Reti di Elaboratori

Davide Pucci & Marco Panunzio

2016

Indice

Capitolo 1

Introduzione

1.1 Note del corso

Il corso di Reti di Elaboratori segue i capitoli del Kurose-Ross. I tre macro-argomenti sono:

1. Fondamenti di fisica
2. TCP/IP
3. Reti wireless, sicurezza delle reti, e così via ...

1.2 Componenti della rete Internet

Centinaia di milioni di dispositivi (**hosts** o endsystems) che ospitano **network apps** sono interconnessi insieme ai router tramite *connection links*.

- *Hosts* (o *End-Systems*), sui quali girano applicazioni web. *End Systems* perché sono le foglie di un albero di collegamenti che passano per diversi *router* (i quali permettono lo scambio di informazioni tra i vari hosts);
- *Communication Links* (fibra, radio, satellite, e così via ...), a cui è associato un determinato insieme di frequenze, detto *banda*. Conoscendo la banda è possibile ricavare il quantitativo di bit massimo che posso far attraversare, il *datarate* (o *tasso di trasmissione*). La *collisione* è la sovrapposizione di pacchetti di comunicazione da sorgenti diverse, e causa la mancata decodifica dei pacchetti da parte del router;
- *router* è l'elemento cui viene affidato il compito di gestire l'inoltro dei pacchetti dalla sorgente alla destinazione. È formato da link di ingresso, sui quali esegue il *forwarding* verso i link d'uscita. Per poterlo fare, necessita di un'informazione, l'indirizzo del destinatario. Controlla quindi una *tabella d'inoltro*, composta da due colonne: l'indirizzo del destinatario e *next*

hop. Seguendo ogniqualvolta necessario i riferimenti associati all'indirizzo richiesto sulla tabella, il pacchetto arriva a destinazione;

- *Protocolli* implementano funzionalità necessarie per la rete; controllano la ricezione e la trasmissione di messaggi. Alcuni esempi di protocolli sono TCP, IP, HTTP, ETHERNET e SKYPE.
- *Standard*: definiscono le regole attraverso cui vengono applicati tutti i protocolli in *internet*.
  - L'*RFC* (*request for comments*) contiene tutti gli standard pubblicamente riconosciuti e in vigore su internet.
  - Lo *IETF* (*internet engineering task force*) viene consultato ogniqualvolta giunge una richiesta di inclusione di un nuovo standard. Dopo essere stato sufficientemente vagliato, lo standard può essere incluso tra gli RFC.

1.3 Rete Logica e Rete Fisica

- Una **Rete Fisica** è il diagramma che costituisce l'assetto fisico di una rete. Contiene una serie di nodi - ciascun elemento di rete - legati tra loro da un mezzo trasmissivo;
- Una **Rete Logica** è il diagramma che costituisce l'assetto logico di una rete. Partendo da quello fisico, in questo caso, per ogni nodo possono essere applicate delle policies che permettono - o meno - la trasmissione di dati attraverso mezzi trasmissivi, verso un dato elemento di rete ad esso fisicamente collegato.

1.4 Internet: Servizi di comunicazione

I protocolli definiscono come avvengono i colloqui tra dispositivi attraverso specifici set di regole.

1.4.1 TCP

Il *Transmission Control Protocol* è un protocollo di rete che si occupa di rendere affidabile la trasmissione dati in rete tra mittente e destinatario. In particolare, questo protocollo fornisce:

- trasferimento dati ordinato e affidabile: permette riconoscimento e ritrasmissione di eventuali perdite di dati
- controllo del flusso: il mittente non sovraccaricherà il ricevente
- controllo della congestione: il mittente rallenterà la velocità di invio quando la rete è congestionata

Handshake

Si definisce *handshake* (lett. *stretta di mano*) il processo attraverso il quale due calcolatori stabiliscono le regole comuni, ovvero la velocità, i protocolli di compressione, di criptazione, di controllo degli errori etc.

Prima di iniziare la connessione vera e propria tra due macchine si crea questo tipo di connessione che consiste nella trasmissione dei pacchetti per regolare i parametri di connessione.

1.4.2 UDP

Lo *User Datagram Protocol*, a differenza del protocollo TCP non esegue alcun tipo di *handshake* o controllo: questo è utile (o necessario) per alcuni tipi di applicativi che fanno loro componente fondamentale la rapidità di trasmissione di dati, come i servizi di streaming o voip, per i quali diventa di fondamentale importanza la rapidità di ricezione dei pacchetti, piuttosto che la loro qualità.

1.5 Trasmissione Dati

- **Commutazione di Circuito**: si stabilisce un circuito (percorso) tra mittente e destinatario, soltanto dopo aver appurato che le risorse necessarie per effettuare l'effettiva trasmissione dei dati siano disponibili. Qualora ci fossero diversi utenti, la suddivisione delle risorse potrebbe avvenire attraverso due modalità:
  - **FDM**: *Frequence Division Multiplexing*, che consiste nel suddividere le risorse in canali, uno per utente. In questo modo ogni utente avrà 1/x delle risorse totali a propria disposizione, dove x è il numero di utenti che utilizza la rete;
  - **TDM**: *Time Division Multiplexing*, che consiste nel suddividere le risorse in base al tempo. Ogni utente utilizza la totalità delle risorse disponibili per un determinato lasso di tempo, a conclusione del quale l'utilizzo delle risorse passa all'utente successivo fino a ricominciare il ciclo con il primo utente.

  ESEMPIO. Vogliamo ricavarci il tempo necessario per trasmettere 640.000 bit da un host A ad un host B, sapendo che la rete ha una banda di 1,536Mbps.

$$1,536/24 = 64K \rightarrow 64.000/640.000 = 1/10s \tag{1.1}$$

  ESEMPIO. Trasmettiamo 7,5Mbit con una banda massima di 1,5Mbps.

$$3 * 7,5/1,5 = 3 * 5 = 15s \tag{1.2}$$

- **Commutazione di Pacchetto**: ogni informazione da trasmettere viene suddivisa in n pacchetti. Ogni host condivide in questo caso le stesse risorse di rete, ogni pacchetto utilizza tutta la banda disponibile e le risorse vengono adoperate soltanto quando necessario. Solitamente le code di pacchetti sono gestite in ordine di arrivo.

  ESEMPIO. Trasmettiamo 7,5Mbit con una banda massima di 1,5Mbps.

  1. Dividiamo l'informazione da trasmettere in 5000 pacchetti da 1500 bit;
  2. Occorre quindi 1ms per trasmettere pacchetti su un link
  3. Ogni link lavora in parallelo (*pipelining*)
  4. Il delay è ridotto da 15s (nel caso dello stesso esempio applicato in un paradigma a commutazione di circuito) a 5,003s.

Il paradigma a commutazione di pacchetto permette a più utenti di utilizzare - in tutta la sua banda e potenza - la stessa rete.

1.6 Tipi di Sorgenti

Le sorgenti di trasmissione dei dati possono utilizzare tassi di frequenza di trasmissione diversi:

- **Tasso costante**: nel caso della telefonia, per esempio, i pacchetti hanno una dimensione fissa e vengono trasmessi ad intervalli regolari. La frequenza è di 64Kbps;
- **Tasso variabile**: facendo riferimento all'esempio precedente, alcuni software gestiscono in maniera ottimale la trasmissione dei dati: associazione ad una variabile il rumore di sottofondo. Ogniqualvolta riconosceranno - utilizzando quella variabile - che l'utente non sta parlando, non invieranno alcun pacchetto.

1.7 Network Edge

Come già anticipato, ogni *end-system* (*host*) esegue applicativi. Questo può avvenire secondo diverse modalità:

- **Client-Server**: un host (*server*) esegue applicativi adibiti alla manipolazione di informazioni, ma soprattutto allo scambio di informazioni con altri host (*client*). Quindi, essenzialmente, i *client* effettuano richieste ai *server*, che eseguono servizi perennemente in esecuzione (*always-on*);
- **Peer-Peer**: non esiste differenziazione tra *server* e *client*, perché in questo caso gli host eseguono le stesse attività, ma comunicando tra loro (come nei casi di servizi come *Torrent*, *Skype*, e così via ...).

1.8 Accesso alla rete

Ogni dispositivo può essere connesso ad internet tramite diverse *reti di accesso*. Possono essere:

- Reti residenziali
- Reti istituzionali (scuole, aziende, e così via ...)
- Reti mobili (come quelle utilizzate dagli smartphones)

Elementi essenziali della rete di accesso sono la banda (bit/s), l'affidabilità (bit error rates) e se è dedicata o condivisa.

Punti di accesso mobili Un altro elemento fondamentale è il punto di accesso alla rete: essa dev'essere in grado di riconoscere che si sta degradando il segnale e individuare un altro punto di accesso al quale ci stiamo avvicinando, così da evitare un'eventuale degradazione della qualità della rete quando si è in movimento.

1.9 Trasmissione attraverso link fisici

Ciò che viene trasmesso è una sequenza di bit che codifica il contenuto.

- **Bit**: valori che si propagano tra il trasmettitore e il ricevitore
- **Link**: tracce attraverso cui passano i bit
  - **Tracce guidate**: cavi fisici (fibra, e così via ...)
  - **Tracce radio**: le informazioni vengono trasmesse attraverso onde radio

Le informazioni trasmesse tramite onde radio subiscono un'*attenuazione* durante la propagazione dell'informazione, e quindi una *distorsione*. Ciò non avviene in maniera uniforme; bisogna anche mettere in conto eventuali interferenze elettromagnetiche (spesso introdotte dalle stesse apparecchiature utilizzate per la connessione). Il risultato è che la sequenza ricevuta potrebbe non essere corretta. Questo introduce la definizione di *bit error rate*, ovvero la frequenza con cui si verifica un errore nella trasmissione delle informazioni. Minore è il suo valore, migliore e più stabile è la rete. In questo caso, occorre che il dispositivo sia *error-aware*, così da essere autonomo nell'individuare la presenza di errori e tentare di correggerli (possibile fino a un certo numero di bit, altrimenti viene scartato il pacchetto e richiesto indietro).

1.9.1 Codifica NRZ

NRZ: Non Return to Zero. Il tempo è diviso in slot - unità temporali che corrispondono ad un bit. Ogni bit ha associato un valore stabile per la sua intera durata (1: High, 0: Low). Il problema in questo tipo di codifica è la spesso asicronizzazione tra trasmettitore e ricevitre.

1.9.2 Codifica Manchester

Utilizzata in Ethernet. Essenzialmente viene associato un valore ad ogni transizione. Quando la frequenza di trasmissione va dal basso all'alto - di intensità - il valore associato è 1, dall'alto al basso, invece, 0.

1.10 Tecnologie di accesso

1.10.1 Dial-up modem

Velocità trasmissiva: 56Kbps (ca.). Utilizza la stessa infrastruttura telefonica: non *always-on*, impossibile poter usufruire della linea per internet e telefonia allo stesso tempo. Il modem reindirizza le richieste al *central office*, che - qualora si cerchi di navigare in internet - converte la richiesta e la reindirizza all'*ISP* (*Internet Service Provider*).

1.10.2 Digital Subscriber Line

Anch'essa utilizza un'infrastruttura telefonica esistente. Spesso chiamata anche A-DSL (la *A* indica l'*asimmetria* tra la velocità di trasmissione modem-*central office* e *central office*-modem). Essendo la frequenza del segnale vocale sempre in un intervallo tra 0 e 4KHz, il segnale può essere campionato fino a 8M volt/s ed essere successivamente ricostruito; in quella frequenza il segnale è trasmesso *solo* attraverso via telefonica. Tutte le trasmissioni fuori da quell'intervallo vengono quindi campionate come richieste per internet. Le risorse del canale sono divise attraverso *FDM* (*Frequency Division Multiplexing*).
Velocità: fino a 1Mbps up (tipicamente $<$ 256Kbps) e a 8Mbps down (tipicamente $<$ 1Mbps) down. La grande differenza di velocità trasmissiva è causata da un paradigma client/server che dà la precedenza al download dei dati piuttosto che alla loro trasmissione.
Tecnologie di tipo DSL sono progettate per raggiungere distanze non molto elevate (fino a 4/5 km).

ADSL loop extender Dispositivo posizionato tra il cliente e il central office dalla compagnia telefonica per estendere la distanza ed aumentare la velocità di trasmissione.

1.10.3 HFC (in *USA*)

Velocità: fino a 20Mbps down, 2Mbps up. Utilizza - invece dell'infrastruttura telefonica - quella della TV via cavo. Sostanzialmente il *central office*, in questo caso il *cable headend*, gestisce degli anelli di fibra, composti da moltissimi nodi che li collegano ad ogni abitazione. A questo punto, la connessione passa per uno splitter (che utilizza l'*FDM*), che la indirizza alla TV o al dispositivo che richiede la connessione internet.

1.10.4 Fibra ottica

Collegamenti ottici dal *central office* all'abitazione. Raggiunge distanze lunghissime praticamente senza alcuna distorsione del segnale. Due tipologie di accesso:

- *Passive Optical Network* (*PON*): un'unica fibra e uno splitter ottico che smista le informazioni verso diverse abitazioni. Comporta costi decisamente bassi, a scompenso però della qualità:
  - Si possono spesso verificare *collisioni*, evitate attraverso il *TDM*
  - Essendo una linea condivisa si presenta la necessità di dover criptare i messaggi
  - Non raggiunge distanze massime
  - Diventa più difficile isolare eventuali problemi
  - Velocità di accesso non ottimale
- *Active Optical Network* (*PAN*)

1.10.5 Ethernet

Evoluzione che risale agli anni '80; tecnologia basata su formati e protocolli che sono rimasti immutati, se non per supportare il decisivo incremento della velocità di navigazione. Uno switch - con ingressi ethernet, appunto - viene frapposto tra router/modem e diversi host.

1.10.6 Wireless Access Networks

In questo caso, le informazioni sono modulate opportunamente, per raggiungere le frequenze delle onde radio supportate dal ricetrasmettitore radio. Ne esistono due tipi:

1. *Wireless LAN* (o *WiFi*) 802.11b/g: raggiunge una velocità compresa tra gli 11Mbps e i 54Mbps;
2. *Wide-area Wireless Access*: mezzo radio condiviso che raggiunge una velocità compresa tra 1Mbps (*EVDO/HSDPA*) e diverse decine di Mbps (*LTS*).

Essendo la rete condivisa, neccessita di essere criptata. Non utilizza né *FDM* né *TDP*, ma un sistema di gestione proprietario e specifico.

1.11 Reti Mobili

Generazione	**Simbolgia**	**Tecnologia**	**DL Massima** in Mbit/s	**DL Tipica** in Mbit/s
2G	G	GPRS	0,1	0,03
2G	E	EDGE	0,3	0,1
3G	3G	3G (basic)	0,3	0,1
3G	H	HSPA	7,2	1,5
3G	H+	HSPA+	21	1
3G	H+	DC-HSPA+	42	8
4G	4G	LTE	100	15

1.12 Componenti attuali

1. *DSL Fast-Technology* (combinata con fibra): 1Gbps
2. *Ethernet*: 25-40Gbps (con più linee 100-400Gbps)
3. *WiFi IEEE 802.11ac*: 1Gbps
4. *Fibra*: diverse decine di Tbps
5. *Reti mobili*: diverse centinaia di Mbps

1.13 Internet Structure: rete di reti

Internet ha una struttura gerarchica, composta da moltissimi *ISP* (*Internet Service Provider*). La prima categoria è composta dai *Tier-1* (Verizon, Sprint, AT&T, e così via ...). Questi sono interconnessi - per maggiore copertura - tramite dei *POP* (*Point of presence*). Poi ci sono i *Tier-2*, anche loro interconnessi tra loro e con i *Tier-1*. La gerarchia continua seguendo questo schema, fino a raggiungere gli *ISP* regionali/pronvinciali e, infine, le reti residenziali.

1.14 Loss & Delay

Esistono diversi tipi di ritardi:

- **Processing Delay**: ritardo nel processare i dati ricevuti (per analizzare se contengono errori o meno, per leggere l'header e per instradarli nella coda giusta).
- **Queue Delay**: tempo di attesa prima che il link adibito alla consegna dei dati processi il pacchetto.
- **Transmission Delay**: il tempo necessario a trasmettere tutta la sequenza del pacchetto.
  - R = banda link (bps)
  - L = grandezza pacchetto (bit)
  - a = frequenza di arrivo pacchetti
  - *Tempo di trasmissione sul link*: $\frac{L}{R}$
  - *Intensità di traffico*: $\frac{L \times a}{R}$

  L'arrivo dei pacchetti segue una distribuzione di *Poisson*, e quindi:
  - se l'intensità di traffico è 0 ca., si è verificato un delay piccolo;
  - se l'intensità di traffico è 0,5 ca., si è verificato un delay medio;
  - se l'intensità di traffico è 1 ca., si è verificato un delay alto e basta pochissimo per congestionare la rete.
- **Propagation Delay**: tempo utilizzato dai bit per propagarsi (alla velocità della luce, ca.) lungo tutta la linea fino alla destinazione.
  - d = lunghezza fisica del link
  - s = tempo di propagazione in media
  - *Ritardo di propagazione* = $\frac{d}{s}$
- **Nodal delay** = Processing delay + Queue delay + Transmission delay + Propagation delay

Il *trace route program* conta il tempo mandando pacchetti con un contatore *time-to-live* che decrementa di 1 all'incrontro con ogni router.
Il *throughput* indica i bit trasmessi al secondo: il collo di bottiglia sarà il link che avrà il *throughput* minore. Viene quindi inviata la massima quantità pari al *throughput* del link più piccolo.

