protocollo) sono stati iniziati ad essere standardizzati a partire dal 1960+ o -, quando ciò è avvenuto c’era ancora la separazione tra sistemi operativi e reti di calcolatori. Per poter standardizzare i protocolli di rete in maniera indipendente dalla struttura dei sistemi di calcolo, si è dunque inventato il quarto livello, il Transport. Infatti, in un computer normalmente non si vuole far girare solamente un’applicazione per volta, dunque il meccanismo di indirizzamento che viene usato a livello di rete per spostare le sequenze di byte da macchina sorgente a destinazione non era sufficiente (l’indirizzo indica solamente la macchina del destinatario, non l’applicazione in particolare a cui deve arrivare).

Si noti che le unità fondamentali nei sistemi di calcoli sono chiamati “Processi” e il livello di trasporto servirebbe, appunto, per mappare, attraverso il PID (Process ID) fornito dal sistema operativo, a quale Processo -> Applicazione debbano arrivare le informazioni. Ma c’è da considerare che non tutti i sistemi operativi sono uguali, non sempre c’è un PID, o se c’è può essere gestito in maniera diversa.   
Quindi quando si progetta il sistema di comunicazione è necessario non tanto identificare un programma quanto identificare la funzionalità di questo programma.

È così che viene introdotto il concetto di “Porta” (port) di comunicazione, anch’essa un intero su 16 bit come il PID. Le porte servono per mappare delle funzionalità attraverso dei programmi chiamati server. C’è un’organizzazione internazionale che ha stabilito una serie di numeri di porta che vanno associati a determinati server (ogni volta che se ne va aggiunto uno quindi bisogna chiedere a loro).

Il servizio di posta elettronica, per esempio, è stato uno dei primi servizi utilizzati in internet, dunque ha una porta molto piccola, il numero 25.

Quindi se si vuole mandare un messaggio di posta elettronica a un amico, bisogna prima specificare l’indirizzo ip del destinatario e poi indicare la porta 25. Un altro protocollo è il protocollo FTP (File Transfer Protocol). A seguito di un’autenticazione si può accedere al file system di un’altra persona ed effettuare il download di un file particolare sul proprio. L’FTP utilizza le porte 20 e 21.

(Per realizzare un protocollo di questo livello è necessario utilizzare i protocolli di livello più basso.)

Il vantaggio di utilizzare queste porte è che se c’è un server FTP attivo questo starà in ascolto sempre sulla porta 20, indipendentemente da quale PID gli è assegnato dal sistema operativo.

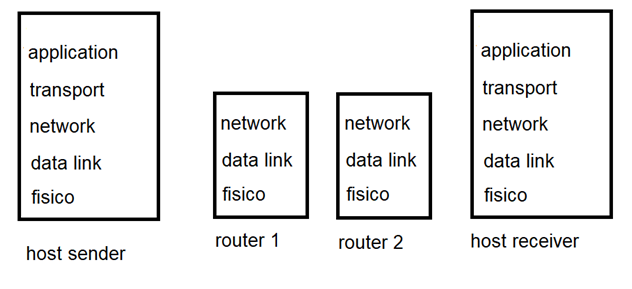
La funzione principale del livello di trasporto è quindi il multiplexing e il demultiplexing (ossia la virtualizzazione nella rete) dei processi che girano nella macchina. Infatti le porte predefinite possono essere usate in ricezione per i server quando vogliono ricevere un messaggio, ma quando vogliono inviare una risposta al client vengono utilizzate porte molto grandi (dette “effimere” / ephemeral). Un esempio si ha col protocollo http, che di base ascolta sulla porta 80.

(Si ricorda che mentre si possono avere più client, e quindi c’è bisogno di porte effimere, di server c’è n’è solo uno, dunque questo può utilizzare una porta standard)

Esempio: Il Browser di Alice vuole accedere al sito di Bob, per fare ciò il server di Bob si aspetta le richieste sulla porta 80, tuttavia il Browser di Alice dovrà ricevere una risposta per poter visualizzare la pagina. Dunque appena prima di inviare il messaggio il browser riceve assegnato un numero di porta effimero. Tale messaggio dovrà quindi contenere anche la porta effimera, così facendo quando poi il web server dovrà mandare il file di risposta basterà scambiare i dati tra sorgente e destinatario e il client otterrà le informazioni richieste.

Dunque i router dovranno fare l’operazione di multiplexing per permetter a più applicazioni di effettuare richieste sulla rete e l’operazione di demultiplexing per distribuire le risposte.

Nota: attualmente tutti i protocolli di rete prevede che i messaggi siano inviati con una parte iniziale (un header), una piccola parte finale e una grossa parte centrale chiamata payload, ossia il carico di lavoro. Le informazioni ausiliarie, come gli indirizzi, si trovano negli header.