Capitolo 2

TCP/IP

2.1 Protocolli

- **TCP**: protocollo di trasporto
- **IP**: protocollo di instradamento

In una *TCP connection request/response* i protocolli devono specificare:

- il formato delle informazioni (ordine dei bit, e quant'altro . . .)
- messaggio da scambiare
- ordine degli eventi

TCP prevede che prima dello scambio delle informazioni ci sia la *request/response*, un dialogo il cui fine è assicurare che:

- i dati siano in ordine e completi
- avvenga il *controllo di flusso* (consente di sapere la velocità e la quantità di dati che è possibile trasferire)
- avvenga il *controllo di congestione* (i router si accorgono se il flusso in arrivo è maggiore di quello in uscita)

A differenza del *TCP*, l'*UDP* non prevede alcuno scambio *request/response*, prima dell'effettivo trasferimento dell'informazione. Il sistema dei *DNS* utilizza *UDP*: quando si ha un *URL*, bisogna ottenerne l'*indirizzo IP* associato, perciò viene richiesta l'informazione tramite il protocollo *UDP*, che garantisce una trasmissione rapida, ma non da certezze che questa vada a buon fine.

2.2 Pila TCP/IP

La *pila TCP/IP* è una pila protocollare, che indica i livelli di protocollo, organizzata a livelli (modularità, così che se variano le regole di un livello, questo cambiamento non tange il funzionamento degli altri). Tutte le funzionalità logiche si dividono in gruppi, per rendere la progettazione più semplice e più estensibile.
Ciascuna funzionalità ha diversi protocolli. Il gruppo più in alto corrisponde alla logica applicativa. Spesso si arricchisce ogni gruppo con microfunzionalità e questo da un servizio aggiuntivo al livello superiore, e queste microfunzionalità possono essere mappate (duplicate) in livelli diversi. Diverse entità della rete hanno diversi sottoinsiemi delle funzionalità. Vengono stabiliti gli *access service point* che sono i punti in cui - nell'architettura - c'è il passaggio da un livello ad un altro. Lo svantaggio di un'architettura di questo tipo è che livelli non adiacenti non hanno la possibilità di comunicare (esiste però un sistema, chiamato *cross layering*, che permette di leggere dati di livelli non adiacenti).
I livelli sono solitamente i seguenti:

1. **Lv "applicativo"** (*software*): molte funzionalità diverse (come la sincronia);
2. **Lv di "trasporto"** (*software*): utilizza di base il protocollo *UDP* e controlla che il messaggio venga consegnato al giusto destinatario. Col protocollo *TCP* c'è anche il controllo se il pacchetto è arrivato ed è stato realmente consegnato al giusto destinatario;
3. **Lv di "rete"** (*network*): adibito al tentativo di capire il percorso da intraprendere, attraverso algoritmi di instradamenti, per poi inviare il pacchetto attraverso quel percorso;
4. **Lv "link"**, o **datalink** (*hardware*): lavora con i *frame* (*trama*). Verifica se ci sono stati errori di trasmissione, analizzando anche gli indirizzi *MAC* (*Medium Access Control*);

Sugli *end-systems* ci saranno tutti e 4 i livelli. Sugli elementi intermedi di commutazione ci saranno solamente i 3 livelli più bassi. L'*ack* (*acknowledgement*) è un tipo di messaggio che viene utilizzato per far capire al sorgente/destinatario che il pacchetto è arrivato. Ogni livello aggiunge un *header*, che per il livello successivo diventa parte integrante del messaggio stesso da consegnare. Per ciascun livello, ecco l'header:

- Applicativo: il messaggio stesso (*messaggio*, o *message*);
- Trasporto: il *segmento*, o *segment*;
- Rete: il *pacchetto*, o *datagram*;
- Datalink: la *trama*, o *frame*.

I vantaggi della stratificazione sono la modularità e la gestione dell'eterogeneità.

2.3 Storia

Internet è stato inventato da *Leonard Kleinrock*, quand'era ancora studente ed ebbe l'idea dell'architettura a pacchetto, nel 1961. Nel 1969, 8 anni dopo aver concepito questa idea, viene attivata la prima rete con 4 nodi (le 4 importanti università della California), utilizzando il protocollo *NCP*.
Nel 1972 viene allargata a 15 nodi. Negli anni successivi altri studenti hanno avuto l'idea di una rete *wireless*, utilzzando il protocollo *ALOHA*, o anche quella di connettere stampanti - o altri dispositivi -, concependo per la prima volta il concetto di *LAN*.
Robert E. Kahn e *Vinton Cerf* capirono l'importanza di mantenere l'eterogeneità della rete. Definirono l'architettura di internet odierna.
Così:

- 1979: raggiungimento di 200 nodi;
- 1982: nasce l'*SMTP* per le email;
- 1983: inventato il *TCP/IP*;
- 1985: nasce l'*FTP* per lo scambio di file;
- 1988: viene aggiunto il controllo di congestione al protocollo TCP.

Ciononostante, la rete rimaneva una realtà legata all'ambito accademico.
Nei primi anni '90, *Tim Berners-Lee* ebbe l'idea di web così come oggi è conosciuto, inventando l'*HTTP* e l'*HTML*, sulla base di documenti del 1945 di *Vannevar Bush* che accennava agli *ipertesti*.
Da qui in poi internet ha avuto una crescita esponenziale, sia in velocità che in applicazioni.

Capitolo 3

Livello Applicativo

3.1 Accenni

Il *Protocollo Applicativo* della pila *TCP/IP* definisce:

- i tipi di messaggi scambiati;
- la sintassi del messaggio;
- la sementica dei campi;
- le regole che stabiliscono quando inviare richieste/risposte.

3.2 HTTP

Prima di introdurre il concetto di *HTTP* occorre spiegare il paradigma *client-server*:

- *Client*: processo client (come un browser, quando richiede una *URL - Uniform Resource Locator*) che può fare richieste ed interpretare le risposte;
- *Server*: processo server, che fornisce le risposte.

Il *client* viene attivato solo quando necessario, mentre invece il *server* è sempre in ascolto. Un *IP* identifica univocamente un host nella rete (in realtà una scheda di rete). Attraverso l'indirizzo *IP* si può raggiungere l'host ma non il processo cui occorre effettivamente consegnare il messaggio. Per questo, vengono utilizzate porte differenti.
Il binomio *IP-porta* permette a due processi su due host diversi di comunicare. *IANA* è l'*ente* che definisce l'uso delle porte: generalmente, le porte 0-1023 sono riservate, mentre le successive sono per le *user applications*.
Sostanzialmente il processo che è in ascolto di messaggi apre un *socket* sulla porta specifica e rimane in attesa di connessioni dall'esterno su quella porta e quel socket (per ogni connessione entrante, nell'header del livello di trasporto

viene specificata la porta di destinazione, mentre invece l'IP e l'ID nel livello di rete).
Quando viene richiesta una pagina *HTML*, la reazione del protocollo *HTTP* varia in base alla versione:

- *HTTP/1.0* faceva tante connessioni quanti erano gli oggetti (caso non persistente) in canali dedicati;
- *HTTP/1.1* fa invece una sola connessione che rimane aperta e richiede tutti gli oggetti tramite questa sola connessione.

Si definisce *RTT* (*Round Trip Time*), il tempo che un pacchetto impiega ad arrivare al server e tornare indietro. Per *HTTP* esistono quattro metodi essenziali:

1. *GET*: richiede una pagina
2. *HEAD*: con lo stesso principio del *GET*, riceve soltanto l'header della pagina HTML (utilizzato per motivi di debug)
3. *POST*: invia dati al server tramite form
4. *PUT*: carica un file sul server

Esistono poi quattro tipi di header:

1. generali
2. *request* header
3. *response* header
4. *entity* header

Un campo specifico da sottolineare è quello dell'*if-modified-since*, che fa parte dei *request* headers: il client specifica una data e richiede al server la risorsa solo se questa è stata modificata dopo quella data.
Tra il client e il server ci sono degli intermediari chiamati *proxy-servers*, che mantengono risorse in cache e forniscono le risorse al posto del server interessato. Sostanzialmente, sono delle macchine che lavorano a livello applicativo, con grandi quantità di memoria; leggono le richieste dei client e se possono servire la richiesta lo fanno al posto del server. Chiaramente, sono in costante aggiornamento così da evitare problemi di consistenza.
Esistono diversi tipi di *proxy*:

- *caching proxy* (più vicino al client);
- *forwarding proxy*:
  - *trasparent proxy*;
  - *non-trasparent proxy.*

- *reverse proxy* (più vicino al server, intercetta le richieste e le invia ai server - in maniera bilanciata -; fanno spesso un lavoro di criptaggio dei messagi).

Utilizzando il parametro *no-cache* si specifica il desiderio di non passare per il sistema di *caching* dei *proxy-servers*.
Chiaramente tutti i sistemi di *caching* utilizzati seguono una gerarchia, a capo della quale vi è quella locale del proprio browser.

3.3 Cookies

I *cookies* rappresentano una sorta di gettone identificativo, usato dai server web per poter riconoscere i browser durante comunicazioni con il protocollo HTTP usato per la navigazione web. Tale riconoscimento permette di realizzare meccanismi di autenticazione, di memorizzare dati utili alla sessione di navigazione - come le preferenze sull'aspetto grafico o linguistico del sito -, di associare dati memorizzati dal server - ad esempio il contenuto del carrello di un negozio elettronico -, di tracciare la navigazione dell'utente - ad esempio per fini statistici o pubblicitari.
Date le implicazioni per la riservatezza dei naviganti del web, l'uso dei *cookies* è categorizzato e disciplinato negli ordinamenti giuridici di numerosi paesi, tra cui quelli europei, inclusa l'Italia.
Nel *cookie* solitamente possiamo trovare quattro attributi:

- Nome/valore è una variabile ed un campo obbligatorio;
- Scadenza (*expiration date*) è un attributo opzionale che permette di stabilire la data di scadenza del *cookie*. Può essere espressa come data, come numero massimo di giorni oppure come *Now* (adesso) (implica che il *cookie* viene eliminato subito dal computer dell'utente in quanto scade nel momento in cui viene creato) o *Never* (mai) (implica che il *cookie* non è soggetto a scadenza e questi sono denominati persistenti);
- Modalità d'accesso (*HttpOnly*) rende il *cookie* invisibile a *javascript* e altri linguaggi *client-side* presenti nella pagina;
- Sicuro (*secure*) indica se il *cookie* debba essere trasmesso criptato con *HTTPS*.

Il dominio (*domain*) e il percorso (*path*) definiscono l'ambito di visibilità del *cookie*, indicano al browser che il *cookie* può essere inviato al server solo per il dominio e il percorso indicati. Se non specificati, come predefiniti prendono il valore del dominio e del percorso che li ha inizialmente richiesti.

Un esempio di *cookie* è il seguente:

```
Set-Cookie: LSID=DQAAAK...Eaem_vYg; Domain=docs.foo.com; Path=/
    accounts; Expires=Wed, 13-Jan-20021 22:23:01 GMT; Secure;
    HttpOnly
Set-Cookie: HSID=AYQEVn...DKrdst; Domain=.foo.com; Path=/; Expires=
    Wed, 13-Jan-20021 22:23:01 GMT; HttpOnly
Set-Cookie: SSID=Ap4P...GTEq; Domain=.foo.com; Path=/; Expires=Wed,
     13-Jan-20021 22:23:01 GMT; Secure; HttpOnly
```

3.4 Domain Name System

Il **DNS** identifica sia il *database distribuito*, che contiene l'associazione tra nome e IP, sia il protocollo applicativo per l'effettiva risoluzione *nome-IP* (o *IP-nome*). I suoi principali servizi sono:

- traduzione tra *hostname* e *IP*;
- gestione di aliasing degli host;
- gestione di aliasing dei mail server;
- distribuzione del carico, o *load balancing* (per evitare di sovraccaricare il server, tramite l'associazione di più IP ad un unico nome).

È strutturato come database distribuito per le seguenti ragioni:

- possibilità di isolamento di eventuali problemi;
- gestione del traffico;
- gestione delle distanze;
- manutenzione.

Da qui è opportuno definire il concetto di **scalabilità**: capacità di un sistema di mantenere un comportamento ottimale alla crescita degli utenti che ne fanno uso.
La distribuzione dei DNS avviene in maniera gerarchica:

1. **Root** DNS Servers;
2. **TLD** (*Top-level domain*) DNS Servers (ciascuno per gruppi di *TLD* diversi: .com, .org, .edu, e così via ...);
3. **Authoritative** DNS Servers (per .com vi potrebbero essere amazon.com e yahoo.com, per .edu invece poly.edu, per esempio; e così via ...);
4. **Local** DNS Servers.

3.4.1 Risoluzione del nome

La risoluzione del nome avviene leggendo il dominio da destra (ogni dominio dopo il *TLD* ha un punto). Nel caso di ebay.it(.), per esempio: si legge da destra fino al primo ".". Nel primo passaggio la stringa ottenuta sarà vuota, per questo si interrogherà il Root DNS Server circa la stringa successiva, ovvero da dove si è arrivati fino al "." successivo: si è ottenuta la stringa ".it". Una volta ottenuta risposta dal Root DNS, sapremo chi è delegato dei domini associati al *TLD* .it. A questo punto, continuando con lo stesso criterio, chiederemo al DNS Server ottenuto la stringa ottenuta successivamente, ovvero ".ebay". Procedendo in questo modo, si arriva alla risoluzione completa del nome.
Per questa procedura si possono usare due tipi di approcci:

- **iterativo**:
    1. Il client chiede al nameserver locale la risoluzione del nome;
    2. Il nameserver locale chiede e ottiene la risposta dal Root DNS;
    3. Il nameserver locale chiede (l'informazione ottenuta nel passaggio precedente) e ottiene la risposta dal TLD DNS;
    4. Il nameserver locale chiede (l'informazione ottenuta nel passaggio precedente) e ottiene la risposta dall'Authoritative DNS;
    5. Ottenuta la risposta definitiva, la consegna al client.
- **ricorsivo**:
    1. Il client chiede al nameserver locale la risoluzione del nome;
    2. Il nameserver chiede al Root DNS la risoluzione del nome;
    3. Il Root DNS chiede al TLD DNS la risoluzione del TLD;
    4. Il TLD DNS chiede all'Authoritative DNS la risoluzione del nome;
    5. L'Authoritative DNS consegna la risposta.

3.4.2 Cache

I DNS utilizzano un parametro in ogni loro record, che ne gestisce il tempo di aggiornamento per il flush del record stesso, in modo tale da rendere l'aggiornamento periodico ogni tot tempo, e mantenerlo - nel restante - in cache, aumentando le prestazioni di risoluzione dei domini.

3.4.3 Records

Il DNS conserva *resource records* di vari tipi, tutti con il seguente formato: {name, value, type, ttl}. I quattro formati più importanti:

- **A**:
  - name: hostname
  - value: IP
- **NS**:
  - name: alias
  - value: hostname server
- **CNAME**:
  - name: alias
  - value: nome a cui punta l'alias
- **MX**
  - name: dominio
  - value: hostname server

3.5 FTP

L'*FTP* (*File Transfer Protocol*) è un protocollo utilizzato per il trasferimento di file da/a un host remoto.
Lavora sulla porta 21, ma ne usa - in fasi particolari anche un'altra:

- **Porta 21**: porta standard per la connessione di controllo (persistente), che gestisce i l'autenticazione e un set di comandi per navigare nella directory FTP;
- **Porta 20**: porta - non persistente - utilizzata unicamente per l'effettivo trasferimento del file, che la chiude a trasferimento concluso.

Tutte le richieste FTP sono inviate come messaggi ASCII attraverso il canale di controllo.

3.6 Mail

Composta da:

- Agents (*Thunderbird*, e così via ...);
- Servers (contenente *mailboxes* e le code dei messaggi in uscita - *message queues*);
- Protocolli (*SMTP*, *POP*, *IMAP*).

Ciascun protocollo gestisce funzioni diverse:

- **SMTP** (in TCP, sulla porta 25, RFC 2821) (*Simple Mail Transfer Protocol*): a questo protocollo è delegato il trasferimento delle emails. Segue tre fasi:

  1. handshaking;
  2. transfer;
  3. closure.

  Avviene tutto tramite comandi formati da un ID e un messaggio (codificati in 7 bit). Mentre HTTP è un protocollo *pull* (che richiede risorse), SMTP è un protocollo *push* (che le invia).

- **POP** (in TCP, sulla porta 993, RFC 1939) (*Post Office Protocol*): molto vecchio. Segue unicamente due fasi: l'autenticazione e la transazione (per il download delle email).

- **IMAP** (in TCP, sulla porta 143, RFC 3501) (*Internet Mail Access Protocol*): mantiene ben organizzate tutte le emails sul server, fornendone un'interazione molto più completa.

3.7 Content Delivery Networks

I *CDN Providers* sono infrastrutture che forniscono caching dei contenuti web su larga scala, al posto dei *WSP* (*Web Service Provider*).

Esempio. *Akamai* è uno dei *CDN Provider* più famosi e importanti al mondo, e ha raggiunto nell'ultimo anno 200.000 server, distribuiti in 110 paesi. Si stima che abbia raccolto 30 terabit di dati e 2 trilioni di interazioni al giorno. Le CDN vengono quindi utilizzate per migliorare il tempo di risposta percepito dal client portando i contenuti più vicini al network edge, quindi più vicino all'end-user.

Sono composte delle seguenti componenti:

1. *Distribution System*: struttura effettiva di caching;
2. *Request Router*: struttura delegata per il routing da dominio al CDN più *vicino* (non necessariamente a livello geografico, tendenzialmente per tempi di *ping*);
3. *Acounting-Billing*: struttura che gestisce le iscrizioni, i pagamenti e la configurazione degli IP/domini.

Le CDN sfruttano un meccanismo di redirezione di due tipi:

- **DNS Redirect**: a livello DNS, tramite l'associazione del record *A* del dato dominio su un IP della CDN;
- **HTTP Redirect**: viene utilizzata un'URL apposita in cui specificata sia la *redirezione* alla CDN, con il dominio di riferimento.

3.8 P2P

Nasce a fine anni '80 e diventa popolare con *Napster*, chiuso da un ordinamento del 2001 per violazione di copyright.

3.8.1 P2P networks

Un tipo di network dove ogni peer può agire come client e come server e non deve essere sempre attivo: i peers entrano ed escono dalla rete continuamente.
L'idea di base fondamentale è l'aumento di scalabilità, di disponibilità di risorse, della privacy dei singoli peer e, infine, una drastica diminuzione dei costi.
Al tempo stesso, però, i peer non sono affidabili a causa della possibilità di disconnesione, sono eterogenei a livello di potenza computazionale, di rete e di storage, la ricerca/scoperta delle risorse all'interno della rete P2P spesso non è immediata, senza considerarne i problemi d'integrità.
Paragonate alle CDN, le reti P2P hanno diversi pro, a scapito della *QoS* (*Quality of Service*).

Sono spesso organizzate in maniera molto diversa tra loro.

- **Rete non strutturata** (*LimeWire*): non c'è nessuna regola. Una volta entrato nella rete, viene fornito un insieme di peer e stabilita una connessione. La ricerca di una risorsa avviene tramite *flagging continuo*: se necessito di qualcosa vado dai peer vicini e la richiedo; se questi non la hanno essi stessi fanno la stessa richiesta ai peer a loro più vicini;
- **Rete strutturata** (*KAD*, alcuni *BitTorrent*): quando i peer si connettono vengono indicizzate le loro risorse in una *DHT* (*Distributed Hash Table*), che viene interrogata ogni qualvolta sia necessario ad un peer generico

cercare una data risorsa, accellerandone i tempi di ottenimento, rispetto alle reti non strutturate. Tutto questo è però a scapito dei tempi di login nella / logout dalla rete;

- **Reti ibride**: reti istituite con la logica P2P, ma *aiutate* tramite l'architettura client-server. Un set di server fornisce un indice centralizzato di risorse. Il problema foindamentale è che se questo set di server non fa parte più della rete - per qualsiasi ragione -, la rete non ha più senso di esistere;

- **Reti gerarchiche**: viene dato favore ai *super-peer* (i peer migliori per capacità computazionali, di rete e di storage), ai quali si connette ogni altro peer.

3.8.2 BitTorrent

Sviluppato in Python nel 2001 da Bram Cohen. Prevede un torrent separato per ogni file. I peer scaricano e caricano simultaneamente tutti i torrent. L'insieme di peer attivi è chiamato *swarm*. Questi si dividono in *seed*, che hanno il file completo, e in *leechers*, che lo stanno ancora scaricando.
Gli utenti devono capire quali peers hanno una copia del file. Una volta ottenuto un torrent, questo include un insieme di indirizzi IP, che puntano ad una categoria di server, chiamati *tracker*, delegati di gestire le richieste di ciascun peer. Il *tracker* consegna una lista di peer (50 file) a cui potersi connettere. Questo non è assolutamente coinvolto con la reale distribuzione del file; è una lista di 50 peers al quale il client si connette via TCP.
Il *tracker* inoltre mantiene informazioni sullo stato dei peer: ogni peer ogni 3 minuti inviano informazioni su loro stessi, circa il loro stato. Quando le connessioni di un peer agli originari 50 peer sono decrementate sotto le 20, viene fatta una nuova richiesta al *tracker*, che sostituisce la lista attuale con una nuova.
Le porzioni di file che compongono il file che si ha intenzione di scaricare sono detti *chunk* e hanno una dimensione che varia tra i 10KB a 1MB, anche se tendenzialmente è di 256KB. I *chunk* non vengono scaricati a caso. Alla connesione, il client comincia per prima cosa a scaricare *chunk* randomici, subito dopo quelli più rari (per il rischio di disconnessione e perdita definitiva e assoluta di quei *chunk*).
Come il peer ottiene un *chunk*, lo mette in upload ai peer collegati a lui: in base alla mia velocità di diffusione (in upload), vengo premiato in velocità di ricezione (in download). L'upload usualmente viene indirizzato contemporaneamente a 5 peers:

- 4/5: quelli da cui scarico più velocemente;

- 1/5: randomico, per evitare di provocarne il *choking* (*strozzamento*), perché altrimenti non verrebbe scelto mai.

3.8.3 Spotify

Servizio di streaming di musica on demand *peer-assisted*.
Utilizza un protocollo proprietario. Dal 2014 ha cominciato a dismettere la sua rete P2P per il grande numero di utenti e lasciare solo CDN.
Utilizzava un metodo ibrido di distribuzione contenuti, facendo in modo tale che soltanto 8,8% delle risorse venisse scaricato tramite i server CDN.
Per quanto riguarda la struttura P2P, era una rete di tipo non strutturato, con *flagging* limitato a due *hop* (due ricorsioni, nel routing della richiesta della risorsa ai peer vicini), un sistema semplice e funzionale attuabile grazie alla struttura (di *fallback*) della CDN.

Caching Ogni canzone è criptata in una cache locale. Vantaggi: la canzone non va riscaricata e c'è più possibilità che un client riceva una traccia via P2P. Svantaggio: impatto sullo storage locale dell'utente. Per ovviare a questo problema, è applicata una *Politica LRU* (*least recently used*) per la pulizia della traccia.
Un client non può fare upload se non ha l'intera traccia. Ciò semplifica molto il protocollo: il client non deve comunicare con i peer di quali parti di traccia è in possesso.

Locating Peers Un tracker gestisce un insieme di peer. Ogni server è responsabile per una rete P2P di client distinta e indipendente. Per ogni traccia il server è a conoscenza solo degli ultimi 20 peer che l'hanno riprodotta.
In secondo luogo, ogni client è connesso ad un insieme di *vicini* nella rete, ai quali può far richiesta di risorse tramite flagging - fermato automaticamente dopo 2 hop. Un peer trasmette in upload ad un massimo 4 peers simultaneamente, attraverso chunk di dimensione fissa di 16KB.
Essendo *Spotify* un servizio scritto per essere scalabile in maniera ottimale anche su piattaforme mobili, si è presupposto che una grande quantità di clienti facessero uso di rete mobile per accedere al servizio. Per questo è nata la necessità di gestire in maniera più pulita e funzionale possibile i limiti di banda. Esistono due limiti:

- *Soft*: il client non fa più richieste, ma risponde alle richieste di altri;
- *Hard*: smette totalmente di aprire connessioni.

Predicibilità delle tracce

Circa il 61% delle tracce sono riprodotte in un ordine predicibile, il restante 39 per cento in maniera casuale.
Quando le tracce sono riprodotte in maniere casuale, utilizzare la rete P2P sarebbe improponibile in termini di immediatezza di reperibilità del file della nuova traccia. Spotify utilizza in questo caso solo la CDN: questo significa più peso sulla sua rete - motivo per cui non riproducono più dei primi 15 secondi tramite quella connessione. I 15s sono sufficienti per determinare se l'utente

ascolterà quella canzone o cambierà traccia, e quindi continuare a scaricarla - utilizzando la rete P2P -, o cambiarla.
Nel 92% dei casi, quando un utente ascolta fino agli ultimi 30s di una canzone, ascolterà anche quella successiva. Per questo:

- Mancando 20-30s: buffering via P2P;
- Mancando 10s (o skip track prima che finisca): buffering sulla CDN.

Il client monitora in continuazione il *playout buffer*: se quest va sotto i 3 secondi va in emergenza, bloccando le trasmissioni in upload e richiedendo il file tramite CDN.

3.9 Creazione di un sistema Client/Server

Essendo i protocolli TCP e UDP implementati direttamente dal SO, implementare le opportune funzioni per interagire con essi è l'unica cosa necessaria per permettere l'invio e la ricezione dei dati. I SO mettono a disposizione delle API, insiemi di funzioni per facilitare l'utilizzo di questi protocolli agli sviluppatori. I **socket** sono fondamentalmente delle Internet API. Un socket è una *porta* che collega un'applicazione con un protocollo di trasporto; in Python è rappresentato da una struttura dati.

```
import socket
s = socket.socket(addr_family, type, protocol)
```

Gli argomenti della funzione socket() specificano le informazioni necessarie per la creazione di un nuovo socket.

- **addr_family**: famiglia di protocolli.
  - socket.AF_INET: IPv4;
  - socket.AF_INET6: IPv6,
  - socket.AF_UNIX: per gestire comunicazioni tra processi sulla stessa macchina fisica.
- **type**: specifica se il tipo di comunicazione è orientata allo stream o a pacchetto (sostanzialmente, la differenza tra TCP e UDP):
  - socket.SOCK_STREAM: stream-oriented (TCP);
  - socket.SOCK_DGRAM: message-oriented (UDP);
  - socket.SOCK_RAW: fornisce accesso al network layer.
- protocol: se 0, specifica che il protocollo - in definitiva - da utilizzare è quello definito dalla tupla *(addr_family, type)*.

3.9.1 TCP

SERVER-SIDE

socket() Definito nella pagina precedente.

bind(*address*) *address* rappresenta una tupla *(host,port)*, dove *host* può essere un hostname o un indirizzo IP. Qualora *host* fosse una strina vuota, si farebbe un *bind* su tutte le interfacce di rete; invece, se fosse *localhost*, lo si farebbe soltanto su quella di *loopback*. *port*: rappresenta il numero di porta su cui fare il *bind*

listen(*backlog*) Rimane in attesa di richieste di connessione su quel socket. Il *backlog* specifica il numero massimo di connessioni accodate (su Linux di default a 5, per motivi di sicurezza, come prevenzione da *SYN flood attacks*). Se il *backlog* è pieno, nuove connessioni possono essere ignorate o rifiutate.

accept() Preleva e stabilisce una connessione dal *backlog*, creando un nuovo *socket attivo*.

send() Descritto nel *client-side*.

recv(*bufsize[, flags]*) Ritorna la stringa di dati ricevuti, delimitati da una quota massima, il *bufsize*. In caso di assenza di dati, si blocca e il programma è in pausa fino all'arrivo di dati. Se invece torna una stringa vuota, significa che l'host dall'altro capo ha chiuso la connessione.

close() Descritto nel *client-side*.

CLIENT-SIDE

socket() Definito nella pagina precedente.

connect(address) Si connette ad un socket remoto. Il SO assegna automaticamente un indirizzo e un numero di porta.
Argomenti: come server-side. Se è utilizzato un socket TCP, comunica al SO di iniziare la *3-way handshake*.

send() Invia i dati in input, ritornando un intero che indica il numero di byte (rispetto alla lunghezza dell'input) effettivamente inviati. Possono verificarsi i seguenti casi:

- I dati sono correttamente ed interamente inviati;
- I dati sono parzialmente inviati (buffer parzialmente pieno): TCP deve spezzare i dati e ritorna soltanto la lunghezza in byte che è stata effettivamente inviata (il comando *sendall* si assicura che venga inviato tutto);
- I dati non possono essere inviati (buffer pieno).

recv() Descritto nel *server-side*.

close() Chiude il socket: tutte le operazioni future sull'oggetto falliranno. La connessione non è chiusa immediatamente, perché il SO deve prima finire di inviare tutti i dati ancora presenti nel buffer.

Chiusura di un socket

Si è accennato all'utilizzo di una procedura *close()*, che permette la chiusura - non propriamente immediata - di un socket *attivo*. Qualora fosse però necessario definire in maniera più dettagliata il comportamento di un socket, o meglio, di una connessione relativa ad un socket, sarebbe più opportuno utilizzare il metodo *shutdown()*.
Questo - a differenza del *close()* - chiude uno dei due sensi di comunicazione, senza distruggere il socket:

- SHUT_RD: chiude il canale server-client;
- SHUT_WR: chiude il canale client-server;
- SHUT_RDWR: chiude entrambi i canali.

Interfaccia di *Loopback*

Interfaccia network virtuale per gestire comunicazioni tra processi sullo stesso host. Bypassa l'hardware dell'interfaccia network locale e i livelli più bassi della pila TCP/IP

Socket Passivo / Attivo

Il socket di cui fino a questo punto si è parlato è il socket *passivo*. Come accennato nelle procedure di implementazione di un socket in *python*. I *socket attivi* si distinguono da quelli *passivi* per il fatto che sono effettivamente usati per la trasmissione dati. I *passivi* sono creati da un processo, ma poi delegano a nuovi socket, con lo stesso *address* (la tupla *(indirizzo, porta)*), lo scambio effettivo dei dati. Un *socket attivo* è definito univocamente dalla quadrupla *(local_ip, local_port, remote_ip, remote_port)*.

Porte

I numeri di porta - essendo in *16bit* - vanno dalla porta 0 alla 65535:

- porte note: 0-1023 (dedicate, necessari i privilegi di root);
- porte registrate: 1024-49151 (utilizzate da servizi noti, non necessari però i privilegi di root);
- porte dinamiche: 49152-65535 (porte utilizzabili).

IANA (o *Internet Assigned Numbers Authority*) è l'organizzazione che si occupa di gestire la mappatura di associazioni *porta-servizio*.

Implementazioni utili

```
def recv_all(sock, len):
    data = ""
    while len(data) < len:
        read_data = sock.recv(len - len(data))
        if read_data == ‘‘:
            raise EOFError(’socket closed.’)
        data += read\_data
    return data
```

3.9.2 UDP

In UDP - non esistendo connessione - si utilizza soltanto il comando di creazione socket, il *bind()*, il *sendto()* (che prende in input anche un nuovo parametro *address*, che specifica la destinazione dei dati da inviare), il *recvfrom()* (che da in output un nuovo parametro *address* in riferimento al client inviante).
Si può usare la funzione *connect()* con sockets UDP per evitare di specificare ogni volta l'indirizzo del server sul quale vogliamo chiamare una *sendto()*. In questo modo il client non prevede la ricezione di pacchetti da altri mittenti. Si noti che l'utilizzo di *connect()* non implica l'invio di dati.

socket.settimeout(*value*) Imposta un timeout (in secondi) ricaricato ogni volta che si ricevono dati, al termine del quale il SO lancia una socket.timeout exception.

Capitolo 4

Livello di Trasporto

4.1 Protocolli di Trasporto

I protocolli di trasporto sono un insieme di protocolli che risiedono sugli host e sugli endsystems; si occupano di fornire le regole per una comunicazione logica tra due entità remote.
A livello di invio, si occupano della ricezione dei messaggi in segmenti e li inviano al network layer; alla ricezione, i segmenti sono riassemblati in messaggi e passati al livello applicativo.
Mentre il livello di rete offre comunicazione tra due endsystems (trasportando informazioni da un host all'altro), il livello di trasporto crea un collegamento logico tra due *processi*. Deve quindi essere in grado di raccogliere flussi informativi, utilizzare il livello di rete per trasportarli e successivamente smistarli nei vari processi applicativi.

4.2 Internet transport-layer protocols

A dispetto del fatto che il livello rete utilizza una best-effort policy che non dà nessuna garanzia di consegna, **TCP** si occuperà di applicare meccanismi per la traduzione di informazioni agli opportuni processi anche nel caso di perdita di dati.
I principali tipi di perdita dei dati sono:

- un eccesso di informazioni arrivate al router intermedio.
  La conseguenza è un riempimento delle code fino al raggiungimento di una congestione in rete e all'inizio di una perdita di pacchetti.

- il ricevimento a destinazione di un tasso di informazioni troppo elevato rispetto alla velocità di lettura.

Si noti che un controllo del flusso di informazioni richiede algoritmi diversi rispetto al controllo di congestione; entrambi i tipi di controllo sono implementati da TCP.

4.3 Multiplexing e Demultiplexing

L'invio di informazioni da parte di un processo è eseguito attraverso una determinata porta dotata di interfacce ben definite. Tale porta, che consente il colloqui con l'entità sottostante di livello trasporto, prende il nome di *socket*. Poiché più flussi informativi sono trasmessi tramite lo stesso canale logico, occorre stabilire delle funzioni di smistamento di dati. Queste funzioni sono implementate nei protocolli TCP e UDP in maniera differente.

4.3.1 UDP: Connectionless demux

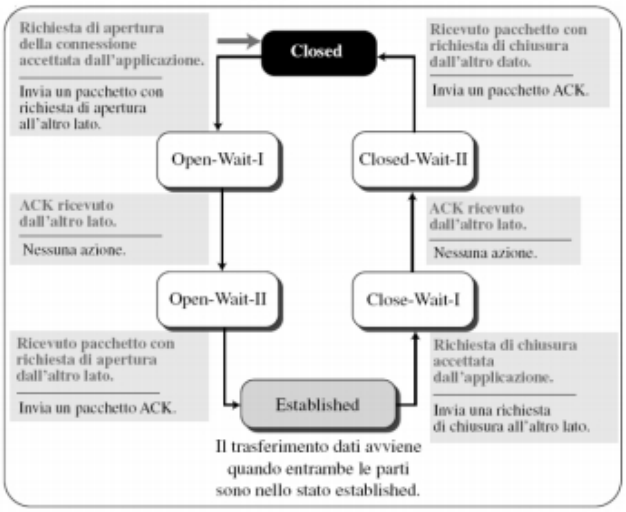
Il socket creato è dotato di un numero di porta dell'host locale; quando viene creato un *datagram* (pacchetto) da inviare nel socket UDP, questo deve specificare l'indirizzo IP e il numero di porta dell'host di destinazione. Quando l'host remoto riceve un segmento via UDP, controlla il suo numero di porta di destinazione e lo indirizza alla porta corrispondente. Pacchetti con stesso numero di porta di destinazione, ma indirizzi IP o porte di origine differenti saranno quindi orientati allo stesso socket a destinazione.