Poiché un router si occupa solo dell’instradamento dell’informazione non c’è motivo per un router di implementare protocolli oltre il livello 3.

La comunicazione a pari livello (per esempio tra protocolli http mittente e destinatario) avviene attraverso un canale virtuale costruito da altri livelli di protocollo. I protocolli di trasporto, TCP e UDP, e quelli Applicativi (es http) parlano quindi direttamente tra mittente e destinatario. Tuttavia quando si scende a livello di network (quindi relativo all’IP) il pari livello si trova a distanza di un HOP (a distanza di un router), il Secondo HOP sarà tra il livello IP del primo router al livello IP del secondo router, fino ad arrivare alla macchina destinataria.

Poiché la comunicazione avviene solo a pari livello, quando si osserva la comunicazione tra due protocolli si possono ignorare tutti gli altri (quelli sottostanti per l’incapsulazione e quelli superiori perché non necessari per la comprensione della comunicazione).

Il concetto di incapsulamento: abbiamo un payload (contenuto), e il protocollo applicativo aggiunge il suo header: tale intestazione verrà letta e tolta dal livello 5 (applicativo) destinatario. Ai livelli inferiori abbiamo un comportamento analogo. La differenza è che al livello 4 il payload è sia il payload + l’header di livello 5. Il livello 4 quindi aggiunge l’header livello 4 (ad esempio l’header TCP) e tale header verrà interpretato dal protocollo 4 destinatario (per poter sapere qual è la porta destinataria). Tolto quello il payload viene recapitato all’applicazione che ha richiesto l’informazione ( e che costituisce al livello 5).

Al livello 3 succede la stessa cosa: il payload + header 5 + header 4 è trattato come payload e ci viene aggiunto un header di livello 3 (interpretato dai network).

Al livello 2 succede la stessa cosa che al livello 3, con la differenza che viene aggiunta anche una piccola parte finale, in cui di solito si trova una verifica di integrità del messaggio (correzione degli errori, o meglio scarto del messaggio in caso di errori). Un esempio di protocollo di livello 2 di questo tipo è l’Ethernet.

Dal livello 3 in poi l’aggiunta e rimozione dell’header avviene ad ogni Hop, poiché ogni router lo toglie e mette il proprio.

(Dal punto di vista del protocollo IP tutto quello che è contenuto sotto all’header di livello 3 (quindi anche header di livello 5 e livello 4) è considerato payload).

Non esiste un header di livello 1 perché a quel livello abbiamo i dispositivi fisici, per passare dal livello 2 all’1 non serve allungare il messaggio si passa direttamente alla codifica (tramite segnali elettrici).

Come può il programmatore usare queste conoscenze a suo vantaggio? Sappiamo innanzitutto che il nostro programma girerà sulla macchina sotto forma di processo. Dal punto di vista del sistema operativo il nostro processo è qualcosa di chiuso in se stesso, costituito da un codice e da i vari segmenti dati (statici/stack/heap). È chiuso in se stesso perché io posso inviare tanti processi sulla stessa macchina e questi possono non interagire tra loro: ognuno di essi ha la propria macchina virtuale.  
Tuttavia ciò è un po’ limitante, perché a noi serve poter anche interagire con il mondo esterno, ad esempio tramite l’interazione coi dispositivi di input output. Nei dispositivi POSIX ciò avviene tramite l’utilizzo dei File Descriptor (i predefiniti per i programmi in esecutioni sono stdin, input da tastiera, stdout, stderr, output sul terminale per stampa normale e segnalazioni di errore). Tipicamente input e output sono bufferizzati (tranne stderr). Questi file descriptor sono come dei buchi nel processo chiuso per fare entrare e uscire informazioni.   
Se noi vogliamo aggiungere una primitiva di comunicazione che ci permetta di scrivere o ricevere messaggi attraverso la rete possiamo pensare di fare la stessa cosa, e di usare quindi sempre il concetto di file descriptor. Infatti mentre quelli di default sono tipicamente visti come dei file (e sono sempre presenti grazie al sistema operativo) per poter accedere a un file vero e proprio è necessario chiamare una system call OPEN per poter usare un nuovo file descriptor (quello del file che ci serve modificare o leggere). La OPEN in particolare prende come parametro il nome del file e restituisce il file descriptor come un intero, che però rappresenta un buco nel processo (stdin, stout e stderr hanno di base 0, 1, 2 assegnatoli).

Per poter leggere e scrivere basta usare le call READ e WRITE col File descriptor.

Quando si chiude un file con la CLOSE il buco viene “tappato”.

Per la comunicazione di rete si fa in maniera analoga, ma anziché OPEN si usa SOCKET: questa system call restituisce un file descriptor che rappresenta un buco nel processo perché possa interagire con il protocollo di livello 5 della NIC. (Il SOCKET è una virtualizzazione della comunicazione via rete attraverso la NIC).