4.4 TCP: Connection-oriented demux

In questo caso viene creato un socket dedicato al flusso di informazioni identificato dalla presenza di quattro campi: due per gli indirizzi IP del mittente e del destinatario, due per i numeri di porta del mittente e del destinatario. Nel caso di più client connessi allo stesso webserver, all'inizio viene fatta una richiesta su un *welcome-socket* generale; successivamente le richieste sono smistate ad un socket dedicato su ciascuna delle connessioni: in questo modo è assicurato che il flusso di dati passerà per socket differenti.

4.5 Reliable Data Transfer

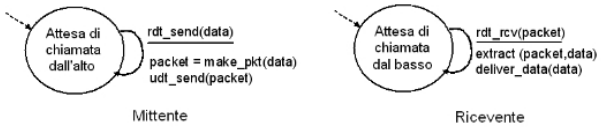
La necessità di strutturare un protocollo di trasferimento di dati affidabile nasce dall'importanza che due entità, in comunicazione, siano in equilibrio in termini di velocità di produzione/trasmissione/consumo dei dati, senza che l'eventuale fallimento nella trasmissione tramite livelli inferiori - nella pila TCP/IP - causi il definitivo fallimento della trasmissione.
Sarà fatto uso della rappresentazione FSM, una tipologia di rappresentazione utile alla sintetizzazione grafica dei servizi orientati alla connessione.



4.5.1 RDT 1.0

Il caso riguardante *RDT 1.0* è contestualizzato in una situazione perfettamente affidabile:

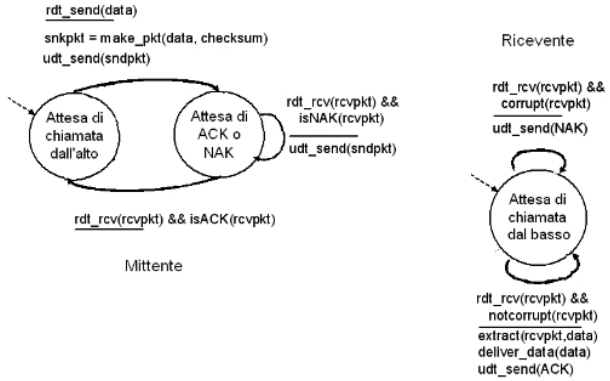
- non si verificano **mai** errori nei bit trasmessi;
- non si verifica **mai** una perdita di pacchetti.



4.5.2 RDT 2.0

In questo caso, il canale di trasmissione effettiva può confondere i bit nel pacchetto. Da qui la necessità di introdurre dei messaggi di risposta alla ricezione di un pacchetto:

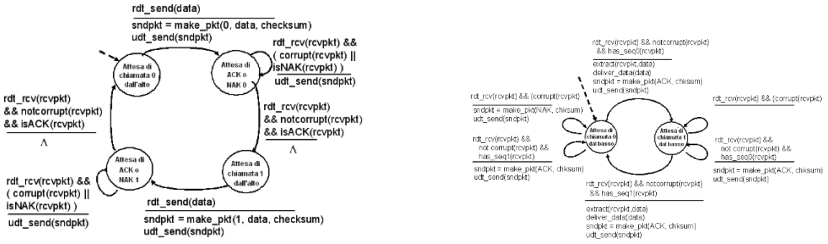
- **ACK** (*ACKnowledgment*, o *conferma di ricezione*): utilizzato per confermare che il pacchetto è stato ricevuto ed è integro;
- **NAK** (*Negative AcKnowledgment*, o *conferma negativa*): il pacchetto contiene errori. Il mittente reinvia il pacchetto quando riceve indietro questo tipo di messaggio: motivo per cui si tratta di un protocollo *ARQ* (*Automatic Repeat reQuest*, o *Richiesta Automatica di Ripetizione*).



RDT 2.0 può però andare incontro ad un difetto fatale: la mancata gestione degli eventi in cui i pacchetti ACK/NAK siano danneggiati.

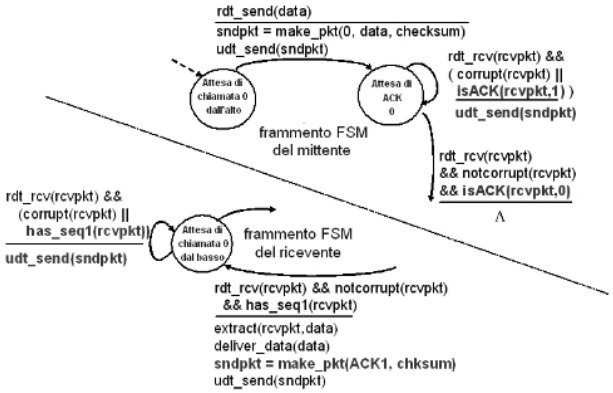
4.5.3 RDT 2.1

Viene dunque aggiunto - alternativamente - un valore binario a ciascun pacchetto, un *numero di sequenza*. In questo modo, viene comunicato non solo l'esito della ricezione al mittente, ma anche il pacchetto a cui questo fa riferimento.



4.5.4 RDT 2.2

RDT 2.2 fornisce le stesse funzionalità di *RDT 2.1*, utilizzando però soltanto gli ACK, escludendo i NAK. Nel caso in cui il destinatario dovesse inviare un NAK, invia nuovamente un ACK con numero di sequenza dell'ultimo pacchetto ricevuto correttamente.



4.5.5 RDT 3.0

L'alternativa è invece introdurre una nuova funzione di timing. Per ogni pacchetto inviato, il mittente implementa un timer: una volta che questo raggiunge un timeout, se non ha ricevuto nessun pacchetto che gli comunichi l'esito (l'ACK), allora reinvia il pacchetto. Per questo motivo:

- il mittente deve tenere una copia del pacchetto spedito fino a che non ne riceve riscontro dell'esito;
- per garantire il controllo di flusso, non si spedisce più di un pacchetto alla volta.

Il parametro RTT (o *Round Trip Time*) viene definito come il tempo necessario per l'invio di un pacchetto sommato a quello per la ricezione di un pacchetto riferito all'esito del precedente.

Esempio. In un link da 1Gbps, con un *propagation delay* di 15ms e 8000bit per ciascun pacchetto abbiamo:

$$D = \frac{L}{R} = \frac{8000bit}{10^9 bit/s} = 8microsecs$$
$$U = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{0{,}0008}{30{,}0008} = 0{,}00027$$

4.5.6 Protocolli a Ritrasmissione Automatica (o *a finestra*)

A differenza dei precedenti, viene trasmessa sempre una sequenza di pacchetti. A man mano che si ricevono gli ACK, vengono aggiunti nuovi pacchetti in coda.
In questo caso: $U = \frac{\frac{3L}{R}}{RTT+\frac{L}{R}}$

Go-back-N. Il mittente può avere fino a N pacchetti non riconosciuti nella pipeline. Il destinatario invia solo *ACK cumulativi*, ovvero relativi ad un gruppo di pacchetti ricevuti. Il mittente utilizza un timer per legato al primo pacchetto inviato che ancora non ha ricevuto ACK: al suo raggiungimento, reinvia tutti i pacchetti a partire da quello per il quale ha impostato il timeout.

Selective Repeat. Il mittente può avere fino a N pacchetti. Il destinatario invia un *ACK individuale*, relativo quindi ad ogni pacchetto. Il mittente utilizza invece un timer per ogni pacchetto di cui non ha ricevuto esito; quando raggiunge il timeout, ritrasmette solo quel dato pacchetto.

4.6 Ritrasmissione Automatica in TCP

TCP utilizza una versione ibrida dei protocolli *Go-back-N* e *Selective repeat*: invia un flusso di singoli bytes che formano - per mano del ricevente - un segmento, il cui *sequence number* è quello del primo byte contenuto, e la cui lunghezza è determinata dal numero totale di bytes. Utilizza inoltre una variabile definita come *MSS*, che indica la lunghezza massima di ciascun segmento.
In un'unica connessione, il flusso di dati è bidirezionale e, per istanziarla, viene eseguito un *handshake* per inizializzare i buffer e alcune variabili (come la *MSS* stessa). La dimensione dei buffer varia nel tempo, sulla base delle informazioni raccolte dai controlli di flusso e congestione.
Ogni segmento è composto dai seguenti campi (ciascuno da 32bit):

- Porta sorgente / destinazione;
- *Sequence number*;
- *Acknowledgment number*: eventuale ACK di riscontro per un pacchetto appena ricevuto (necessario perché la connessione è bidirezionale);
- *Checksum*;
- *Header length*: se si desidera aggiungere informazioni addizionali, è necessario specificarne la lunghezza;
- *Receive window*: indica quanto spazio ho ancora nel buffer (necessario per la gestione del flusso);

- *RST/SYN/PIN*: variabili settate a *1* in fase di apertura/chiusura della connessione;
- *ACK*: variabile settata a *1* se il ricevente deve leggere il campo *Acknowledgment number*;
- *URG*: indica che si tratta di un pacchetto più urgente della media;
- *Application Data*: dati da trasmettere.

4.6.1 Meccanismi di controllo affidabile

In *TCP* - lato *sender* - viene, per ciascuna connessione e trasmissione, creato un segmento con *numero di sequenza n*, che corrisponde al primo byte del flusso di bytes in trasmissione attraverso quello stesso segmento.
Se un timer è già attivo, rimane invariato, perché ancora non è stato ricevuto riscontro su una serie di segmenti; altrimenti, lo avvio.
Al timeout, ritrasmesso il primo segmento che lo ha causato, ma non tutto, perché *TCP* permette l'inserimento dei buffer contenenti i flussi di bytes ricevuti non necessariamente in sequenza per conservarli. Poi viene riavviato il timer.
Al ricevimento dell'*ACK*, si controlla se è relativo ad un segmento di cui non si conosce l'esito di ricezione:

1. applico l'*ACK* a tutti i segmenti di cui non avevo ancora ricevuto l'esito fino a quest'ultima ricezione;
2. avvio un nuovo timer.

4.6.2 Variabilità del retransmission timeout

Ovviamente, il timeout deve essere necessariamente $< RTT$ (*RoundTrip Time*).
Inoltre:

- se troppo corto, avvengono ritrasmissioni inutili e non necessarie;
- se troppo lungo, i tempi di reazione diventano troppo lunghi se i segmenti sono andati perduti.

Per ovviare a questi problemi, è stato introdotto una stima calcolata per ottenere dinamicamente un timeout più adatto alle prestazioni di rete di ciascuna connessione. Viene utilizzato un *SampleRTT*, ovvero l'RTT dell'ultima trasmissione eseguita della quale è stato ricevuto l'*ACK*. A questo punto viene utilizzata una media esponenziale pesata per tenere in conto della variabilità nel tempo dei vari *SampleRTT*, nella cui formula viene utilizzata una variabile α, tipicamente impostata a *0,125*, la cui funzione è quella di dare maggiore importanza ai *SampleRTT* più recenti, rispetto che a quelli più lontani nel tempo. Il risultato è la seguente formula:

$$EstimatedRTT = (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT$$

A questo punto, questa stima viene utilizzata nel calcolo della deviazione dell'RTT, in cui compare una nuova variabile β, la cui funzione è identica a quella della variabile α vista in precedenza, ma che è tipicamente valutata *0,25*:

$$DevRTT = (1 - \beta) \times DevRTT + \beta \times |SampleRTT - EstimatedRTT|$$

Infine, il valore del timeout stimato come migliore possibile ed effettivamente utilizzato:

$$Timeout = EstimatedRTT + 4 \times DevRTT$$

4.6.3 Controllo del flusso

Il *Flow Control* è utilizzato per evitare di saturare il *buffer* dentro cui vengono conservati i pacchetti ricevuti, prima che vengano prelevati dal *layer applicativo*. Se questo venisse riempito troppo rapidamente, rispetto alla velocità di prelevamento dei dati, ci si troverebbe in una situazione di criticità.
Per questa ragione, vengono segnalate al *sender* informazioni sullo spazio a disposizione, nel campo *receive window* (da *16bit*) nel segmento: il *sender* adegua il flusso dei dati trasmesso di conseguenza.
La *finestra* e il *buffer* sono configurati nell'*handshake* della connessione, tramite i campi:

- **MSS** (*Maximum Sequence Size*): impostato tipicamente a 2K, indica la lunghezza massima del segmento;
- **ISN** (*Initial Sequence Number)*: impostato tipicamente a 2047 (e chiaramente variando in base al valore del *MSS*), indica il valore iniziale del *sequence number*;
- **WIN** (*WINdow*): impostato tipicamente a 4K, indica la dimensione della finestra.

Ad ogni trasmissione, viene specificato anche il valore *WIN* sulla base dello spazio rimanente disponibile sul *buffer* del ricevitore.
Potrebbe però accadere che il segmento di notifica da parte del ricevitore che lo spazio sul *buffer* è tornato disponibile vada perduta. Per questa ragione, il *sender* continua *sempre* ad inviare pacchetti - piccoli, se il *buffer* è stato segnalato come saturo -, a distanza di timeout crescenti nel tempo (possono raggiungere massimo 60s), così da rimanere in contatto con il ricevitore e poter sapere se il *buffer* contiene nuovamente spazio a sufficienza per ricevere dati utili.

4.7 Handshakes

Prima di scambiare dati, il ricevente e il destinatario compiono una "stretta di mano" (*handshake*): questo si traduce nello stabilire che entrambi sono a conoscenza del fatto che l'altro vuole connettersi e che entrambi sono d'accordo sui parametri di connessione.
Un handshake semplice prevede che il client mandi una richiesta di connessione, riceva una risposta positiva dal server e successivamente possa iniziare ad inviare dati.
Il problema è che i dati possono essere intercettati, ritardati, duplicati o addirittura perduti. Prendendo in considerazione l'invio di dati importanti, questo potrebbe portare a conseguenze disastrose. Per questo motivo, al momento della richiesta si inseriscono un segmento di SYN e una 'proposta' nel sequence number ($seq = x$). Quando il destinatario riceve la richiesta, invia un segmento di SYNACK per comunicare di aver ricevuto la richiesta ($seq = y; ack = x+1$). Al momento della ricezione del segmento di SYNACK, il client stesso invia un segmento che contiene l'ACK della ricezione e i dati da inviare ($seq = x+1; ack = y+1$)

Initial Sequence Number. Per motivi di sicurezza, i sequence numbers dovrebbero cambiare nel tempo.
RFC 793 suggerisce di generare sequence numbers iniziali (ISN) semplici come un counter da 32 bit incrementati ogni $4\mu s$ e trasmessi ogni volta che la flag SYN è attiva.
Notare che sia il mittente sia il destinatario trasmettono il *loro* ISN. A partire dall'ISN, i byte di dati sono numerati da $ISN + 1$ in modo da permettere il *synack*.

Forbidden Region. Per impedire che due ISN identici si trovino in rete nello stesso momento, è stato introdotto un "periodo di silenzio" nel quale non possono essere inviati segmenti.

4.7.1 Il problema dei due eserciti

Il problema del controllo nella trasmissione dati è facilmente spiegabile con il seguente esempio. Prendiamo in considerazione tre armate, due appartenenti alla fazione *verde* ed una appartenente alla fazione *rossa*. Le due armate verdi sono singolarmente più deboli di quella rossa, ma l'armata rossa è globalmente più debole. Le armate verdi devono quindi attaccare quella rossa nello stesso momento per vincere la battaglia. La prima soluzione che può venire in mente per fare in modo che le armate verdi si accordino per attaccare è che una invii un messaggio con l'orario di attacco e l'altra ne invii uno per confermare la ricezione.
I messaggeri che li portano possono però essere catturati, quindi il messaggio può non arrivare correttamente a destinazione. Per assicurarsi che ogni messaggio sia correttamente ricevuto bisognerebbe quindi inviare un messaggio di conferma

ad ogni ricezione, ma ciò porterebbe ad un loop in cui ogni armata continua ad inviare messaggeri per confermare la ricezione del messaggio dell'altra. Per ovviare a questo problema, si adotta la seguente soluzione:

```
g1 to g2: "attacco alle 6."
g2 to g1: "conferma ricezione messaggio."
g1 to g2: "conferma ricevuta, fine trasmissione."
g2 to g1: "conferma fine trasmissione."
```

Introducendo una comunicazione di fine trasmissione, il ricevente non deve fare altro che inviare un'ultima conferma per porre fine alla trasmissione.
Ciò si traduce in TCP con una flag denominata *FIN* che stabilisce il termine della connessione; poichè la connessione è doppia, FIN dev'essere inviato (e confermato) da entrambi i lati.

```
a to b: connection request
b to a: connection ack
a to b: ack, data
b to a: data ack
-----------------------
a to b: data
b to a: data + ack
a to b: data + ack
b to a: data + ack
[...]
-----------------------
a to b: fin
b to a: ack fin
b to a: fin
a to b: ack
```

4.7.2 3 Way Handshake in TCP

1. Il client invia un segmento TCP SYN al server e specifica ISN senza invio di dati.

2. Il server riceve SYN e risponde con SYNACK; alloca il buffer e specifica ISN.

3. Il client riceve SYNACK; alloca buffer e variabili, risponde con ACK che potrebbe contenere dati.

4. Per chiudere la connessione uno dei due estremi invia un messaggio con flag $FIN = 1$; il ricevente conferma e manda a sua volta un messaggio con $FIN = 1$ e attende l'ultima conferma.

4.7.3 Connection states: client

0: closed
1: syn_sent (ricevo *synack*; invio *ack*)
2.1: established;
...
2.n: established (invio *fin*)
3: fin_wait_1 (ricevo *ack*)
4: fin_wait_2 (ricevo *fin*, invio *ack*)
5: close wait

4.7.4 Connection states: server

0: closed
1: listen (ricevo *syn*, invio *synack*)
2: syn_rcvd
3: established
4: close_wait
5: last_ack

4.8 Principi di controllo della congestione

Nelle reti a pacchetto, i pacchetti attraversano una grande quantità di dispositivi diversi, come ad esempio router, switch e bridge. Questi dispositivi, e i collegamenti che li interconnettono, hanno capacità di elaborazione e di trasmissione finite che possono portare, in molti casi, a situazioni di congestione: i nodi suddetti potrebbero cioè non essere in grado di smistare tutto il traffico offerto in ingresso da varie connessioni tra utenti causando perdita di pacchetti e/o eccessivi ritardi di coda.
Il controllo della congestione permette dunque di migliorare le prestazioni della rete evitando perdite di pacchetti e limitando il ritardo a causa delle ritrasmissioni dei pacchetti persi.

Sono due i principali approcci implementati ai fini del controllo della congestione; il primo end-to-end, il secondo network-assisted.

End-End Congestion Control. Questo meccanismo di controllo non prevede feedback esplicito dalla rete; la congestione è dedotta dagli end-systems che si occupano direttamente di osservare la perdita e il ritardo dei pacchetti.

Network-Assisted Congestion Control. I routers forniscono feedback agli end-systems tramite un singolo bit indicante il tipo di congestione (SNA, DECbit, TCP/IP ECN, ATM); in alternativa tentano di prevenire la congestione specif-

icando una quantità massima di dati che è possibile inviare.

TCP congestion control: additive increase, multiplicative decrease. L'approccio di TCP prevede che il mittente aumenti il tasso di trasmissione (*window size*) fino al verificarsi di una perdita dati.
Quando la trasmissione è normale, l' incremento è additivo: viene aumentata la dimensione della "finestra" di 1 MSS ogni RTT. Al verificarsi di una perdita la dimensione della finestra è dimezzata.

TCP Slow Start. All'avvio della connessione, inizialmente il **cwnd** è impostato a 1 MSS. TCP raddoppia il cwnd ogni RTT, incrementandolo per ogni ACK.

Capitolo 5

Livello di Rete

Il livello di rete si occupa dell'effettivo trasporto delle informazioni dal sender al receiver, incapsulandole in datagrammi - lato sender - e consegnandoli al livello di trasporto - lato receiver. Per la consegna, esamina il pacchetto, analizzando l'header e tutte le informazioni in esso contenute (principalmente l'IP), e determinandone l'instradamento più consono.

Forwarding Dirotta i pacchetti dall'input del router all'output dello stesso, associando un dato valore dell'header ad un link locale.

Routing Determina la rotta che collegherà la sorgente al destinatario, utilizzando appositi algoritmi di routing.

5.1 Connection Setup

Prima di verificare un percorso si deve verificare che ci sia tutto il necessario perché la trasmissione arrivi a compimento (che esistano le risorse necessarie, e così via ...). Quindi, prima che i datagrammi comincino a scorrere, i due *end-hosts* e il router delegato aprono una *connessione virtuale* dedicata. Inoltre, spesso sul livello direte vengono utilizzate delle funzioni importanti di terze parti, che definisco l'architettura di rete:

- **ATM** (o *Asynchronous Transfer Model*);
- **frame-relay**;
- **X.25**.

La differenza sostanziale tra il *livello di rete* e il *livello di trasporto* è che il primo mette in comunicazione due hosts (ed i vari router delegati, eventualmente),

mentre il secondo due processi.
Il livello di rete può garantire diversi gruppi di garanzie:

- Sul singolo datagramma:
  - garanzia di consegna;
  - garanzia di consegna entro un massimo di 40ms di ritardo.
- Sull'intero flusso di datagrammi:
  - consegna in ordine;
  - garanzia di non superare la banda di flusso;
  - garanzia di applicazione di restrizioni imposte nel corso della trasmissione.

Internet (inteso come architettura di rete) non fornisce alcuna garanzia.
La rete a datagrammi però fornisce un servizio *connectionless*, la cui alternativa è quella di utilizzare un approccio a *circuito virtuale* utilizzato - per esempio - nell'architettura *ATM* (determinando in una fase di setup il percorso da seguire - tabelle di routing - e mantenendo le informazioni di stato, che permettono di allocare risorse utili alla trasmissione).

VC (Virtual Circuit) Il sistema a *VC* fa utilizzo di una tabelle di instradamento, che mette associazione ogni VC entrante - tramite un *ID* univoco identificativo - ad un altro *VC* uscente.
L'approccio a *VC* viene utilizzato nelle architetture *ATM*, *frame-relay* e *X.25*, ma non nell'architettura di internet attuale.

Nel caso di un approccio di tipo datagram - usato in internet -, non ho una fase di setup - e quindi nessun'informazione di stato -, e ogni pacchetto contiene tutte le informazioni e le coordinate necessarie per la consegna.
La forwarding table cerca di compattare il maggior numero di informazioni nel minor spazio possibile, raggruppando gli IP per intervalli, e associando questi gruppi ad un dato link di uscita. Più concretamente, viene calcolato un *longest address prefix* (e quindi cominciando con una dato prefisso binario - dove la sequenza binaria corrisponde all'IP): tutti gli IP che cominciando con lo stesso prefisso, vengono instradati verso lo stesso link.

Esempio. Ho una sequenza: *11001000 00010111 00010110 10100001*. Leggendo questa sequenza bit per bit, controllo il più lungo prefisso che ha in comune con uno degli instradamenti descritti nella tabella di instradamento, dirottandolo verso il link associato a questo.

Quindi, dal router, vengono eseguiti degli algoritmi/protocolli di routing (*RIP*, *OSPF*, *BHP*), per collegare il link d'entrata con quello d'uscita.

5.2 Funzioni di un router

5.2.1 Input ports

Viene gestito un buffer che contiene una coda dei pacchetti da instradare. In questa fase si utilizza (o si stabilisce) una tabella degli instradamenti, che mettono in comunicazione link di ingresso e di uscita tramite degli appositi switch.

5.2.2 Switching fabrics

Vari tipi di switch:

1. Switching a memoria: prima generazione.
2. Switching a bus: ho n linee di ingresso e n linee di uscita, messe in collegamento tramite un bus condiviso. La pecca è che la velocità è limitata dall'architettura del bus. Inoltre, si può prelevare un datagramma alla volta, solamente per intero.
3. Switching a rete di interconnessione (*crossbar*): ho n linee di ingresso e n linee di uscita, interconnesse tra loro. In questo modo posso raggiungere tramite qualsiasi linea di ingresso, una qualsiasi linea di uscita. In questo caso i datagrammi possono essere prelevati divendoli a celle di lunghezza prefissata, dando all'architettura più elasticità.

5.2.3 Output ports

Viene gestito un buffer dei datagrammi, in modo tale da gestire anche il caso in cui il tasso di trasmissione e ricezione dei datagrammi sia maggiore rispetto a quello di elaborazione dei datagrammi ricevuti.

HOL (Head-of-the-Line) blocking Blocco della coda dei pacchetti su un link d'ingresso, causato da un pacchetto non può essere ancora consegnato al suo link uscita, perché occupato a sua volta nell'elaborazione di altro pacchetto già inviato.

5.3 Datagramma IP (IPv4)

Grandezza minima header IP: 20byte. Campi:

1. *ver*: versione IP (IPv4 o IPv6);
2. *header length*: lunghezza dell'header in byte;
3. *type of service*: tipo dei dati, di datagramma inviato, base della gestione di servizi diversi attraverso la stessa struttura di datagrammi;

4. *time to live*: massimo numero di link che possono essere attraversati dal datagramma in questione;
5. *upper layer*: simile al numero di porta. Specifica a chi va consegnato a destinazione il datagramma, che sia un protocollo TCP o UDP o un altro tipo di protocollo arbitrario;
6. *16-bit indentifier, flgs, fragment offset*: parametri utilizzati per la frammentazione e la ricomposizione del datagramma una volta arrivato a destinazione;
7. *options*: campo opzionale, utile per tracciare, ottenere informazioni riguardo la rotta e/o la trasmissione generale del datagramma;
8. *32bit source IP*: IP sorgente;
9. *32bit destination IP*: IP destinatario;
10. *data*: dati trasportati.

5.3.1 IP Fragmentation / Riassembly

I link di rete hanno l'MTU (la *max transfer size*). Spesso occorre - per questa ragione - comprimere la grandezza di un datagramma, perché questo possa attraversare tutti i link. Per questa ragione di parla della fragmentation e del riassembly. Attraverso questi parametri viene indicato come è stato frammentato il datagramma (e in quanti sotto-datagrammi) e come poter riassemblarlo una volta giunto a destinazione. Ciascun sotto datagramma ha in comune il parametro identificativo di 16bit e tutti, ad esclusione dell'ultimo pacchetto derivato dalla frammentazione, hanno il parametro *fragflag* settato ad *1*.

5.4 Indirizzo IP

Ogni indirizzo IP viene ad essere associato ad una porta di rete. Infatti, il router, avendo più porte di rete, ha un diverso indirizzo IP per ciascuna.
L'indirizzo IP è una sequenza a 32bit, che viene divisa in 4 sequenze da 8 bit, ciascuna corrispondente al valore decimale delle 4 componenti dell'indirizzo IP comunemente utilizzato: *1.1.1.1* : *00000001 00000001 00000001 0000001*.

5.4.1 Subnet

La subnet è la sottorete che mette in comunicazione tutti gli indirizzi IP con *subnet part* (sequenza *high order bits*) in comune.

CIDR (Classless InterDomain Routing) Approccio di rappresentazione di subnet tramite la rappresentazione dei bit della *subnet part*, alla quale viene aggiunta la *host part* settata a zero. Infine si aggiunge un parametro che indica la capienza totale della subnet: *192.168.1.0/24* (subnet *192.168.1*, che può contenere un massimo di 255 *host part* diverse, e quindi 255 indirizzi IP univoci).

DHCP (Dynamic Host Configuration Protocol) Protocollo di configurazione e associazione automatica dell'indirizzo IP ad un dato host in connessione ad una data subnet. Sostanzialmente l'host è capace in questo modo di ricevere dal server di rete un indirizzo valido dinamicamente, così da potersi unire alla rete. Il DHCP può fornire anche il *first-hop router* (*gateway* della subnet) e il DNS server.
Le richieste DCHP sono incapsulate in *UDP*.
Questo permette il riutilizzo di indirizzi (che vengono tenuti occupati limitatamente al tempo entro il quale sono effettivamente connessi).

5.5 ICMP

Per comunicare informazioni a livello di rete, hosts e routers utilizzano l'*Internet Control Message Protocol*. Tali informazioni includono report di errori (host irraggiungibile, network, porte, protocolli) e richieste/risposte echo (usate da ping).
Tali messaggi risiedono nei datagrammi IP e sono composti da tipo, codice e i primi 8 byte del datagramma IP che ha causato l'errore.

Type	Code	Description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest. host unreachable
3	2	dest. protocol unreachable
3	3	dest. port unreachable
3	6	dest. network unknown
3	7	dest. host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Traceroute e ICMP Il mittente invia una serie di segmenti UDP a destinazione, il primo con TTL = 1, il secondo con TTL = 2 e così via. Quando

l'n-esimo set di datagrammi arriva all'n-esimo router, il router scarta i datagrami e invia i messaggi ICMP del mittente. Una volta arrivati, il mittente registra gli RTT.
Per terminare il processo, quando tutti i segmenti UDP arrivano a destinazione, il destinatario restituisce un ICMP di tipo 3, codice 3: *port unreachable*. A questo punto, il mittente interrompe l'invio.

5.6 IPv6

Il motivo iniziale dell'utilizzo di questa versione di IP è il fatto che lo spazio di un indirizzo da 32 bit impiega poco tempo per essere completamente allocato.
Il formato dei datagrammi IPv6 prevede un header a dimensione fissa di 40 byte e impedisce la frammentazione.

- **pri**: indica la priorità tra datagrammi nel flusso
- **flow label**: indica datagrammi nello stesso "flusso"
- **next header**: indica il protocollo del livello più alto per i dati

Altri cambiamenti rispetto a IPv4 includono:

- la **checksum** è rimossa interamente per ridurre il tempo di processamento ad ogni hop;
- le **opzioni** sono consentite, ma al di fuori dall'header, e indicate dal campo *next header*;
- **ICMPv6**, una nuova versione di ICMP, include tipi di messaggi aggiuntivi (come *"Packet Too Big"*), e funzioni di gestione multicast di gruppo.

Transizione da IPv4 a IPv6 Non tutti i routers possono essere aggiornati contemporaneamente. Perchè il network possa operare contemporaneamente con router IPv4 e IPv6, è introdotto il **tunneling**: i datagrammi IPv6 sono trasportati come *payload* nei datagrammi IPv4 per i router che li richiedono.

5.7 Algoritmi di routing

La scelta dell'algoritmo di routing più adatto varia in base a due domande fondamentali:

1. Informazione globale o decentralizzata?

- Globale: tutti i router hanno completi "topology, link cost info". Per questa scelta è consigliato l'utilizzo di algoritmi *link state*.

- Decentralizzata: i router conoscono i vicini fisicamente connessi, "link costs to neighbors"; è previsto un processo iterativo di computazione per lo scambio di informazioni con i vicini. Per questa scelta è consigliato l'utilizzo di algoritmi *distance vector*.

2. Statico o dinamico?

- Statico: il router cambia lentamente nel tempo.
- Dinamico: il router cambia più rapidamente e periodicamente in risposta a cambiamenti di "link costs".

5.8 Tabella degli instradamenti

5.8.1 Algoritmo di Dijkstra

L'algoritmo di Dijkstra è un algoritmo routing *link state* che prevede che i costi di linking siano conosciuti da tutti i nodi. Ciò è ottenuto grazie al *link state broadcast*, grazie a cui tutti i nodi avranno le stesse informazioni.
Questo algoritmo calcola i percorsi di costo minimo da un nodo a tutti gli altri, producendo una *forwarding table* per quel nodo. È iterativo: dopo k iterazioni fornisce il percorso di costo minore per k destinazioni.

Notazione

- $c(x, y)$: costo di link dal nodo x al nodo y; corrisponde a ∞ se non sono vicini diretti.
- $D(v)$: valore corrente del costo di percorso dal nodo di partenza al nodo di destinazione v.
- $p(v)$: nodo predecessore nel percorso dal nodo di partenza al nodo v.
- N': set di nodi il cui minore costo di percorso è conosciuto.

Correttezza

Tesi: L'algoritmo di Dijkstra è corretto se eseguito su un grafo pesato diretto $G = (N, E)$ con pesi non negativi, sorgente u, funzione peso c, allora alla terminazione $D(v) = \delta(u, v)$ per ogni nodo v in N, dove $\delta(u, v)$ è la lunghezza del cammino di peso minimo tra u e v.

Algorithm 1 Algoritmo di Dijsktra

function DIJSKTRA
$N' = u$
for v ∈ net **do**
if v.isAdjacent(u) **then**
$D(v) = c(u, v)$
else$D(v) = \infty$
while $N'.isFull()$ **do** ▷ Finchè N' non contiene tutti i nodi
$find(w : (!N'.contains(v))(D(v).isMinimum()))$
$N' \leftarrow w$
$D(v) = min(D(v), D(w) + c(w, v))$ ▷ Il nuovo costo per v è uguale al vecchio costo o uguale al costo del percorso minimo per w più il costo da w a v

Dimostrazione: $D(v)$ nonè più aggiornato nel momento in cui v è inserito in N'. Dovremo quindi mostrare che $D(v) = \delta(u, v) \forall v$.
Ragioniamo per assurdo: sia x il primo nodo (nell'ordine di inserimento in N') per cui vale $D(v) \neq \delta(u, v)$ al momento in cui x è inserito nell'insieme N'.
Ne consegue che $x \neq u$, essendo il nodo sorgente u inserito nella fase di inizializzazione e valendo $D(u) = \delta(u, v) = 0$. Deve inoltre esistere un percorso di costo non infinito da u a x, dato che altrimenti il valore a cui D(x) è inizializzato (cioè ∞) sarebbe uguale a delta(u,v). Esiste quindi un percorso di costo minimo $p = u \dots v \to y \dots x$, dove y è il primo nodo sul percorso di costo minimo *non* in N' (quindi i nodi u ... v sono *tutti* in '). Il percorso p può quindi essere diviso in due percorsi: p_1, che va da u a y, e p_2, che va da y a x.
Da notare che il percorso p_1 è anch'esso il percorso di costo minimo che unisce u a y: se non lo fosse, e ci fosse un percorso p_3 che unisce u a y di costo minore del costo di p_1, allora la concatenazione di p_2 e p_2 sarebbe un percorso p' da u a x di costo minore di p, ma ciò sarebbe assurdo a causa dell'assunto iniziale secondo cui p è un percorso minimo.
Quando x è inserito in N', $D(y) = \delta(u, y)$.
Infatti, in quel momento v è stato già inserito in N' e dopo il suo inserimento y ha ricalcolato $D(y) = D(v) + c)(v, y) = \delta(u, v) + c(v, y)$, dato che per ipotesi x è il primo nodo per cui all'inserimento in N' la stima dei costi non corrisponde al percorso di costo minimo $\delta(u, y)$.
Dato che y precede x sul percorso minimo ed i pesi sugli archi sono non negativi, vale che:
$\delta(u, x) \geq \delta(u, y) = D(y)$
e quindi anche
$D(x) \geq \delta(u, x) \geq \delta(u, y) = D(y)$.

D'altro canto, dato che x viene inserito in N' prima di y, vale che:
$\delta(u, x) \leq D(x) \leq D(y) = \delta(u, y)$;

quindi
$\delta(u,x) = D(x) = D(y) = \delta(u,y)$,
cosa che porlta alla contraddizione.

Discussione La complessità dell'algoritmo per n nodi, contando che ogni iterazione necessita di controllare tutti i nodi w non in N, è $\frac{n(n+1)}{2} \approx O(n^2)$. Il costo della più efficiente implementazione possibile è $O(nlogn)$.

5.8.2 DistanceVector

Uno degli algoritmi alternativi a quello di *Dijsktra*, è noto come algoritmo di *Bellman-Ford*. Questo dice: essendo $d_x(y)$ definito come il percorso di costo minimo da x a y, allora $d_x(y) = min\{c(x,v), d_v(y)\}$, dove il min indica il minimo preso per ciascun vicino comune tra il v e quello x, *c(x,v)* è il costo per arrivare dal nodo x a quello v e $d_v(y)$ è il costo dal nodo v a quello y. In quasto caso il raggiungimento con percorso minimo di un nodo è il successivo hop nel percorso minore, usato nella tabella degli instradamenti.
Altro modo di rappresentare questa equazione è:

$$d_x(y)^{h+1} = min\{d_x(y)^h, min\{c(x,v) + d_v(y)^h\}\}$$

Quindi: devo andare da x a y, passando al più per h o esattamente *h+1* uscite. La potenza dell'equazione di *Bellman-Ford* è che non è richiesta sincronizzazione tra i vari nodi all'interno di una rete.
Genericamente, ogni nodo tiene un *distance vector* che memorizza tutti i costi del nodo verso i suoi vicini. Ogni nodo invia il suo *distance vector* ai vicini, che quando lo ricevono aggiornano il proprio utilizzando l'equazione di *Bellman-Ford*, per ogni nodo. Sotto condizioni naturali, la stima della distanza a costo minimo calcolata con questo sistema converge in quella realmente tale.
Dunque: questo sistema è *iterativo*, *asincrono* e *distribuito*. Non esiste né un inizio né una fine dell'algoritmo, perché per ogni cambiamento minimo nel *distance vector* di un nodo, questo viene propagato e notificato ad ogni suo vicino. Il punto di debolezza di questo sistema è il *count to infinity*, cioè la lentezza nel ricalcolare le stime migliori al cambio di anche un solo costo: per ogni minimo cambiamento - come l'eventuale scomparsa di un collegamento -, sono necessarie altre nuove n iterazioni prima di ritrovarci in una situazione stabile ed ottimale.

Poisoned reverse Se il nodo z decide di voler passare per y per arrivare ad x, z comunica a y che la distanza che separa z da x è pari ad infinito, così da permettere l'instradamento per y. Questo però non permette totalmente di risolvere i problemi di *count-to-infinity*.

Costi e conclusioni

Una volta calcolate le tabelle dei cammini minimi, popolo la tabella degli instradamenti, che sappiamo avere:

- Complessità dei messaggi:
  - *LinkState*: dati n nodi ed E links, ho un numero di messaggi inviati pari a $O(n \times E)$;
  - *DistanceVector*: variabile, in quanto - per ciascun nodo - avvengono scambi e interazioni con i suoi vicini.
- Robustezza:
  - *LinkState*: dati n nodi, ho una velocità di $O(n^2)$; potrebbe portare oscillazioni;
  - *DistanceVector*: variabile; potrebbe portare a loop o al problema di *count-to-infinity*.

Tutte queste soluzioni di instradamento vengono utilizzate in un solo *AS*.

5.9 Autonomous System (AS)

Sistema di aggregazione di router in diverse regioni, governato da un entità - il *gateway router* - tramite un protocollo di instradamento *intra-as* (variabile in base all'*AS* preso in considerazione). Esistono poi i protocolli *inter-as*, che si occupano di mettere in connessione differenti *AS* tramite *router di frontiera*.

- L'*intra-as* avrà la gestione degli instradamenti interni all'*AS*, per avere informazioni sul percorso minimo per raggiungere un nodo *interno* all'*AS*: si appoggia sulla necessità di presenza di criteri per rendere la gestione degli indirizzamenti il più organizzata possibile, spesso anche a scapito della performance;
- L'*inter-as* avrà la gestione degli instradamenti esterni all'*AS*, per ottenere informazioni sui percorsi utilizzati per mettere in comunicazione router appartenenti ad *AS* differneti: si appoggia sulla necessità di performance.

Esempio Mi trovo in un *AS*, l'*AS1*. Ricevo un datagramma destinato ad un nodo esterno all'*AS1*. Devo inviarlo al gateway, ma quale? L'*AS1* deve poter apprendere quali sono le destinazioni raggiungibili tramite ciascun suo *AS* vicino, e inoltre deve dare le sue informazioni delle destinazioni raggiungibili tramite l'*AS1* agli stessi, cosi da risultare raggiungibile per ricevere eventuali risposte indietro.
Tramite *inter-as* impara che una subnet x è raggiungibile tramite *AS3*, per esempio, uscendo col *gateway router 1c*: viene impiegato l'*inter-as* per propagare

le informazioni di raggiungibilità e calcolare il percorso migliore.
Ma se invece potessi apprendesse di poter raggiungere la destinazione sia tramite *AS3* che tramite *AS2*? Il router *1d* deve determinare quale dei due sia migliore da adoperare: questa scelta spetta comunque all'*inter-as*. Molto spesso si usa scegliere il più vicino (metodo *hot potato routing*).

5.9.1 Protocolli di instradamento intra-as

Protocolli noti anche come *Interior Gateway Protocols* (*IGP*). Tra i più famosi:

- *RIP*, o *Routing Information Protocol*;
- *OSPF*, o *Open Shortest Path First*;
- *IGRP*, o *Interior Gateway Routing Protocol*.

RIP Protocollo distribuito da *BSD-Unix* nel 1982, che utilizza un approccio di tipo *distance vector*.
Per calcolare il costo di raggiungibilità, conta il numero di *subnet* (numero di hop, il cui massimo imposto è 15, cosi da avere un metodo rapido per stabilire se una rotta è convenzionalmente ritenuta infinita o meno) da attraversare per arrivare a destinazione. Ad ogni di 30 secondi, viene inviato un *advertisement* (un semplice messaggio) sui costi attuali relativi a ciascun nodo: in questo tipo di messaggio posso comunicare al massimo 25 *subnet*. Qualora non si ricevesse nemmeno un *advertisement* di un dato nodo entro 180 secondi, quel nodo verrebbe definitivamente dichiarato dai vicini come morto: le rotte verrebbero invalidate.
Le tabelle di *RIP* vengono gestite a livello applicativo, e gli advertisement sono inviati tramite protocollo *UDP*.

OSPF Approccio completamente *open*, molto più recente rispetto al *RIP*, ma di tipo *link state*: è necessaria una fase di inizializzazione in cui ogni nodo deve comunicare in *broadcast* a tutti i nodi le sue informazioni.
Per comunicare le informazioni - tramite advertisement - occorre autenticarsi.
Dà la possibilità di individuare più di un percorso ottimale e di conservarlo per essere utilizzato per trasmettere dati, permettendo la gestione del *load balancing*: tramite questa funzionalità, supporta non solo l'*unicast* (la trasmissione biunivoca sorgente-destinatario), ma anche il *multicast* (permette di trasmettere da un'unica sorgente ad un gruppo di destinazione).
Non è *flat*, ma gerarchico, così da ovviare a problemi derivanti dalla grandezza di un'eventuale struttura: si divide in n aree, ciascuna gestita da un *router di bordo*, che - oltre a catalogare informazioni di instradamento e raggiungibilità relative a quell'area - si trova in un'area speciale, detta *area di backbone*, che mette in comunicazione tutti i *router di bordo* con il *router di backbone*.

5.9.2 Protocolli di instradamento inter-as

Senza ombra di dubbio, il protocollo *de facto inter-as* più famoso e utilizzato è il *BGP* (o *Border Gateway Protocol*).

BGP Protocollo che non abbraccia propriamente né l'approccio *link state*, né quello *distance vector*. Definisce il suo approccio come *path vector*, in quanto usa dichiarare unicamente l'insieme di vettori che mettono in comunicazione diversi *AS*.
Utilizza due componenti:

- *eBGP*: per comunicare ai diversi *AS* la raggiungibilità di destinazioni possibili tramite il mio *AS*;
- *iBGP*: per propagare la raggiungibilità a tutti i router interni all'*AS*;

Due router *BGP* (due *peers*), per scambiarsi messaggi *BGP* (atti alla scoperta dei percorsi possibili per raggiungere diversi prefissi di rete), richiedono l'apertura di una conessione *TCP*, una sessione dedicata.
Inoltre, con *BGP*, quando un *AS* comunica la raggiungibilità di un dato prefisso di rete, prende l'impegno di poter gestire il traffico derivato dalla richiesta di connessioni vesro quella destinazione.
Infine, utilizza due attributi fondamentali:

- *AS-PATH*: contiene gli *AS* attraverso cui è passato l'advertisement del prefisso di rete;
- *NEXT-HOP*: indica il router interno specifico sull'*AS* di *next-hop* (vicino, adiacente).

5.9.3 Topologia logica inter-as

Esistono diversi metodi per diffondere in broadcast i nodi raggiungibili: lo *spanning tree* è un metodo che, a partire da un nodo della rete, costruisce un albero, la cui radice è il nodo di partenza stesso. Esistono diversi algoritmi (*Prim* o *Kruskal*), utilizzati per cercare di ammortizzare il costo di costruzione dello stesso.
Ogni nodo invia un messaggio di *join* in *unicast* al nodo centrale, che viene reindirizzato fino a che non raggiunge un nodo (già) appartiene allo *spanning tree*.
Molte applicazioni richiedono il trasferimento di pacchetti da uno o più mittenti ad un gruppo di destinatari (e.g., per il trasferimento di aggiornamenti, per lo streaming, per applicazioni con dati condivisi, per aggiornamento di dati, per giochi multiplayer, ...). L'indirizzo che rappresenta un gruppo multicast è un indirizzo IP multicast di classe D, dentro il quale si possono affiliare diversi utenti.

IGMP (Internet Group Management Protocol)

Utilizzato a livello di trasporto: i messaggi sono incapsulati in datagrammi IP, mandati con *TTL* a 1 (perché si tratta di uno scambio tra endpoint e router di accesso delegato della gestione multicast). Sono composti da un tipo (da 8bit: *query* (richiesta dal router), *membership report* (risposta dagli host) o *leave group*), un tempo di risposta, un *checksum* e un *group address*. I router eseguono un refresh ogni minuto per verificare che i propri host non siano interessati a eventuali cambiamenti del gruppo multicast a cui sono affiliati.
Come calcolare le rotte di raggiungimento da una sorgente ad una destinazione? Come calcolare i percorsi migliori? Si può costruire - alla sorgente - un albero di copertura utilizzando il *reverse path forwarding*, appoggiandosi a quanto noto al router per i percorsi minimi di unicast. In questo caso, occorre - per essere rimosso completamente dall'albero - un messaggio di *pruning* al router delegato, per accertarmi di non essere più raggiunto per ogni richiesta dei membri multicast.
L'altro approccio è il *center-based tree*, indipendente dalla sorgente. Viene utilizzato un albero di consegna unico e comune per tutti: prendo un nodo di riferimento, *centro* dell'albero.
Nella pratica, vengono utilizzati entrambi gli approcci:

- *DVMRP* (o *Distance Vector Multicast Routing Protocol*): utilizza un approccio *flood and prune* e *reverse path forwarding*. Ad ogni minuto, *dimentica* gli endpoint esclusi, per cui risulta poco performante se sono pochi gli host inclusi nel gruppo multicast per le troppe richiestte di pruning;
- *PIM*: (o *Protocol Indepenetent Multicast*): non dipende da nessun algoritmo di routing di unicast. Due metodi di distribuzione in multicast: *dense* e *sparse*.

Capitolo 6

Livello di Link (DataLink)

Come si trasmettono le informazioni da un router all'altro fino agli endpoint è opera del livello *datalink*.
Questo livello si appoggia sul concetto di *trama*, un percorso seguito dal flusso, che inizia e finisce su due estremi. Offre diversi servizi:

- rilevamento e correzioni di errori;
- condivisione di canali broadcast, offrendo accessi multipli;
- servizio di gestione degli indirizzi di livello link;
- trasnferimento dati affidabili, con controllo del flusso.

Il livello di rete è implementato sugli *adaptor*, che comunicano l'uno con l'altro - stabilendo una trama per raggiungersi - e che devono rispettare gli standard appena esposti.
Accetta soltanto degli stream di raw bit (flussi di sequenze di bit) e cerca di consegnarlo a destinazione: ciononostante la comunicazione non è necessariamente priva di errori. Per gestire un numero multiplo di flussi di informazioni, divide lo stream in un numero variabile di trama (*framing*) e elabora il checksum per ogni trama (per ovviare a problemi di generazione e correzione errori).
Lo split viene applicato ogni n caratteri, e ciascun flusso è delimitato da dei caratteri speciali: *DLE STX* (*Data Link Escape Start of TeXt*) e DLE ETX (*Data Link Escape End of TeXt*). Qualora dovesse essere inviata esattamente la stringa utilizzata come escape, viene aggiunto un *DLE* per eliminare ambiguità.
Possono essere poi utilizzati degli approcci meno generici, come il sistema *Manchester*.

6.1 Rilevamento di errori

Tra i dati trasmessi, vi sono due blocchi fondamentali:

1. *EDC*: bit - ridondanti - di *error detection and correction*;

2. D: dati protetti dal controllo di errori - eventualmente accompagnati da campi header.

Ovviamente la rilevamento di errori non è affidabile al 100%: si potrebbero raramente non rilevare alcuni errori, in maniera inversamente proporzionale alla dimensione alla dimensione dell'*EDC* utilizzata.
Nel rilevamento di errori, si utilizza il calcolo della distanza di *Hamming*, attraverso cui è possibile determinare in quanti bit differiscano due parole di codice. Sostanzialmente, si esegue lo *XOR* tra le parole e poi si conta il numero di 1 nel risultato: il numero di posizioni nelle quali le due parole di codice differiscono determina la loro distanza di *Hamming*. Se due parole hanno distanza di *Hamming* d, ci vorranno d operazioni sui singoli bit per tramutare una parola di codice nell'altra: per come sono utilizzati i bit di ridonanza, se la lunghezza delle parole di codice è $n = m + 4$, sono possibili 2^m messaggi dati validi, ma non tutte le 2^n parole di codice. Dunque la distanza di *Hamming* di un codice è la minima distanza di *Hamming* tra le due parole di codice. Ancora, per rilevare d errori occorre un codice con distanza di *Hamming* $d + 1$, mentre per correggerli serve un codice con distanza di *Hamming* $2d + 1$.

6.1.1 CRC (Cyclic Redundancy Check)

Vengono visti i bit, D, come un numero binario. Viene scelto un polinomio generatore di grado $r + 1$ (una sequenza binaria in cui il primo elemento è sempre 1). L'obiettivo è trovare un numero r di bits *CRC* di ridondanza R: viene calcolato così che $< D, R >$ (concatenazione) sia esattamente divisibile per G (in modulo 2). Il ricevente conosce G, e divide $< D^1, R^1 >$ per G e controlla che il resto di questa operazione sia proprio 0, il che gli garantisce che l'informazione non contenga errori.
Questo approccio è molto utilizzato in internet.

6.2 Protocolli di accesso multiplo

Due tipologie di link:

- *punto-punto* (sorgente-destinazione): due nodi collegati da un link link (come nel caso dei *dialup PPP* e *link ethernet*). Nessun rischio di collisione;
- *broadcast*: il mezzo trasmissivo viene condiviso tra più nodi (come nel vecchio ethernet con topologia a *bus*, nell'*HFC*, nelle reti satellitari e nell *802.11 wirelless* - wifi). Aperto a rischio di collissioni.

Se tutti i nodi sono liberi di contattare un mezzo trasmittivo, ci si apre al rischio di eventuali collisioni, motivo per cui è necessario introdurre delle regole. Occorre inoltre gestire il problema delle diverse potenze di trasmissione, noto come *capture effect*: se c'è una grossa differenza di potenza tra due trasmissioni distinte, quello che trasmette con minore potenza non compromette completamente il primo; infatti, la sua trasmissione diventa sostanzialmente un *rumore di fondo*, nonostante potrebbe rappresentare un caso di collisione.

6.2.1 Mezzo condiviso

Un protocollo di accesso multpilo è un protocollo che regolamenta l'utilizzo del mezzo trasmissivo per evitare fenomeni di collisione.
Le caratteristiche desiderabili sarebbero:

- Data un'unica stazione di trasmissione, questa possa avere il massimo della capacità trasmissiva;
- Dato un numero variabile di nodi, questi possano avere una distribuzione equa della potenza di trasmissione dedicata a ciascuno;
- Un approccio decentralizzato: non voglio ci sia un elemento a coordinare la trasmissione e che non ci sia la necessità di sincronizzazione di orologi (perché ha un costo);
- Sia il più semplice possibile.

Protocolli MAC effettivamente proposti:

- *Channel Partitioning*: si divide il canale in sezioni, ognuna dedicata a ciascun nodo
  - *TDMA* (*Time Division Multiple Access*): dispositivi hanno una visione comune del tempo, diviso in *trame* (divise in tanti slot quanti sono i nodi che possono utilizzare la stessa stazione di trasmissione). All'interno del tempo di trama viene data la possibilità a un nodo di trasmettere con uno slot di una trama. Richiede sincronizzazione e occorre una porzione di banda detta *banda di guardia*, utilizzata per separare le date sezioni;
  - *FDMA* (*Frequency Division Multiple Access*): la banda viene divisa in tante sezioni quanti sono gli utenti, in modo tale che ciascuno abbia le sue risorse, ma equa capacità di banda.

  Sicuramente si tratta in entrambi i casi di una divisione equa, una soluzione - non ideale - di struttura decentralizzata e i protocolli sono semplici, ma non è possibile trasmettere a potenza massima
- *Random Access*: non diviso, permette collisioni:
  - *Slotted ALOHA*: la trasmissione viene divisa in frame di tempo uguali, ancora divisi in slot; se due o più nodi trasmettono in uno stesso slot, si verifica una collisione. Se si tratta di un frame pulito (nuovo, ancora inutilizzato), si trasmette nel primo slot: se c'è una collisione, si trasmette nel primo slot del prossimo frame. Mentre quando si tramsette lo si fa a potenza massima e per quanto si tratti di un algoritmo semplice, è soltanto parzialmente efficace, perché si verificano troppo spesso collisioni e troppo tempo viene speso senza effettiva trasmissione. Secondo un'analisi sull'efficienza, nel caso migliore, trasmette nel 37% del tempo.

- *Pure (unslotted) ALOHA*: se c'è qualcosa da trasmettere, viene fatto senza precauzioni, per un arco di tempo fisso. Se si presenta una collisione aspetto un arco di tempo fisso e poi avviene un nuovo tentativo. Il problema è che, mentre non c'è più sincronizzazione sulla cadenza tra uno slot e l'altro, si presenta però una sovrapposizione tra trasmissioni di diversi nodi. In questo caso, secondo un'analisi sull'efficienza, risulta trasmettere - nel caso migliore - nel 18% del tempo.
- *CSMA* (*Carrier Sense Multiple Access*): una nuova estensione dell'*ALOHA*. In questo caso, prima di trasmettere si verifica che non ci sia nessun altro nodo in trasmissione. Se così fosse, si ritarda la trasmissione. Ovviamente una collissione si può comunque verificare a causa del tempo di propagazione: si tratta del caso in cui qualcun altro comincia a trasmettere nell'intervallo di tempo che intercorre tra il controllo che il mezzo trasmissivo sia in *idle state* (quindi nessuno sta trasmettendo) e l'effettivo invio dei dei dati da trasmettere. Si tratta di una probabilità di collissione molto più bassa.
- *CSMA/CD* (*Carrier Sense Multiple Access con Collision Detection*): si sta in ascolto se qualcuno sta già trasmettendo (*carrier sensing*). Sostanzialmente, si ascolta per verificare che non ci siano eventuali collisioni, e in tal caso si provvede immediatamente a stoppare la trasmissione e ritardarla. Il tempo di tra un abort di trasmissione a causa di una collisione e una nuova trasmissione viene scelto randomicamente selezionando in un intervallo tra in 0 e 2^{m-1}, dove m è il numero di ritrasmissione, in modo tale che per ogni ritrasmissione, il tempo di *wait* raddoppi, rispetto all'ultima *wait*. L'efficienza del protocollo *CSMA* è tanto più bassa quanto più è alto il tempo di propagazione: decisamente più performante dell'*ALOHA*, più semplilce e decentralizzato.

- *Taking Turns*: a turni, ogni nodo può accedere alla stazione trasmissiva (alcuni potrebbero tenere occupata la stazione più tempo). Si tratta di un approccio a *polling*: c'è un *master node* che gestisce la coda degli *slave nodes*, per trasmettere - a turno - quanto da loro desiderato.

MAC e ARP A differenza dell'indirizzo *IP*, l'indirizzo *MAC* è fisso, perché tipicamente scritto direttamente nell'adattatore di rete utilizzato e si tratta di una sequenza a 48 bit. Viene utilizzato perché possa non esistere alcuna corrispondenza tra l'indirizzo *MAC* e la rete a cui l'adattatore risulta connesso, a differenza dell'indirizzo *IP*. Tutto ciò per rendere il nodo di rete quanto più unicamente qualificabile e rintracciabile in una qualsiasi rete, a prescindere dalla rete stessa.
L'allocazione dell'indirizzo *MAC* è gestita da *IEEE*: ogni società manufatturiera compra una porzione di indirizzo *MAC* per garantirgli l'unicità.
Questo indirizzo viene utilizzato nel livello di rete per raggiungere - da una sor-

gente - una destinazione. Per fare questo viene utilizzato il protocollo *ARP* (*Address Resolution Protocol*): ogni dispotivio nell'adattatore mantiete una *ARP Table*, che associa ogni indirizzo *IP* al corrispettivo indirizzo *MAC* per ciascun nodo di rete, secondo la tripla: $< ip; mac; ttl >$ (la *TTL* è tipicamente settata a 20 minuti). Qualora la *ARP Table* non contenga l'associazione dell'*IP* che ho la necessità di contattare, viene eseguita una *query ARP* in *broadcast* sulla *LAN*: la risposta ottenuta viene salvata nella cache (per un tempo massimo definito dalla *TTL* definita nella tripla ottenuta).
Per ogni adattatore di rete in un nodo viene conservata una *ARP Table* diversa.

6.3 Ethernet

Nei primi anni, veniva utilizzata la *topologia a bus*: i nodi si trovavano tutti nello stesso dominio di collisione (potendo, quindi, collidere ciascuno con l'altro). Col passare del tempo è stata introdotta una struttura topologica migliore, la *topologia a stella*: vi è uno switch al centro, che lega tutte le interfaccie di ciascun nodo a sé stesso, utilizzando un canale separato per ciascuno ed evitando quindi eventuali collisioni tra i vari nodi.
La struttura di una trama Ethernet è semplicissima:

- *preamble* (da 7 byte): utilizzato per garantire la sincronizzazione tra trasmettitore e ricevitore;
- *indirizzo sorgente* e *indirizzo destinazione*: indirizzi *MAC*, estremi della trasmissione;
- *type*: indica il protocollo da utilizzare nel livello superiore (generalmente *IP*, ma possibili molti altri: e.g. *Novel IPX*, *AppleTalk*, ...);
- *crc*: campo per detecting di errori ad opera del ricevitore (se vengono trovati errori, il frame viene scartato).

Ethernet è *connectionless*, *unreliable* e utilizza l'algoritmo *CSMA/CD*.

6.4 Switch

Uno *switch* è un dispositivo di rete che si occupa di commutazione. Agisce sull'indirizzamento e sull'instradamento all'interno delle reti LAN mediante indirizzo fisico, selezionando i frame ricevuti e dirigendoli verso il dispositivo corretto. Presenta le seguenti caratteristiche fondamentali:

- permette trasmissioni multiple simultaneamente
- consente buffer di pacchetti
- gli host hanno connessioni dedicate dirette con lo switch
- il protocollo ethernet è usato su *ogni* link entrante, ma senza collisione (full duplex)

Internamente, uno switch è costituito da una o più schede munite di porte. Ad ogni porta può essere connesso un nodo, che può essere una stazione, un altro switch, un hub o un altro dispositivo di rete. Quando un nodo A cerca di comunicare con un nodo B il comportamento dello switch dipende dalla scheda a cui è collegato B:

- se B è collegato a una porta sulla stessa scheda di A, la scheda stessa inoltra i frame in arrivo su tale porta;
- se B è collegato a una scheda diversa da quella a cui è collegato A, la scheda invia i frame a un canale di trasmissione interno detto *backplane*, caratterizzato da elevata velocità (tipicamente sull'ordine del Gbps), che provvede a consegnare il frame alla scheda giusta.

Per l'instradamento, gli switches contengono al loro interno una **switch table** che consente la memorizzazione di entries contenenti un MAC address e una porta. È importante notare come questo dispositivo non necessiti di nessuna configurazione; è dotato di un meccanismo di *self-learning* per la creazione automatica delle entries.
Quando il nodo A deve trasmettere, nel momento in cui il frame arriva allo switch, questo legge l'indirizzo di A e la porta su cui è arrivato il frame; memorizza sulla tabella le informazioni e registra un *ttl*, fondamentale nel caso in cui il nodo abbia cambiato porta o non sia più disponibile per qualsiasi motivo (MAC = A, interface = 1, TTL = 60).
Se il nodo B, a cui dev'essere inviato il frame, non è presente tra le entries della switching table, lo switch invia il frame in *flagging* ai suoi nodi, e ricorsivamente se incontra altri switch: alla risposta di B, registrerà la entry mancante nella table. Nel caso in cui un qualsiasi nodo si colleghi ad un altro switch, poichè è essenziale che a livello di routing si sappia che ha cambiato subnet, sarà il nodo stesso ad inviare un pacchetto ARP (cfr. *ARP table*) per aggiornare la tabella.

Switches VS Routers
Pur essendo entrambi dispositivi *store-and-forward*[1], switches e routers presentano importanti differenze. Per prima cosa, i router sono dispositivi che agiscono a livello di rete, a differenza degli switches che agiscono a livello datalink.
Va notato che, pur essendo gli switches dei dispositivi *'plug-and-play'* (mentre i router devono essere configurati), il loro utilizzo causa un aumento di overhead proporzionale all'aumento delle dimensioni della rete: oltre al broadcast dovuto dall'utilizzo di ARP, gli switches stessi utilizzano il flagging come tecnica per ovviare all'assenza di entries sulle loro table; il risultato di una tempesta broadcast in una rete mal strutturata di medie dimensioni potrebbe essere disastroso sia per integrità e velocità di trasmissione che per la sicurezza e la privacy.

Per evitare *broadcast storm*, algoritmi di *spanning tree* rendono gli switch in grado di 'staccare' i link che potrebbero creare problemi e redirezionano il traffico utilizzando un altro link posto appositamente nella rete per creare ridondanza nei collegamenti; tale link rimarrà attivo solo il tempo necessario alla riparazione del link danneggiato.

VLAN Per motivi di privacy e controllo dello scambio di dati, è stato introdotto un meccanismo che consente di "etichettare" le porte degli switches, raggruppandole in lan diverse, disconnesse l'una dall'altra: dopo la configurazione di queste "etichette", è necessario solo passare da una porta all'altra (o addirittura solo cambiare l'etichetta della porta) per passare da una lan all'altra.
È inoltre possibile il forwarding tra VLANs:gli switches che lo supportano hanno una specie di piccolo router integrato che all'occasione può mettere in comunicazione le lan. Alla presenza di moltiplici switches con diversi insiemi di VLANs, vengono messe a disposizione le **trunk ports**: porte che non appartengono a nessuna vlan e servono unicamente per mettere in collegamento due switch. Ovviamente occorre che un pacchetto rechi su di sè informazioni sulla VLAN di appartenenza: per questo motivo gli switch allargano l'header di ogni frame passante per una trunk porte inseriscono le informazioni necessarie.

[1]Tecnica in base alla quale un'informazione (suddivisa in pacchetti), nel suo percorso tra le singole stazioni (o nodi) della rete, deve esere totalmente ricevuta prima di poter essere ritrasmessa nel collegamento in uscita.

6.5 Point-to-Point Data Link Control

Garantisce:

- *packet framing*: prende i pacchetti a qualsiasi livello superiore e li inserisce in uno stream di bit.
- *bit transparency*
- *error detection*
- *connection liveness*: detect signal link failure to network layer
- network layer address negotiation: endpoint can learn/configure each other's network address

Non fornisce:

- error connection
- flow control
- ordine trasmissione di pacchetti
- supporto link multipoint (polling)

Data Frame Composto da una flag di inizio, un indirizzo, un byte di controllo, il protocollo utilizzato, l'informazione, un altro check ed una flag di fine.
Se si trasmette uno stream ad un'altra scheda potrebbe campionare bit in maniera sbagliata; è necessario sincronizzare trasmettitore e ricevitore continuamente durante la ricezione di bit. Per fare questo si ricorre al *byte stuffing*. Ogni volta che nel pacchetto si trova una sequenza (fissa, preambolo) 01111110 mette un extra 01111110; il ricevente scarta il secondo byte e continua la ricezione.

Il meccanismo di trasmissione è abbastanza semplice.

- link establishment - configurazione, viene decisa la dimensione massima del frame, definita fase di autenticazione etc.
- configurazione livello di rete
- apertura: scambio di dati
- terminating
- dead

Capitolo 7

A day in the life of a web request

Scenario: uno studente collega il laptop alla rete del campus e richiede/riceve www.google.com

1. Tipicamente non ha nessuna configurazione: parte il protocollo DHCP; ricevi indirizzo IP, gateway, subnetmask, DNS. Il protocollo viaggia su UDP.
   $DHCP \rightarrow UDP \rightarrow IP \rightarrow Eth \rightarrow Phy$

2. DHCP server: tipicamente sul primo router. Vede un frame in broadcast e sale tutta la pila protocollare. Reincapsulato a sua volta e torna indietro fino all'utente; ora anche lo switch ha imparato dove inoltrare i pacchetti. La scheda di rete è configurata.

3. Mandata richiesta per: "www.google.com". Serve il DNS. Si riparte col DNS: crea un altro pacchetto udp, lo mette su un pacchetto ip, lo piazza su ethernet e lo mette su phy. Mandare una query arp in broadcast perchè hai bisogno di sapere dov'è il router! questo risponde e dice "il mio indirizzo MAC è questo qua."

4. il datagramma IP viene inviato al server DNS. Questo: - ha nella sua cache l'indirizzo IP di google e lo invia - non ha indirizzo ip di google. WTF?! Va sulla gerarchia di DNS fino al dns autoritativo e ritorna la risposta.

5. Si parte col TCP. A questo punto , inviate un syn al server per instaurare una connessione. SYNACK. Parte la richiesta HTTP. 3 way handshake inviato contestualmente ai dati. Viaggia dentro la socket tcp, arriva a google, google la legge e risponde.

6. 404, not found. Fine.

Capitolo 8

Sicurezza

8.1 Informazioni generali

La sicurezza informatica si fonda su diversi concetti:

- **confidenzialità**: soltanto *sender* e *receiver* possono *capire* il contenuto di un messaggio;
- **autenticazione**: soltanto *sender* e *receiver* possono confermare l'identità di ciascuno;
- **integrità di messaggio**: *sender* e *receiver* devono essere sicuri che il messaggio sia integro;
- **accesso e disponibilità**: i servizi devono essere accessibili e disponibili agli utenti.

Personaggi noti *Bob* e *Alice* vogliono comunicare in maniera sicura; *Eve* è l'intruso, che vuole ottenere - senza permesso - le informazioni tra i due precedenti comunicanti, tramite *eavesrop* (intercettazione), *iniettazione* dei messaggi nella rete, *personificazione* (pretendendo di essere qualcun altro); *hijacking* (sostituendo uno dei capi della trasmissione con sé stesso), provocazione di *denial of service* (prevenendo il servizio dall'essere usato dagli utenti, attraverso l'*overload* delle risorse, per esempio).

8.2 Crittografia

Il messaggio in chiaro si dice *plain-text*; una volta crittografato, invece, *cipher-text*.
Avendo soltanto il *cipher-text* posso:

1. fare *brute-force*: tentare utilizzando tutte le chiavi in mio possesso per *decriptare* il messaggio;
2. utilizzare un'analisi statistica per risalire attraverso calcoli probabilistici a sequenze di caratteri noti (nel caso in cui si utilizzino metodi di cifrature *monoalfabetici*).

8.2.1 Crittografia a chiave simmetrica

Due o più persone condividono la stessa chiave (simmetrica), utilizzata come pattern ci cifratura a sostituzione mono alfabetica. Come si accordano, però, Bob e Alice? Occorre un sistema di scambio di chiavi attraverso un canale sicuro (anche fisico, per esempio).

Schema di cifratura di Cesare

Sistema di cifratura basato su uno spostamento/slittamento di n lettere sulla sequenza dell'alfabeto. In questo caso, la chiave è proprio n.

DES: Data Encryption Standard

Standard di criptazione, che utilizza chiavi lunghe *56bit*, di cui si usano *48bit*. Il messaggio viene spezzato in frammenti da 64bit, e passa per l'algoritmo blocco per blocco, tramite sistemi di permutazione. E' stato necessario nemmeno un giorno per *bucarlo*.
Per questa ragione hanno sviluppato il *3DES*, algoritmo di criptazione che prevede 3 passaggi: una *encryption* iniziale utilizzando una chiave iniziale, una *decription*, utilizzando una seconda chiave, e una *encryption* finale, tramite la chiave iniziale.

AES: Advanced Encryption Standard

Standard molto più attuale. Utilizza un sistema di criptazione simmetrica che nel 2001 ha rimpiazzato *DES*. Processa dati in blocchi da *128bit*, con chiavi da *128/192/256 bit*, per cui si dice abbia sicurezza di un ordine di 2^{128}.
Mentre occorreva 1s per decriptare lo standard *DES*, occorrono 149 trilioni di anni per farlo con *AES*.

8.2.2 Modus operandi

Il *modus operandi* di un algoritmo di cifratura è il modo con cui vengono scambiati i blocchi, e prende il nome di *ECB* (o *Elecronic CodeBook*). Altro metodo è quello del *CBC* (o *Cipher Block Chaining*), dove l'output del blocco precedente è l'input del blocco successivo.

8.2.3 Crittrogafia a chiave pubblica

Utilizza un approccio totalmente differente: il *sender* e il *receiver* hanno rispettivamente due chiavi: una privata e una pubblica. Non condividono la loro chiave privata, ma solo la pubblica, quindi la chiave di decriptazione privata è nota solo al *receiver*.
Sostanzialmente, quando Alice vuole inviare un messaggio a Bob, lo cripta con la chiave pubblica di Bob, il quale - quando riceverà il messaggio criptato - lo decripterà con la propria chiave privata, e/o viceversa.
Tra gli algoritmi di crittografia a chiave pubblica più importanti troviamo l'*RSA* (dai suoi creatori: *Rivest*, *Shamir* e *Adelson*), che si basa sul fatto che bit può essere univocamente rappresentabile da un numero. Per creare un paio di chiavi *RSA* occorre:

- scegliere due numeri p e q (utilizzando, ad esempio, 1024 bit per ciascuno);
- si calcola $n = p \times q$ e $z = (p-1) \times (q-1)$;
- si sceglie una $e < n$ che non ha nulla in comune con z (per cui si dicono *relativamente primi*);
- si sceglie una d in modo tale che $(e \times d) - 1$ sia divisibile per z.

Infine:

$$\text{Criptazione: } c = m^e \ mod \ n$$
$$\text{Decriptazione: } m = c^d \ mod \ n$$

8.3 Autenticazione

8.3.1 Firme digitali

Una firma digitale rappresenta un insieme di dati in forma elettronica, allegati oppure connessi tramite associazione logica ad altri dati elettronici, utilizzati come metodo di identificazione informatica.

Si propone di soddisfare le seguenti esigenze:

- **autenticazione:** il destinatario può verificare l'identità del mittente.
- **integrità:** il destinatario non può creare o modificare un documento firmato da qualcun altro.

- **non ripudio:** il mittente non può disconoscere un documento da lui firmato. La firma digitale è l'unico meccanismo basato su crittografia a chiave pubblica ad avere questa caratteristica.

Un tipico schema di firma elettronica basato sulla tecnologia della chiave pubblica consiste di tre algoritmi:

1. Un algoritmo che genera una coppia di chiavi (PK, SK), dove PK è la chiave pubblica di verifica della firma e SK è la chiave privata utilizzata per firmare il documento.

2. Un algoritmo di firma che prende in input un messaggio m e una chiave privata SK, calcola il codice hash del messaggio e, crittografandolo con SK, produce una firma σ.

3. un algoritmo di verifica che prende in input la tripla (m, PK, σ) e accetta o rifiuta la firma.

La firma digitale è l'unico caso in cui l'uso delle chiavi è invertito: la chiave pubblica serve a decrittare la firma e trovare poi il *digest* iniziale attraverso l'hash, mentre la chiave privata serve a crittografare una stringa anzichè aprirla.

8.3.2 Message digests

Per dare una sorta di *impronta digitale* di un'entità vengono utilizzati i cosiddetti **message digests**: partendo da un insieme enorme di combinazioni applicabili su ogni tipo di file digitale, una funzione di *hashing* fa corrispondere un'entità a (quasi) una ed una sola sequenza di caratteri. In questo modo è possibile determinare una firma univoca per provare l'integrità di ogni file, potendo garantire che questa sia tale soltanto per quel file. La lunghezza dei valori di hash varia a seconda degli algoritmi utilizzati; il valore più comunemente adottato è di 128 bit, che offre una buona affidabilità in uno spazio relativamente ridotto[1]. La sicurezza di questo sistema è data dal fatto che ogni *hash* è dato da una stringa di lunghezza fissa indipendentemente dalla lunghezza del messaggio.
La firma digitale pul essere definita come *digest* di un documento crittografato con chiave privata.

Checksum

La *checksum* fa una sorta di *hash* come è stato appena introdotto: produce una stringa da 16 bit. Il problema in questo caso riguarda il fatto che è molto più semplice trovare file diversi che abbiano lo stesso *checksum*.

A questo punto possiamo arrivare alla combinazione ottimale: Bob manda un pacchetto di informazioni contenente:

[1] Va registrata la possibilità d'uso di hash di dimensione maggiore (SHA, ad esempio, può anche fornire stringhe di 224, 256, 384 e 512 bit) e minore (anche se fortemente sconsigliato).

- messaggio puro;
- hash del messaggio, crittato con la chiave privata dello stesso Bob.

A questo punto, Alice, per verificare l'autenticità del messaggio, fa l'hash del messaggio e lo confronta con l'output della decrittazione tramite chiave pubblica di Bob dell'hash crittato precedentemente da Bob con la sua chiave privata. Se questi corrispondono, allora il messaggio risulta autentico.

Attualmente esistono diversi tipi di hash, in base al livello di sicurezza: leggermente datato è l'*MD5* (da 128 bit, crittato in 4 passi), più aggiornato è invece lo *SHA-1* (da 160 bit, ancora più sicuro è invece lo *SHA-2*).

8.3.3 CA (Certification Authority)

La *CA* rappresenta l'autorità che assegna un *certificato digitale* (comprensivo dei dati di Bob con la sua firma digitale) ad una specifica entità E appartenente a Bob sulla base della chiave pubblica di E, in modo tale che sia garantita l'identità di E presso la *CA*. Sostanzialmente, le informazioni del certificato digitale vengono prodotte dalla *CA*, crittate con la chiave pubblica di Bob e poi con la chiave privata della *CA*.

8.3.4 Sicurezza e-mail

Alice vuole mandare una email confidenziale a Bob:

1. genera una chiave privata simmetrica randomica, K_s;
2. critta il messaggio con questa chiave K_s;
3. critta l'output con la chiave pubblica di Bob;
4. invia l'output e la chiave generata randomicamente a Bob, crittata ancora con la chiave pubblica di Bob.

Bob, quindi, quando riceve la email, decritterà il messaggio e la chiave segreta generata randomicamente da Alice con la sua chiave privata e - ancora - decritterà il messaggio in definitiva utilizzando la chiave-output della decrittazione della chiave segreta generata randomicamente da Alice.
In alternativa:

1. Alice fa un *hash* del messaggio;
2. impacchetta il messaggio in chiaro e la firma digitale (come già visto con i message digests);
3. il pacchetto viene crittato con una chiave privata simmetrica randomica, K_s;
4. viene mandato il pacchetto e la chiave generata randomicamente a Bob, crittata ancora con la chiave pubblica di Bob.

Bob, quindi, quando riceve la email, decritterà il messaggio e la chiave segreta generata randomicamente da Alice con la sua chiave privata e - ancora - decritterà il messaggio in definitiva utilizzando la chiave-output della decrittazione della chiave segreta generata randomicamente da Alice. Bob, per verificare l'autenticità del messaggio, fa l'hash del messaggio e lo confronta con l'output della decrittazione tramite chiave pubblica di Alice dell'hash crittato precedentemente da Alice con la sua chiave privata. Se questi corrispondono, allora il messaggio risulta autentico.

8.4 Secure Socket Layer

Con l'apertura della Rete a fini pubblici, le problematiche di prevenzione di danni, perdite e attacchi via via sempre più importanti portarono alla nascita di strati aggiuntivi con lo specifico compito di assicurarsi la delle comunicazioni. Le prime implementazioni di SSL erano limitate a una cifratura a chiave simmetrica di soli 40 bit[2]; fu succesivamente aggiornato a TLS (*Transport Layer*

[2]Tale limitazione fu dovuta da restrizioni imposte dal governo statunitense sull'esportazione di tecnologie crittografiche. Fu esplicitamente imposta per rendere la cifratura abbastanza debole da poter essere forzata dalle autorità giudiziarie che volessero decifrare il traffico, ma sufficientemente resistente ad attacchi esterni poichè facente affidamento alle minori disponibilità di risorse tecnologiche e finanziarie dell'epoca.

Security), con chiavi da 128+ bit.

Il protocollo TLS consente alle applicazioni client/server di comunicare attraverso una rete in modo tale da prevenire il *tampering*(manomissione), la falsificazione e l'intecettazione dei dati.
Nell'utilizzo tipico del browser di un end-user, l'autenticazione è *unilaterale*: è il solo server ad autenticarsi presso il client. Il browser valida il certificato del server controllando che la firma digitale dei certificati sia valida e riconosciuta da una *certification authority* conosciuta utilizzando una cifratura a chiave pubblica. Dopo questa autenticazione, il browser indica una connessione sicura.
Quest'autenticazione, però, non è sufficiente per garantire che il sito con cui ci si è collegati sia quello richiesto. Per esserne sicuri è necessario analizzare il contenuto del certificato rilasciato e controllarne la catena di certificazione. I siti che intendono ingannare l'utente non possono utilizzare un certificato del sito che vogliono impersonare, perchè non hanno la possibilità di cifrare in modo valido il certificato, che include l'indirizzo, in modo tale che risulti valido alla destinazione.
Il protocollo TLS permette anche un'autenticazione *bilaterale*, tipicamente utilizzata in applicazioni aziendali, in cui entrambe le parti si autenticano in modo sicuro scambiandosi i relativi certificati. Questa autenticazione (definita Mutual authentication), richiede che anche il client possieda un proprio certificato digitale.

8.4.1 Principi di funzionamento

Il funzionamento del protocollo TLS può essere suddiviso in tre fasi principali:

1. Negoziazione dell'algoritmo da utilizzare;
2. Scambio delle chiavi e autenticazione;
3. Cifratura simmetrica e autenticazione dei messaggi.

Nella prima fase, il client e il server negoziano il protocollo di cifratura che sarà utilizzato nella comunicazione sicura, il protocollo per lo scambio delle chiavi, il MAC (*message authentication code*) e l'algoritmo di autenticazione. L'algoritmo per lo scambio delle chiavi e quello per l'autenticazione normalmente sono a chiave pubblica; l'integrità dei messaggi è garantita da un algoritmo di hash che utilizza funzioni pseudorandom non standard.

Protocolli. All'interno di una sessione vengono tipicamente utilizzati i seguenti protocolli:

- RSA, Diffie-Hellman, ECDH, SRP, PSK per lo scambio di chiavi;
- RSA, DSA, ECDSA per l'autenticazione;

- RC4, DES, 3DES, AES, IDEA o Camellia (anche RC2 nei vecchi SSL) per la cifratura simmetrica;
- HMAC-MD5, HMAC-SHA (TLS); MD5, SHA (SSL) come funzioni di hash per le funzioni crittografiche d'integrità.

ARP Una richiesta *ARP* consiste in una richiesta di risoluzione inversa: invece di chiedere la risoluzione di un nome host, richiediamo la risoluzione inversa, ovvero quella che associa univocamente un IP ad un nome host. Il protocollo *ARP* lavora a livello *datalink* della pila *TCP/IP*, quindi utilizzando come nomi identificativi gli indirizzi *MAC*. Per la richiesta viene richiamato il gateway dell'IP richiesto, il quale fa una richiesta in *broadcast* nella subnet per cercare il dato IP.
In un attacco *arp-spoofing* (o *arp-poisoning*), l'attaccante fa un attacco *man-in-the-middle*, *spacciandosi* per l'IP richiesto.

8.5 IPsec

Simile al protocollo *TLS* (che lavora sopra al livello di trasporto, per cui inutile in caso di attacci su livelli inferiori rispetto a quest'ultimo), che lavora a livello di rete. L'obiettivo è cifrare il *payload* (un segmento *TCP*/U*DP*, un messaggio *ICMP*/*OSPF*, ...): per motivi di sicurezza spesso si usano reti private, motivo per cui diventa necessaria una *VPN* (o *Virtual Private Network*), per mettere in comunicazione reti private attraverso internet, mantenendo tutto quello che passa per internet cifrato e separato logicamente dal traffico comune. Le *VPN* quindi stabiliscono reti private virtuali, per racchiudere n non fisicamente - e al contempo privatamente - collegate.
IPsec fornisce i seguenti servizi:

- integrità dei dati;
- autenticazione;
- prevenzione ad attacchi di tipo *replay*[3];
- confidenzialità.

Può lavorare - in base alla decisione presa dall'amministratore dell'infrastruttura - con due protocolli:

- *AH* (*Authentication Header*): fornisce garanzie su integrità e autenticazione;
- *ESP* (*Encapsulation Security Protocol*): fornisce garanzie su integrità, autenticazione e confidenzialità.

[3]Nell'ambito della sicurezza informatica il replay-attack è una forma di attacco di rete che consiste nell'impossessarsi di una credenziale di autenticazione comunicata da un host ad un altro, e riproporla successivamente simulando l'identità dell'emittente. In genere l'azione viene compiuta da un attaccante che s'interpone tra i due lati comunicanti.

e due modalità:

- Modalità *transport*: il processo di rendere sicuri i pacchetti è fatto direttamente dagli host delle reti;
- Modalità *tunneling*: il processo è gestito unicamente dagli *edge router*.

Attualmente la combinazione migliore e più diffusa è la *ESP-tunneling*.

Per ciascun host che entri in comunicazione con un altro della seconda (o n-esima) rete appartenente alla *VPN*, viene stabilita una *Security Association* (*SA*) un canale di comunicazione privato a questi due, che contiene:

- identificativo (da 32 bit);
- interfaccia *SA* di origine (*edge router* di origine);
- interfaccia *SA* di destinazione (*edge router* di destinazione);
- tipo di cifratura usata (e.g. *3DES* con *CBC*);
- chiave di cifratura;
- tipo di algoritmo di integrità (e.g. *HMAC* con *MD5*);
- chiave di autenticazione.

Tutte le *SA* sono memorizzate in un database, chiamato *Security Association Database* (*SAD*).

Nella modalità di *tunneling* con *ESP*, viene gestito un datagramma, composto da diversi passaggi: partendo dal messaggio originario in chiaro, seguito dal *padding* del messaggio (necessario a saturare lo spazio in bit dedicato al messaggio), viene cifrato, e poi viene aggiunto in testa l'header *ESP*. Su tutta questa sequenza viene applicato un primo protocollo di autenticazione, che prende il nome di *enchilada*. A questa ultima sequenza viene aggiunto, infine, l'IP di destinazione e applicato il protocollo di autenticazione relativo a tutto il datagramma.
Il *Security Policy Database* (*SPD*) gestisce invece un insieme di regole generali che stabiliscono che reagire a tutti i datagrammi uscenti dalla rete: come/se elaborarli.
Per la generazione delle *SA* automatica arriva in aiuto il protocollo *IKE* (o *Internet Key Exchange*), sfruttabile tramite *PKI* e/o *PSK*:

1. *IKE* stabilisce una *SA* bidirezionale;
2. *ISAKMP* viene utilizzato per negoziare la tupla di connessioni speculari delle *SA*.

8.6 Sicurezza nelle reti Wireless

8.6.1 WEP

Primo protocollo noto sulla sicurezza wireless è il *WEP* (*Wired Equivalent Protocol*), nato nel tentativo di raggiungere la sicurezza come nelle reti via capo, ottenendo:

- confidenzialità;
- autorizzazione dell'end-host;
- integrità.

Inoltre occorreva avere un protocollo non solo efficiente, ma che potesse allo stesso tempo cifrare ogni pacchetto trasmesso indipendentemente l'uno dall'altro (anche se uno dei precedenti fosse andato perso, a differenza del *Cipher Block Chaining*, *CBC*, in cui l'output della cifratura del pacchetto precedente era preso come input per quella del successivo). Usava un *cifrario a flusso*: cioè una funzione che prendesse in input una chiave - costante - e un *initialization vector*, che potesse dare uno stream di chiavi utile per cifrare (facendo lo *XOR* tra stream e messaggio da cifrare), ma soprattutto per decifrare (facendo lo *XOR* tra stream e messaggio cifrato).
Tra i problemi principali troviamo che il campo relativo all'*initialization vector* è troppo piccolo per evitare che si ripeti dopo un quantitativo di tempo ridotto. Questo rende il *WEP* quasi totalmente insicuro, perdendo integrità e confidenzialità.

8.6.2 802.11i

Protocollo molto più sicuro rispetto al *WEP*, che mette a disposizione numerosi tipi di cifratura (e.g. *WPA*, *WPA2*) e utilizza un server separato per l'autenticazione, rispetto a quello utilizzato per l'accesso. Mentre *WPA2* è molto sicuro, è stata introdotta una funziona che potrebbe facilmente *disarcionarlo*: *WPS*.
WPS è un protocollo per facilitare la connessione per l'estensione del segnale: che sostanzialmente permette la decifratura per parti, così da dare un eventuale OK, se è stata decifrata correttamente la prima parte. Negli ultimi anni si è pensato di mettere un numero di tentativi massimi, al termine dei quali viene sganciato il richiedente.
802.11i ha 4 fasi di operazione che sostanzialmente permettono di generare n per n client connessi: questa funzione prende il nome di *EAP* (*Extensible Authentication Protocol*).

8.7 Firewall

I firewall si occupano di gestire delle policy volte all'autorizzazione di utilizzo di determinati servizi, attraverso la generazione di regole che lavorano su a livello di porte e servizi per *TCP*, di tipologia di messaggio *ICMP* e di bit *TCP SYN* e *ACK*.
Un firewall può implementare uno *stateless packet-filter*, cioè basato unicamente su indirizzo e porta, o anche *stateful packet-filter*, che non solo ammette le possibilità della versione *staless*, ma anche quella di monitorare e gestire lo stato delle connessioni *TCP* già aperte e in corso.

8.7.1 IDS

Gli *Intrusion Detectiom System* può fare un molteplice attività:

- ispezione dei pacchetti: controlla il contenuto di ciascun pacchetto transitato, istruito con le *signature* prelevate da database di *virus*, o attacchi noti;
- esaminare la correlazione tra diversi pacchetti, per evitare si possa fare uno scan delle porte, un mapping della rete o un attacco *DoS*.

8.8 Tor

Tor è un servizio di comunicazione anonimo distribuito che usa un livello indipendente di rete che permette alle persone di migliorare la loro privacy su Internet. Si usa *tor* per mantenere i siti web liberi da meccanismi di *tracking*.
Essendo una rete, è formato da un numero decisamente elevato di host che funzionano da router - gli *onion router* (o *OR*), che dirigono il traffico - ma soprattutto da un *onion proxy* (o *OP*), installato sul client utilizzato per navigare sul *deep web*.
Tutte le informazioni che passano dal client ai vari nodi della rete vengono cifrate: l'unico *hop* che rimane in chiaro è quello che lega l'ultimo nodo che lega il primo client al secondo da raggiungere: sostanziamente, Alice - che cerca di mandare un messaggio a Tor - invierà delle informazioni; queste informazioni passeranno per un numero n di *OR*, e ciascuno di essi applicherà una cifratura sul pacchetto ricevuto. Toccherà decifrare all'$n$esimo *OR* - l'*exit router* -, che consegnerà le informazioni a Bob.
Il metodo di cifratura utilizzato da *Tor* è *TLS*.